Model Selection

Model Selection | Validation

أكاديمية طويق
Tuwaiq Academy

Concept of Validation:

- An important decision when developing any machine learning model is **how to evaluate its final performance.**
    - To get an **unbiased estimate** of the model's performance, we need to **evaluate it on the data we didn't use for training.**

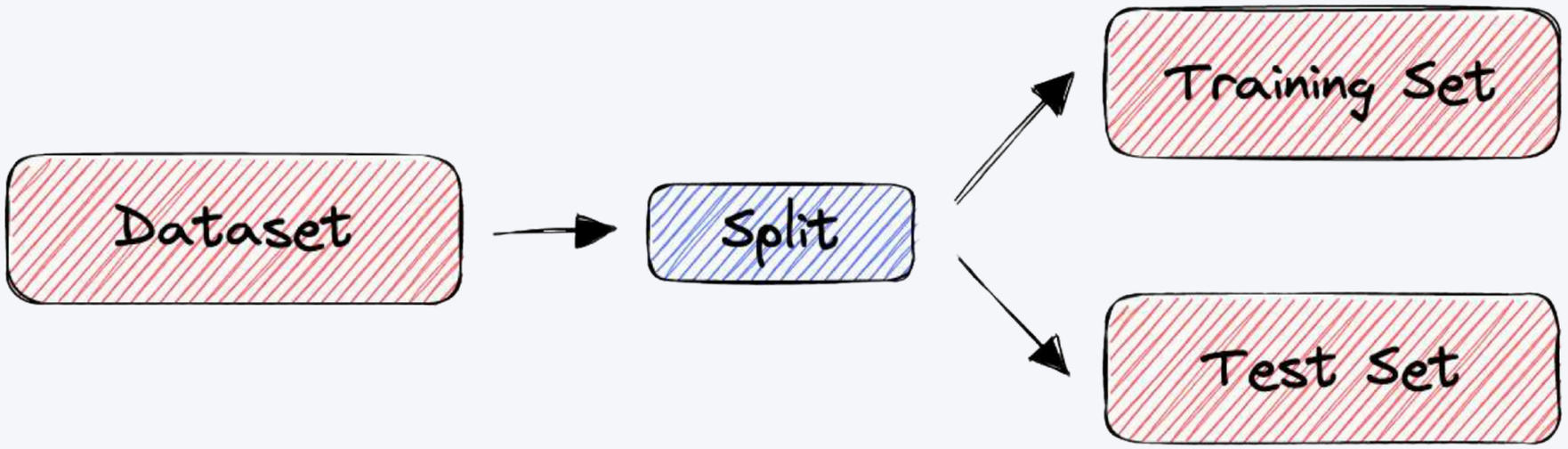
- The simplest way to split the data is to use the **train-test split** method. It randomly partitions the dataset into two subsets (called training and test sets) so that the predefined percentage of the entire dataset is in the training set.

- Then, we train our machine learning model on the training set and evaluate its performance on the test set.
    - In this way, we are always sure that the samples used for training are not used for evaluation and vice versa

Model Selection | Validation

Train-Test Split

- Dividing a dataset in to two different complementary subsets. Then, use one subset for training and another subset for testing. The testing subset is never getting trained over here.



Model Selection | Validation

أكاديمية طويق
Tuwaiq Academy

Cross Validation

- The train-split method has certain limitations. When the dataset is small, the method is prone to **high variance.**

- Due to the random partition, the results can be entirely different for different test sets. **Why?**

    - Because in some partitions, samples that are easy to classify get into the test set, while in others, the test set receives the ‘difficult’ ones.

- To deal with this issue, we use **cross-validation to evaluate the performance of a machine learning model.**
    - In **cross-validation**, we don’t divide the dataset into training and test sets **only once.**

- Instead, we ***repeatedly partition the dataset into smaller groups and then average the performance in each group. That way, we reduce the impact of partition randomness on the results.***

- Many cross-validation techniques define different ways to divide the dataset at hand. We’ll focus on the two most frequently used: T**he k-fold & The leave-one-out methods.**

Model Selection | Validation

أكاديمية طويق
Tuwaiq Academy

K-fold cross-validation

Dividing a dataset into k number of subsets.

→

In one epoch, use k-1 subsets of data for training and use the remaining dataset for testing.

→

Like this, for every epoch testing dataset will be different, but it will be out of those k subsets of data. This is also called as k-fold cross validation .

Example

For example, let's suppose that we have a dataset $S = \{x_1, x_2, x_3, x_4, x_5, x_6\}$ containing 6 samples and that we want to perform a 3-fold cross-validation.

First, we divide S into 3 subsets randomly. For instance:

$S_1 = \{x_1, x_2\}$
$S_2 = \{x_3, x_4\}$
$S_3 = \{x_5, x_6\}$

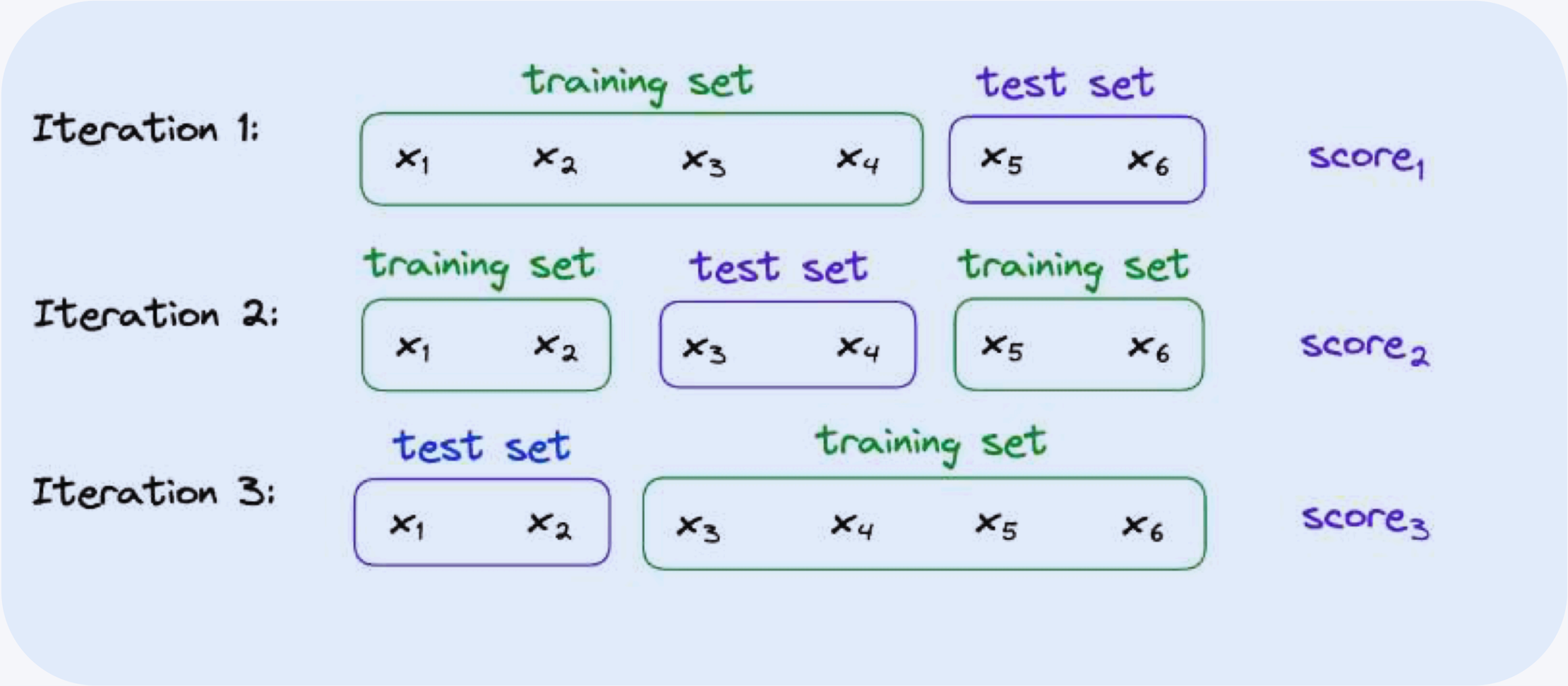
Then, we train and evaluate our machine-learning model 3 times. Each time, two subsets form the training set, while the remaining one acts as the test set. In our example:

Finally, the overall performance is the average of the model's performance scores on those three test sets.

Model Selection | Validation

أكاديمية طويق
Tuwaiq Academy

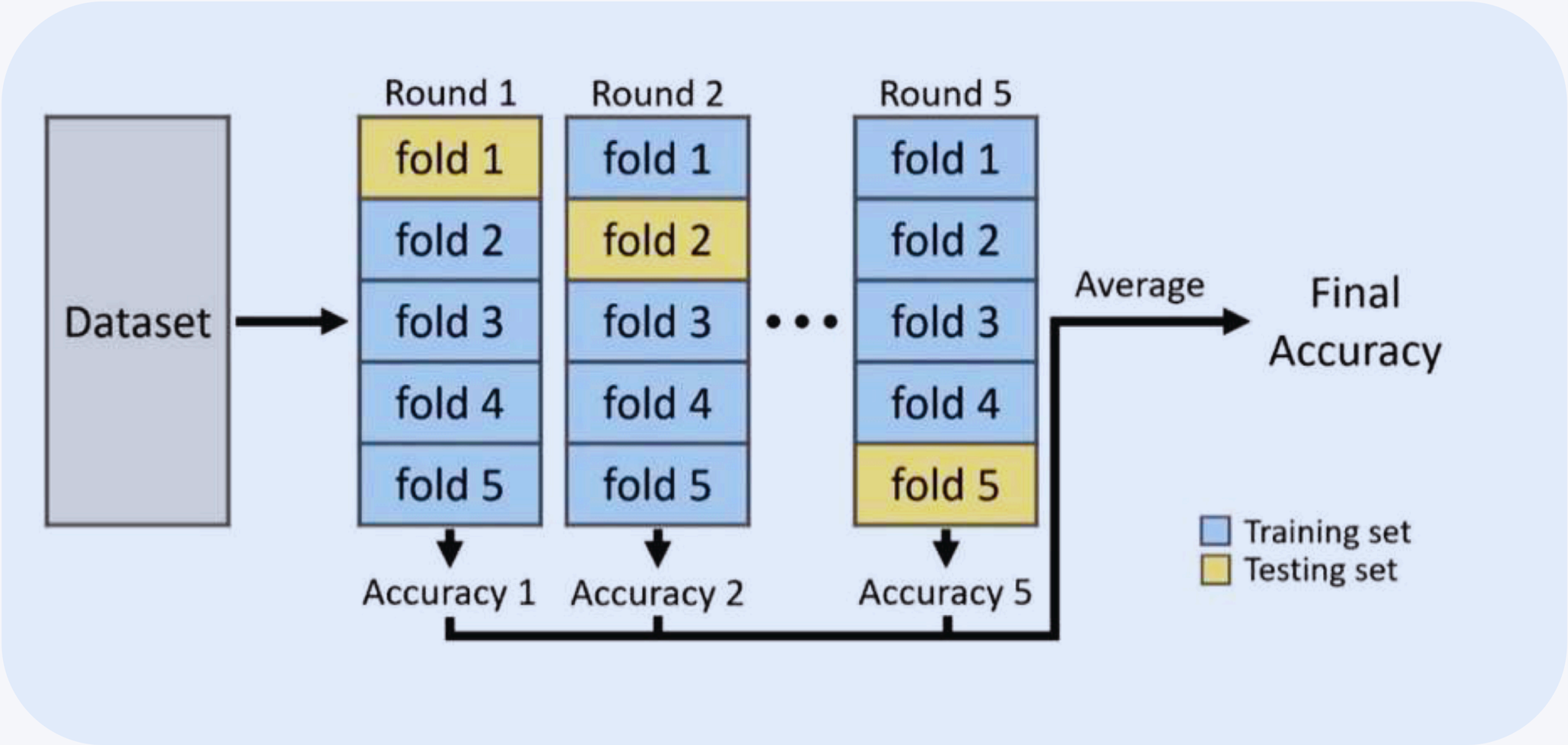
K-fold cross-validation



Model Selection | Validation

أكاديمية طويق
Tuwaiq Academy

5-fold cross validation



Model Selection | Validation

Leave-One-Out Cross-Validation

In the leave-one-out (LOO) cross-validation, we train our machine-learning model n times where n is to our dataset's size. **Each time, only one sample is used as a test set while the rest are used to train our model.**

We'll show that **LOO is an extreme case of k-fold where $\mathbf{k = n}$.** If we apply LOO to the previous example, we'll have 6 test subsets:

$$S_1 = \{x_1\}$$
$$S_2 = \{x_2\}$$
$$S_3 = \{x_3\}$$
$$S_4 = \{x_4\}$$
$$S_5 = \{x_5\}$$
$$S_6 = \{x_6\}$$

Iterating over them, we use $S \setminus S_i$ as the training data in iteration $i = 1, 2, \ldots, 6$, and evaluate the model on S_i.

The final performance estimate is the average of the six individual scores:

$$\text{overall score} = \frac{score_1 + score_2 + score_3 + score_4 + score_5 + score_6}{6}$$

Model Selection | Validation

أكاديمية طويق
Tuwaiq Academy

Leave-One-Out Cross-Validation

Model Selection | Validation

K-fold vs LOO

- When the size is **small**, LOO is more appropriate since it will use more training samples in each iteration.
  - That will enable our model to learn better representations.

- Coversely, we use **K-fold method** to train a model on large dataset since LOO trains n models, one per sample in the data.

- When our dataset contains a lot of samples, training so many models will take too long. So, the k-fold cross-validation is more appropriate.

- Also, in a large dataset, it is sufficient to use less than n folds since the test folds are large enough for the estimates to be sufficiently precise.

Bias & Variance Trade-off

Bias & Variance Trade-off

Bias and Variance

- **Bias:** It is the amount by which Machine Learning (ML) model predictions **differ from the actual value of the target.**

$$e = y_{actual} - y_{pred}$$

Where **e=Bias Error**, **yactual** = Actual or Target Output and **ypred**= Predicted Output.

- **Variance**: It is the amount by which the ML model prediction **would change if we estimate it using different training datasets.**

Bias & Variance Trade-off

Bias and Variance

- Suppose e1, e2, and e3 are the **bias errors** of the model with *three different training datasets.*

- **Average Bias Error** = b = (e1 + e2 + e3) / 3

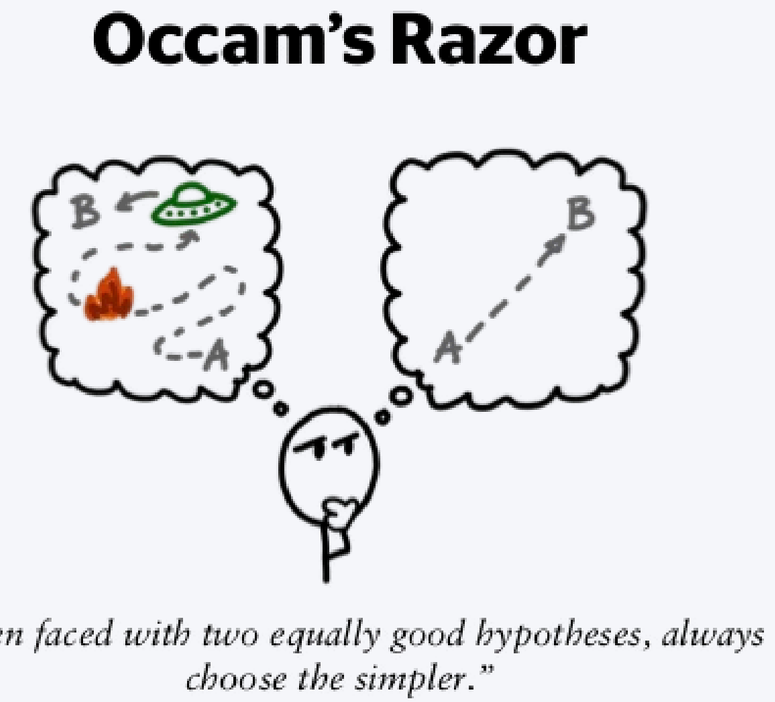
- **Average Variance Error** = [(e1 - b)2 + (e2 - b)2 + (e3 - b)2] / 3

- **Total Error** = Bias + Variance

Bias & Variance Trade-off

Occam’s Razor Principle

- Construct the simplest ML model which gives the acceptable accuracy on training datasets and don’t complicate the model to over fit the training dataset.



Bias & Variance Trade-off

Under fitting and Overfitting

- **Under fitting:** The ML model with the high bias pays very little attention to the training dataset and leads to high error on training as well as testing datasets.

High bias tends to under fitting

- **Over fitting:** The model with high variance pays a lot of attention to the training dataset and does not generalize the unseen data.

High variance tends to over fitting

Bias & Variance Trade-off

Under fitting and Overfitting

Low Bias and Low Variance leads to Ideal ML model with acceptable performance.

- Linear Regression, Logistic Regression, and Linear Discriminant Analysis are **High Bias ML algorithms**
- Decision Tree, Support Vector Machine, and K-Nearest Neighbor are **High Variance ML algorithms.**

Bias & Variance Trade-off

أكاديمية طويق
Tuwaiq Academy

Under fitting and Overfitting

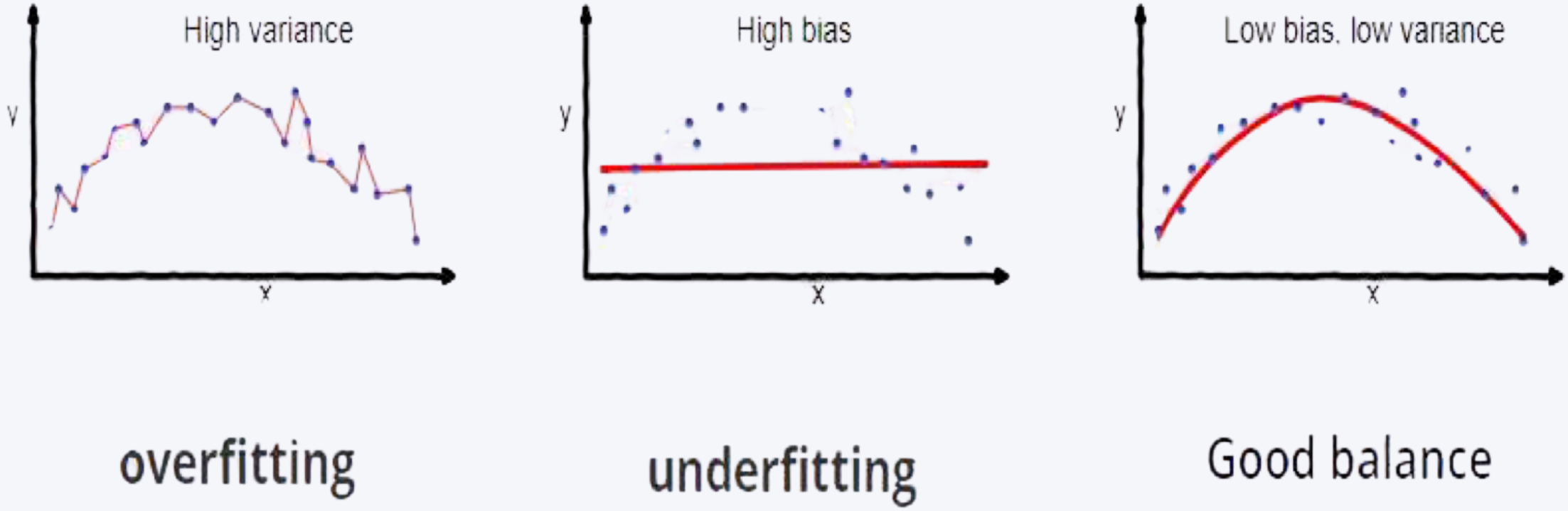


Figure 1 shows that over fit model covers all training samples where as under fit model covers only very few samples. Good balance model covers the samples with acceptable accuracy.

Bias & Variance Trade-off

أكاديمية طويق
Tuwaiq Academy

Under fitting and Overfitting

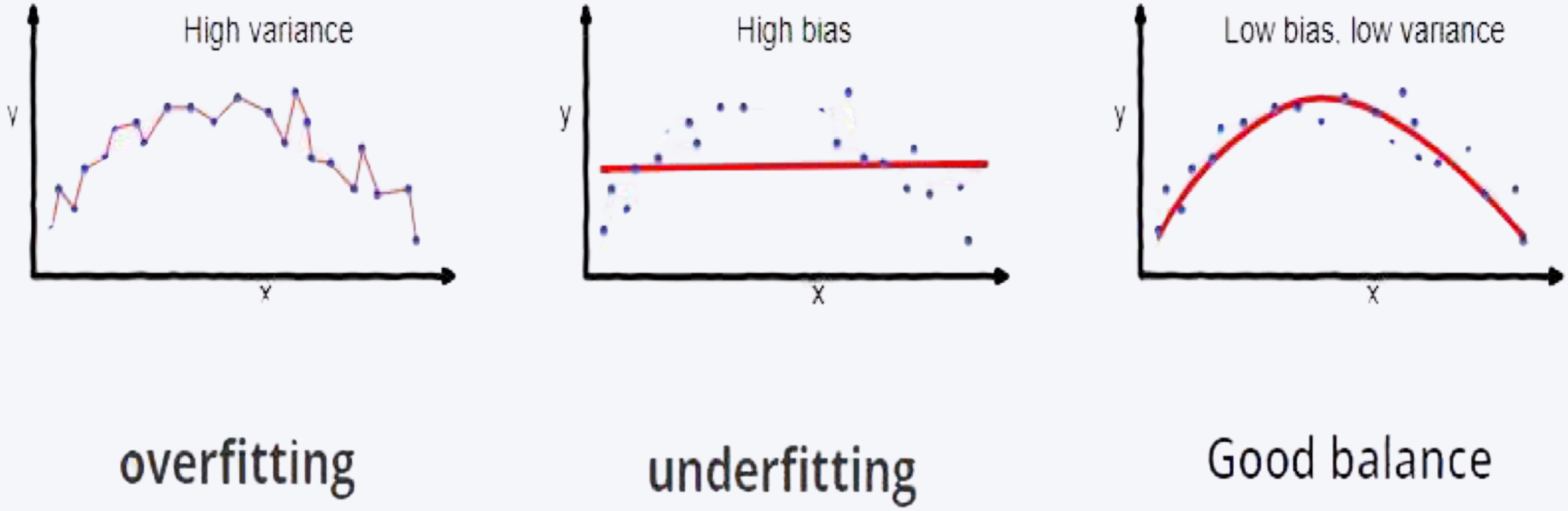


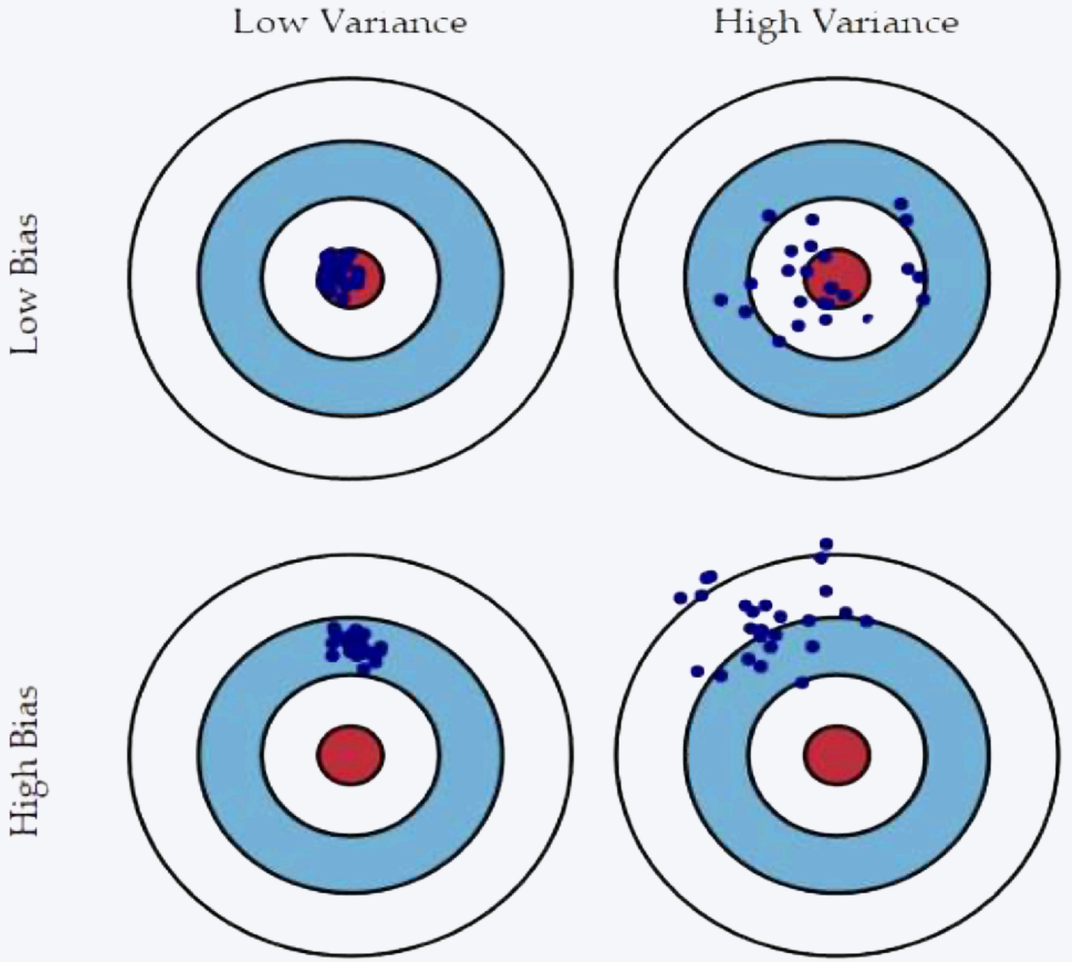
Figure 1 shows that over fit model covers all training samples where as under fit model covers only very few samples. Good balance model covers the samples with acceptable accuracy.

Bias & Variance Trade-off

Bull’s Eye for Bias and Variance Tradeoff

Figure 2 shows Bull’s Eye for Bias and Variance tradeoff.

- High Bias and Low Variance leads to Under fitting.
- Low Bias and High Variance leads to Over fitting.
- Low Bias and Low Variance leads to Ideal Model or Good Model.



أكاديمية طويق
Tuwaiq Academy

Hands-On Code

Implementing K-Fold CV in Python

GRID SEARCH

GRID SEARCH

أكاديمية طويق
Tuwaiq Academy

Concept of Grid Search

- **Grid search is a popular hyperparameter tuning technique** used in machine learning to find the optimal hyperparameters for a model.

- **Hyperparameters** are parameters that are set by the user before training the model and cannot be learned by the model during training.

GRID SEARCH

HOW TO USE GRID SEARCH

- **The grid search algorithm** involves specifying a range of values for each hyperparameter and then evaluating the performance of the model for all possible combinations of these values.

- The **performance** of the model is usually measured by a scoring metric such as **accuracy or F1 score.**

- The combination of hyperparameters that results in the best performance is selected as the optimal set of hyperparameters for the model.

GRID SEARCH

ADVANTAGES OF GRID SEARCH

- **Exhaustive search:** Grid search exhaustively searches through a hyperparameter space, ensuring that the optimal set of hyperparameters is found.
- **Simple implementation:** Grid search is simple to implement and does not require any advanced optimization techniques.
- **Reproducible results:** Grid search produces reproducible results, as the same hyperparameters will always produce the same model.

GRID SEARCH

DISADVANTAGES OF GRID SEARCH

- **Computationally expensive:** Grid search can be computationally expensive, particularly when there are many hyperparameters or large datasets.

- **Limited search space:** Grid search is limited to the hyperparameters and their respective ranges specified by the user, which may not include the optimal set of hyperparameters.

GRID SEARCH

GRID SEARCH USING PYTHON

- Finding the values of the important parameters of a model **(the ones that provide the best generalization performance)** is a tricky task, but necessary for almost all models and datasets.

- Because it is such a common task, there are standard methods in scikit-learn to help you with it.

- The most commonly used method is **grid search**, which basically means trying all possible combinations of the parameters of interest.

GRID SEARCH

أكاديمية طويق
Tuwaiq Academy

Example

- Consider the case of a **kernel SVM with an RBF** (radial basis function) kernel, as implemented in the SVC class.

- There are two important parameters: the kernel bandwidth, **gamma**, and the regularization parameter, **C**.

- Say we want to try the values 0.001, 0.01, 0.1, 1, 10, and 100 for the parameter C, and the same for gamma.

	C = 0.001	C = 0.01	…	C = 10
gamma=0.001	SVC(C=0.001, gamma=0.001)	SVC(C=0.01, gamma=0.001)	…	SVC(C=10, gamma=0.001)
gamma=0.01	SVC(C=0.001, gamma=0.01)	SVC(C=0.01, gamma=0.01)	…	SVC(C=10, gamma=0.01)
…	…	…	…	…
gamma=100	SVC(C=0.001, gamma=100)	SVC(C=0.01, gamma=100)	…	SVC(C=10, gamma=100)

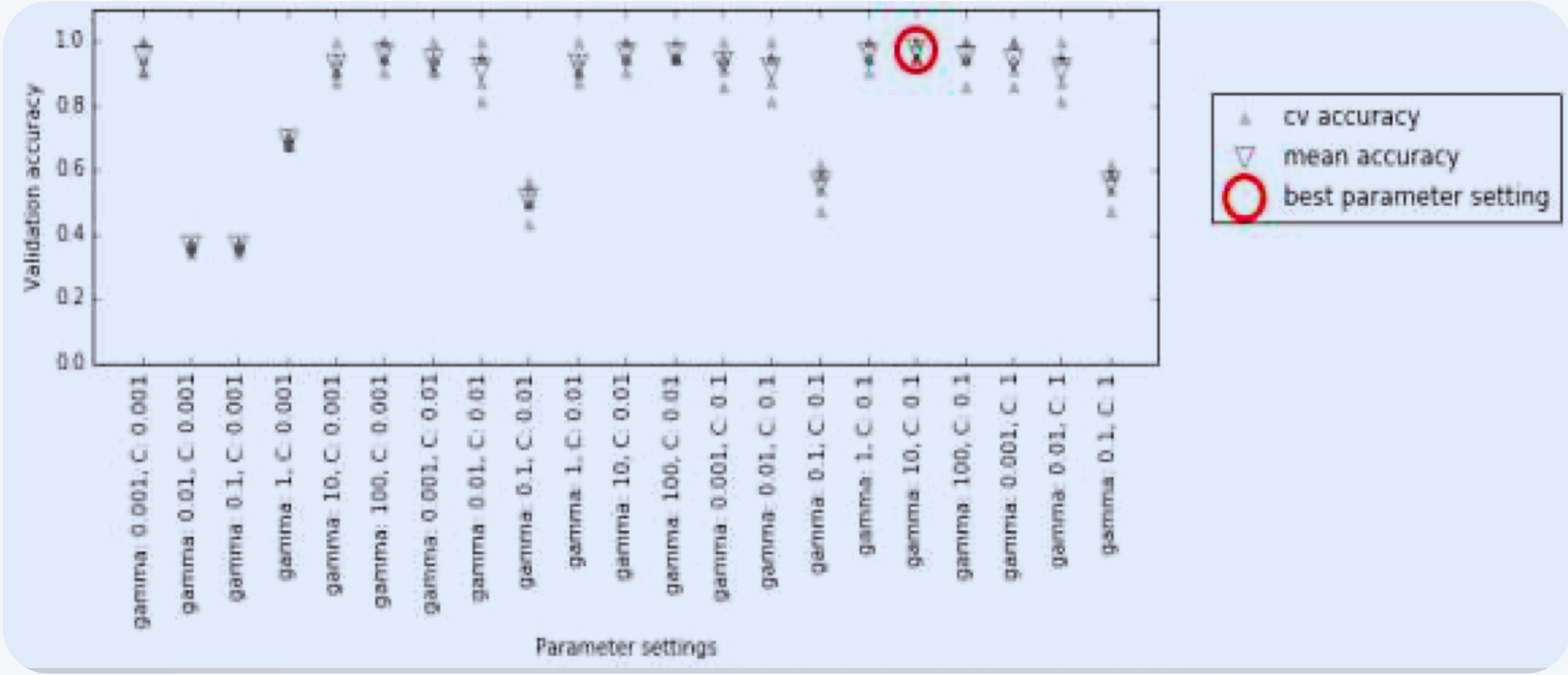
GRID SEARCH

GRID SEARCH WITH CROSS VALIDATION

- Involves creating a grid of hyperparameters and performing **k-fold cross-validation for each combination**
- The algorithm selects the hyperparameters that result in the best average cross-validation score
- Helps to avoid overfitting by evaluating the model on multiple validation sets

GRID SEARCH

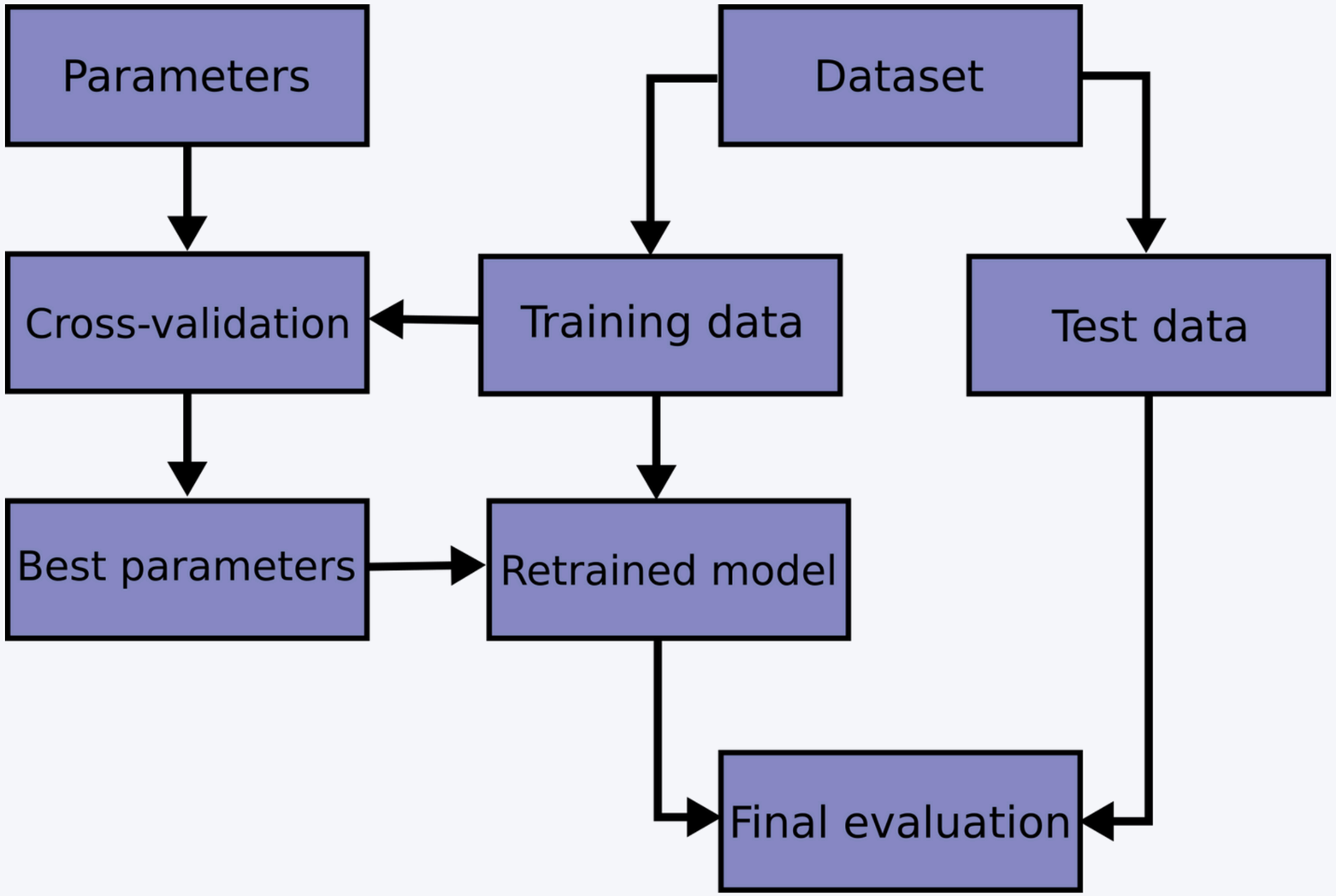
GRID SEARCH WITH CROSS VALIDATION



GRID SEARCH

أكاديمية طويق
Tuwaiq Academy

Overview of the process of parameter selection and model evaluation with GridSearchCV



Hands-On Code

GridSearchCV Implementation

Xgboost

Xgboost

أكاديمية طويق
Tuwaiq Academy

What is Xgboost?

- **XgBoost** is a powerful machine learning algorithm.
- It is designed to optimize performance and computational speed
- XGBoost (Extreme Gradient Boosting) is an optimized version of **Gradient Boosting.**
- Used for structured data tasks like **classification and regression.**

Xgboost

Why XGBoost?

- Fast and Efficient – Parallel computing & optimized execution
- Handles Missing Data – Built-in techniques for handling NaN values
- Prevents Overfitting – Regularization methods included
- Scalable – Works on large datasets efficiently

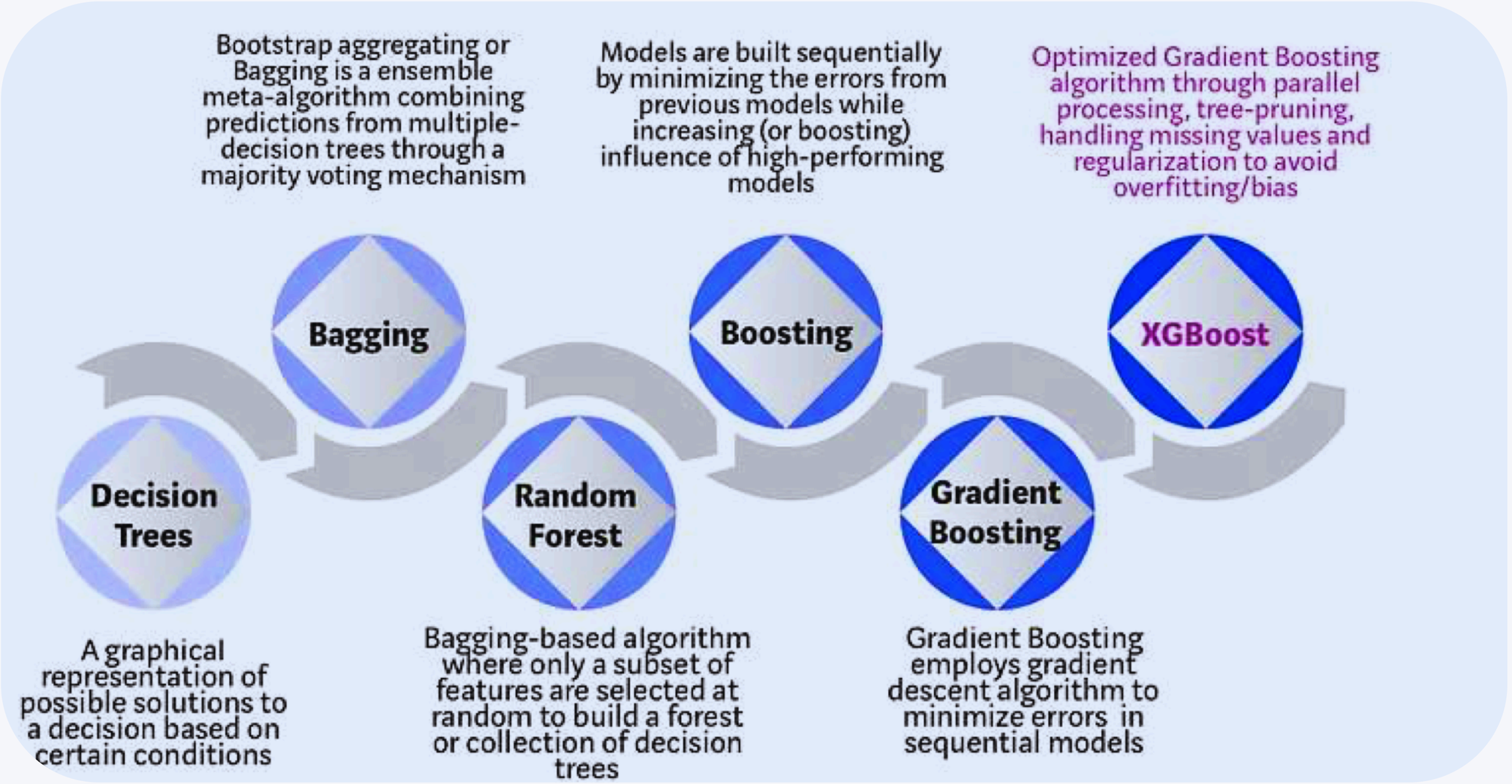
Evolution of XGBoost

From Decision Trees to **XGBoost:**

- Decision Trees – **Simple rule-based models**
- Bagging (Random Forest) – **Combines multiple trees to reduce variance**
- Boosting (Gradient Boosting) – **Sequential training to minimize errors**
- XGBoost – **Optimized Gradient Boosting with additional features**

Xgboost

Evolution of XGBoost



How Gradient Boosting Works – Key Tips

1. Starts with a Weak Model

- The process begins with a simple decision tree (often called a "weak learner").
- This model makes basic predictions but has errors (residuals).

2. Focuses on Errors

- Instead of learning from scratch, gradient boosting builds new trees to correct errors from the previous model.
- It identifies where predictions are wrong and focuses on improving those areas.

3. Uses Gradient Descent

- The algorithm minimizes the loss function by using gradient descent (step-by-step improvement).
- Each new model moves in the direction of reducing error (like walking downhill).

How Gradient Boosting Works – Key Tips

4. Adds Models Sequentially

- Each new tree is added one at a time to refine predictions.
- Models work together to create a strong final model.

5. Controls Overfitting

- Learning rate controls how much each new model adjusts.
- Tree pruning & regularization help prevent overfitting (memorizing training data).

6. Final Prediction = Sum of All Trees

The final result is a combination of all the models, leading to high accuracy and strong generalization.

Important Notes about Xgboost

Loss Function in XGBoost

- XGBoost minimizes the loss function (error) using gradient boosting.
- Common loss functions:
  - Regression: Mean Squared Error (MSE)
  - Classification: Log Loss (for binary) / Softmax Loss (for multi-class)

Loss Function in XGBoost

- XGBoost minimizes the loss function (error) using gradient boosting.
- Common loss functions:
  - Regression: Mean Squared Error (MSE)
  - Classification: Log Loss (for binary) / Softmax Loss (for multi-class)

Important Notes about Xgboost

Hyperparameters in XGBoost

Before running the code, you should understand key parameters:

- n_estimators – Number of trees (boosting rounds).
- learning_rate (eta) – Controls step size of boosting (small values prevent overfitting).
- max_depth – Limits tree depth to prevent overfitting.
- subsample – Percentage of data used in each boosting round.
- colsample_bytree – Fraction of features used per tree (like Random Forest).
- lambda & alpha – L2 and L1 regularization terms.

Handling Missing Values

- XGBoost can automatically handle missing data by learning optimal splits.
- No need to manually fill missing values before training.

Important Notes about Xgboost

Understanding XGBoost Output (Feature Importance)

- XGBoost provides feature importance scores, helping understand which features contribute the most.
- Useful for feature selection and improving model performance.

Why is XGBoost faster?

- Uses parallel computing (multi-threading) to train trees quickly.
- Optimized memory usage to handle large datasets efficiently.
- Tree pruning (depth-wise growth) improves speed & reduces overfitting.

AdaBoost

أكاديمية طويق
Tuwaiq Academy

AdaBoost (Adaptive Boosting) and XGBoost (Extreme Gradient Boosting) are both boosting algorithms that improve weak learners into strong classifiers.

Homework

Feature	AdaBoost	XGBoost
Base Model		
Optimization		
Weighting Strategy		
Loss Function		
Regularization		
Speed & Performance		
Handling Missing Data		
Tree Pruning		
Parallelization		
Use Case		

أكاديمية طويق
Tuwaiq Academy

Hands-On Code

XGBoost Implementation

The End.