Unit 5: Assessing and Comparing Learning Algorithms

Cross-validation and Resampling

Cross-validation and resampling are statistical techniques used in machine learning to evaluate a model's performance and ensure it generalizes well to new, unseen data. The primary goal is to get a more accurate and stable estimate of model performance and to avoid problems like **overfitting**, where a model learns the training data too well and performs poorly on new data.

Cross-validation and Resampling

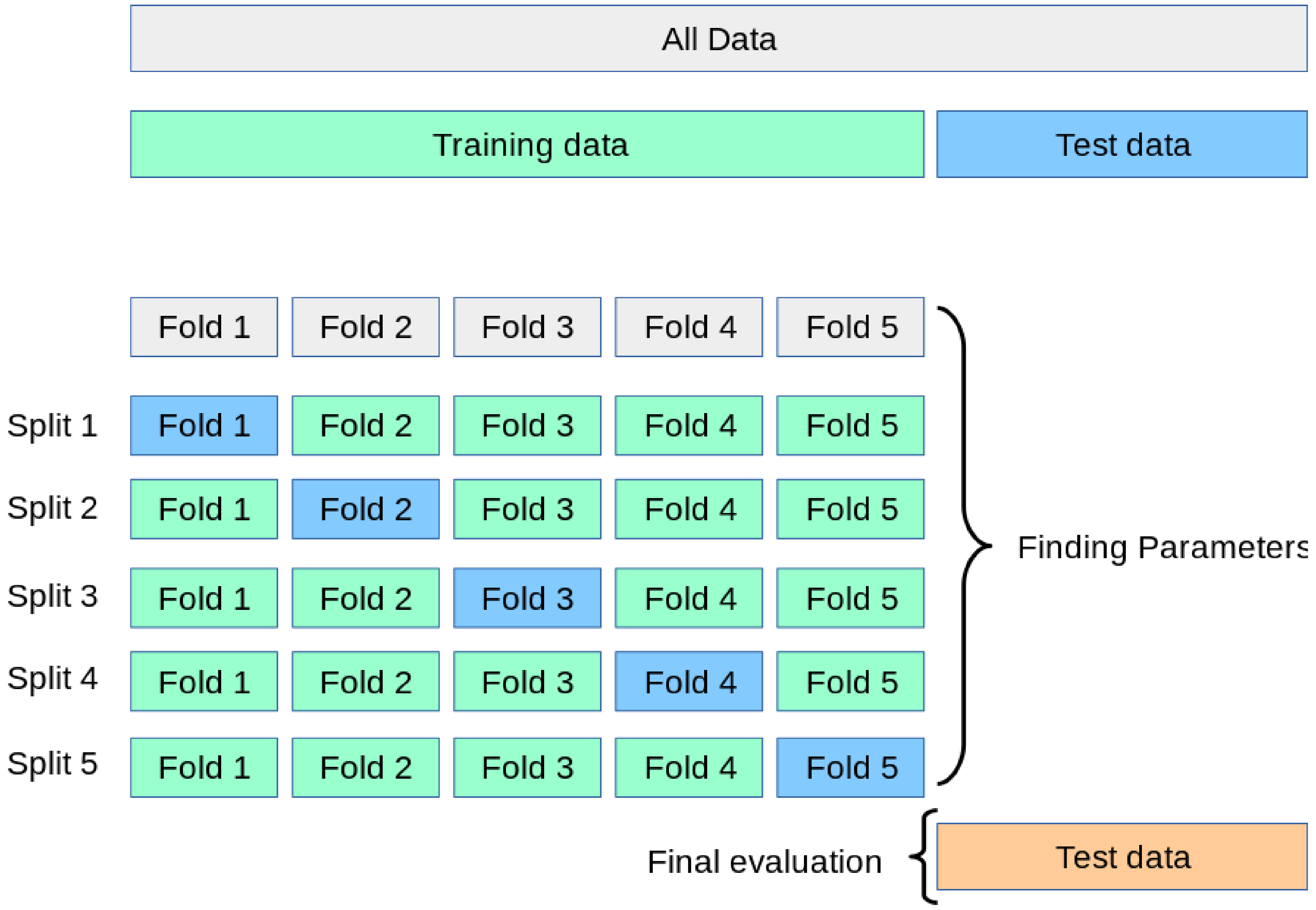
Cross-Validation

Cross-validation is a resampling procedure that involves partitioning a dataset into complementary subsets, training the model on one subset (the training set), and validating it on the other (the testing or validation set). This process is repeated multiple times to reduce variability, and the results are averaged to provide a more reliable estimate of the model's performance.

The following is the procedure deployed in almost all types of cross-validation:

- **Partition or divide** the dataset into several subsets
- At one time, keep or hold out one of the set and train the model on the remaining set
- Perform the model testing on the holdout dataset

Cross-validation and Resampling

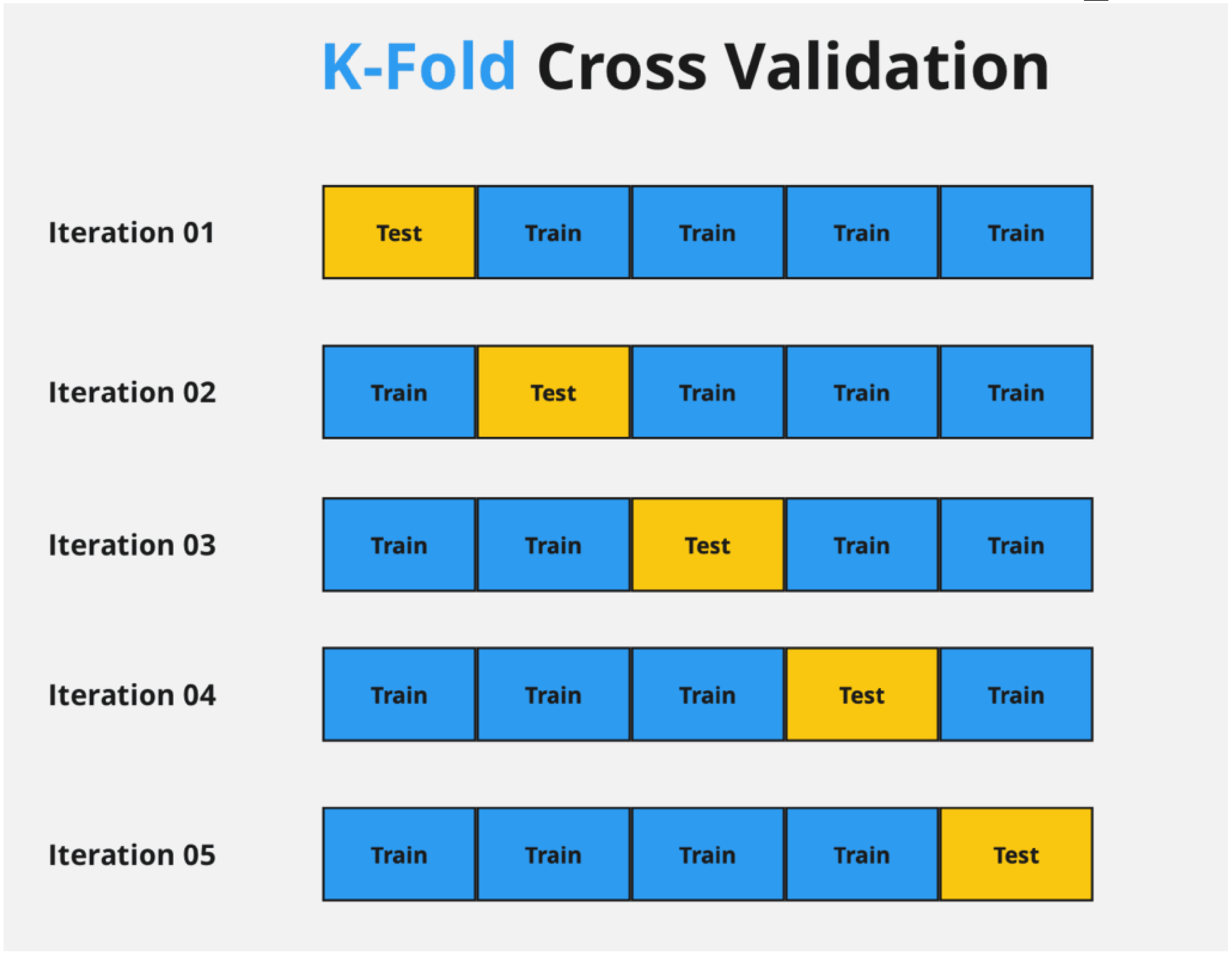


Cross-validation and Resampling

Several methods of cross-validation exist:

K-Fold Cross-Validation: This is the most common technique. The dataset is randomly split into 'k' equal-sized subsets, or "folds". The model is then trained and evaluated 'k' times. In each iteration, one fold is used as the test set, and the remaining k-1 folds are used for training. The performance scores from each of the 'k' iterations are then averaged to produce a single, more robust performance estimate. Common choices for 'k' are 5 or 10.

Cross-validation and Resampling



Cross-validation and Resampling

Let’s say K = 5 (5-Fold Cross-Validation):
Divide your dataset into 5 equal parts (folds).
Iteration 1: Train on folds 2–5, test on fold 1
Iteration 2: Train on folds 1, 3–5, test on fold 2
Iteration 3: Train on folds 1–2, 4–5, test on fold 3
Iteration 4: Train on folds 1–3, 5, test on fold 4
Iteration 5: Train on folds 1–4, test on fold 5
After all 5 iterations, you’ll have 5 test results (accuracies, RMSE, etc.).
Average those results → gives the **final model performance**.

Cross-validation and Resampling

Example

Let’s say you have 100 samples and use **K = 5**

Each fold = 20 samples

If the accuracies for each fold are:

80%, 82%, 78%, 81%, 79%

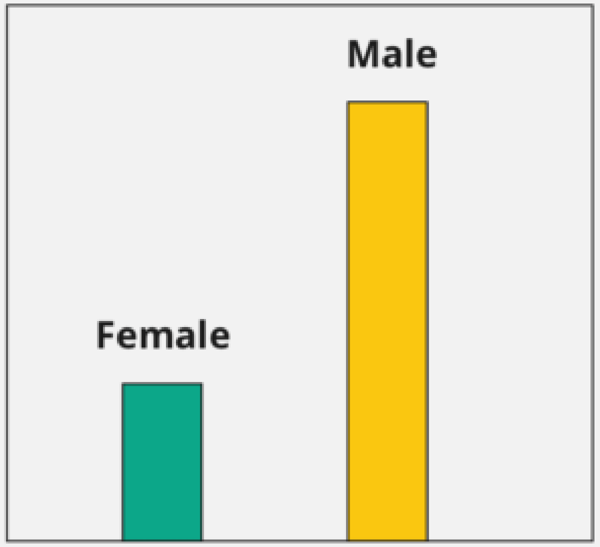
Then the **final accuracy** = (80 + 82 + 78 + 81 + 79) / 5 = **80%**

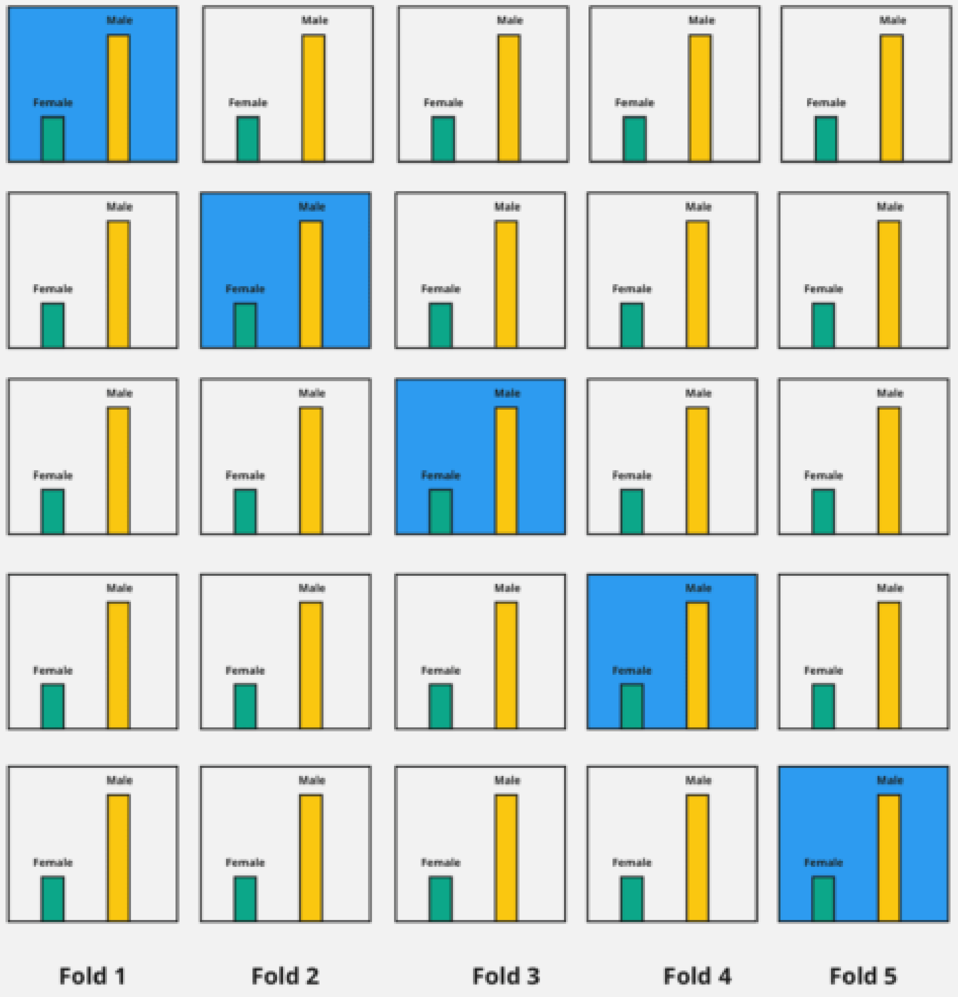
Cross-validation and Resampling

Stratified K-Fold Cross-Validation: This is a variation of k-fold cross-validation used for classification problems, especially with imbalanced datasets. It ensures that each fold has the same proportion of class labels as the original dataset. This prevents situations where a fold might, by chance, contain no samples from a minority class, leading to a biased evaluation.

Stratified K-Fold Cross Validation

Target Class Distribution





Stratified K-Fold Cross-Validation:

How It Works

Let's say you're doing **binary classification** (e.g., Spam / Not Spam) with **100 samples**:

- 70 samples = Class 0
- 30 samples = Class 1

➢ Now if you use **regular K-Fold (K=5)**, some folds might accidentally have very few Class 1 samples — which leads to *biased training/testing*.

But with **Stratified K-Fold**:

- Each of the 5 folds will have **14 samples of Class 1** and **6 samples of Class 0** (same ratio: 70:30).
- So, every fold is **representative** of the whole dataset.

Stratified K-Fold Cross-Validation:

Example (K=5)

1. Dataset divided into 5 equal parts
2. Each fold keeps the same proportion of classes
3. Perform 5 iterations:
    - Train on 4 folds, test on the remaining 1
4. Record accuracy for each iteration
5. Average them → gives **final performance.**

- Always use Stratified K-Fold for classification problems (especially when class imbalance exists).
- For regression, use regular K-Fold (since class labels don’t exist).

Cross-validation and Resampling

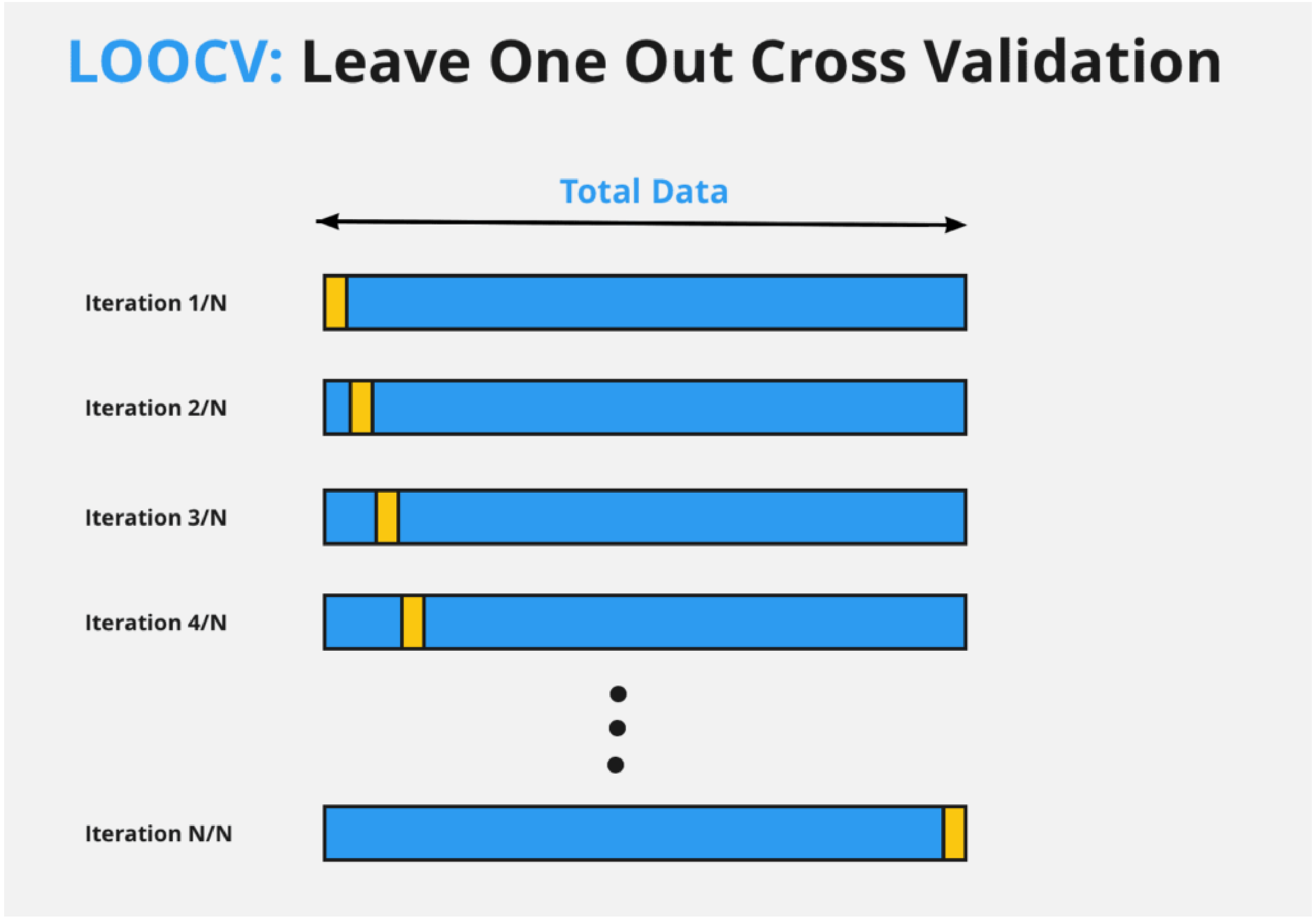
Leave-One-Out Cross-Validation (LOOCV): This is an exhaustive form of k-fold cross-validation where 'k' is equal to the number of data points in the dataset. In each iteration, the model is trained on all data points except for one, which is used for testing. This is repeated for every data point. While LOOCV provides a low-bias estimate, it can be computationally very expensive for large datasets.

Makes full use of data — almost all samples are used for training each time.

Unbiased estimate of model performance

Useful when dataset is very small

Cross-validation and Resampling



Leave-One-Out Cross-Validation

Ex:

Let’s say your dataset has **5 samples**:

[A, B, C, D, E]

You’ll do **5 iterations:**

Iteration	Training Data	Test Data
1	B, C, D, E	A
2	A, C, D, E	B
3	A, B, D, E	C
4	A, B, C, E	D
5	A, B, C, D	E

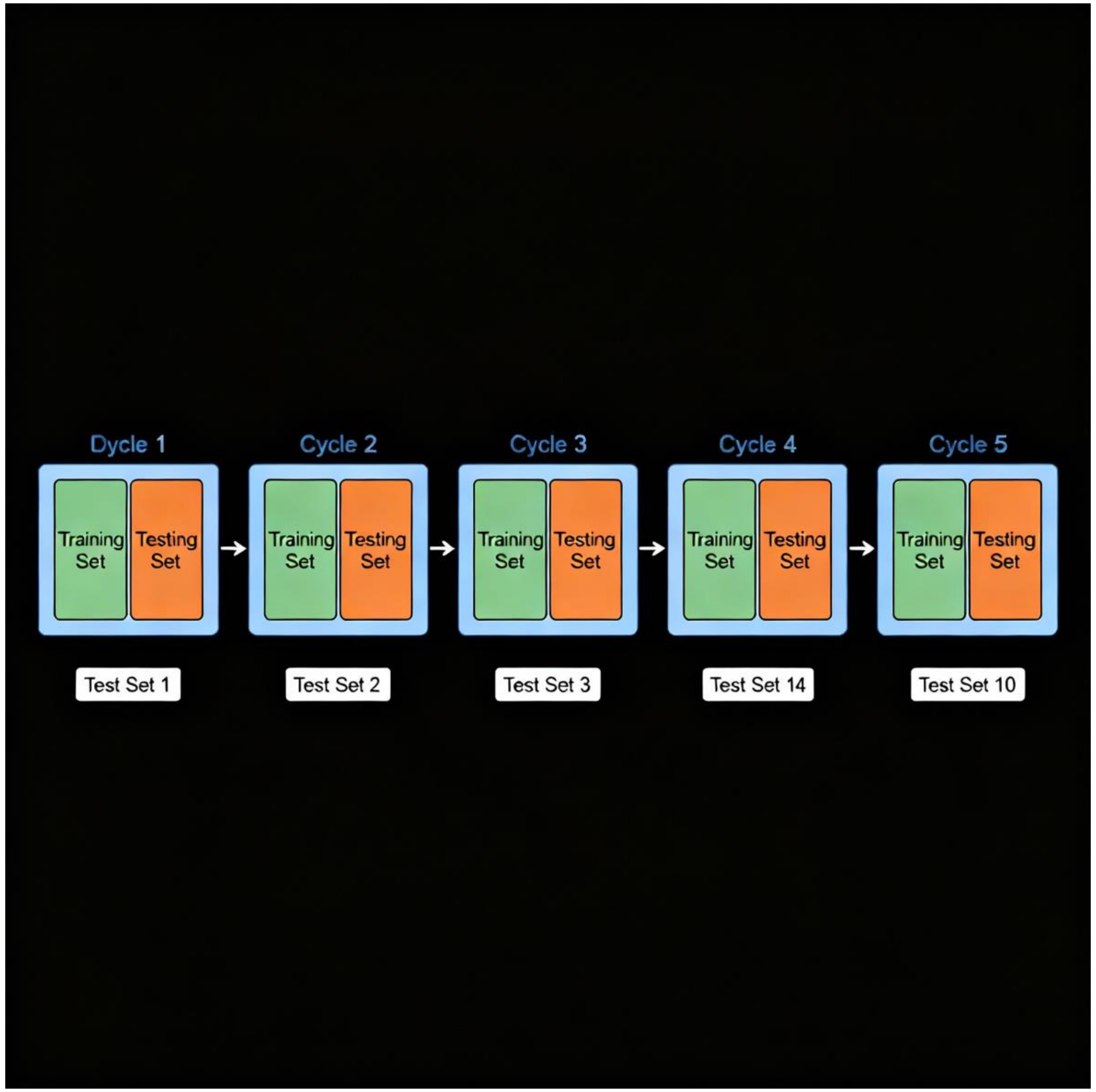
After all 5 runs:

- **You’ll have 5 accuracy (or error) values**
- **Average them → gives your final model performance**

Cross-validation and Resampling

5x2 Cross-Validation: This a specific procedure where a 2-fold cross-validation is performed five times. In each of the five repetitions, the data is randomly split into two equal halves. The model is trained on one half and tested on the other, and then the roles are reversed. The error rates from the ten resulting test sets are used to evaluate the model. This method is often used when comparing the performance of two different algorithms.

5x2 Cross-Validation:



5x2 Cross-Validation

1.The dataset is **randomly divided into 2 equal halves** — say **A** and **B**.

2.Perform **2-fold validation**:

- **Run 1:** Train on **A**, test on **B**
- **Run 2:** Train on **B**, test on **A**

3.Compute the **error (or accuracy)** for both runs.

That completes **one repetition**.

4.Now, **repeat the entire process 5 times**, each time with a **different random split** of the data into A and B.

5.This gives you a total of **10 test results (2 × 5 = 10)**.

Finally, the **average error (or accuracy)** from all 10 runs is used as the final performance estimate.

Resampling

Resampling

Resampling is the broader category of methods that includes cross-validation. It involves repeatedly drawing samples from a dataset and refitting a model on each sample. This is done to gain additional information about the model, such as the variability of its performance estimates. Bootstrapping, another key resampling method mentioned in your syllabus, involves creating multiple datasets by sampling with replacement from the original dataset and training a model on each new set. This is often used to estimate the uncertainty of a model's performance.

Bootstrapping

Bootstrapping is a resampling technique used in statistics and machine learning to estimate the sampling distribution of a statistic. It is particularly useful for assessing the uncertainty of a model's performance, such as its accuracy or the stability of its predictions, without needing to collect new data.

How Bootstrapping Statistics Works?

Sampling with Replacement

The fundamental concept behind bootstrapping is sampling with replacement.

Here’s how the process generally works:

Start with the Original Dataset: You have one original dataset containing 'n' data points.

Create Bootstrap Samples: You generate a new dataset (a "bootstrap sample") by randomly drawing 'n' data points from the original dataset, with the key condition that each data point is put back after being selected. This means a single data point from the original set can appear multiple times in a bootstrap sample, while others might not appear at all.

How Bootstrapping Statistics Works?

Repeat Many Times: This process is repeated many times (e.g., hundreds or thousands of times) to create a large number of different bootstrap samples, each of the same size as the original dataset.

Calculate the Statistic of Interest: For each of these bootstrap samples, you calculate the statistic you are interested in. In machine learning, this could be:

- The accuracy of a classification model trained on that bootstrap sample.
- The value of a regression coefficient.
- The mean or median of a set of values.

Estimate the Distribution: You now have a large collection of calculated statistics (e.g., a list of 1,000 accuracy scores). This collection forms an empirical distribution of your statistic, from which you can estimate properties like the standard error, confidence intervals, or bias.

How Bootstrapping Statistics Works?

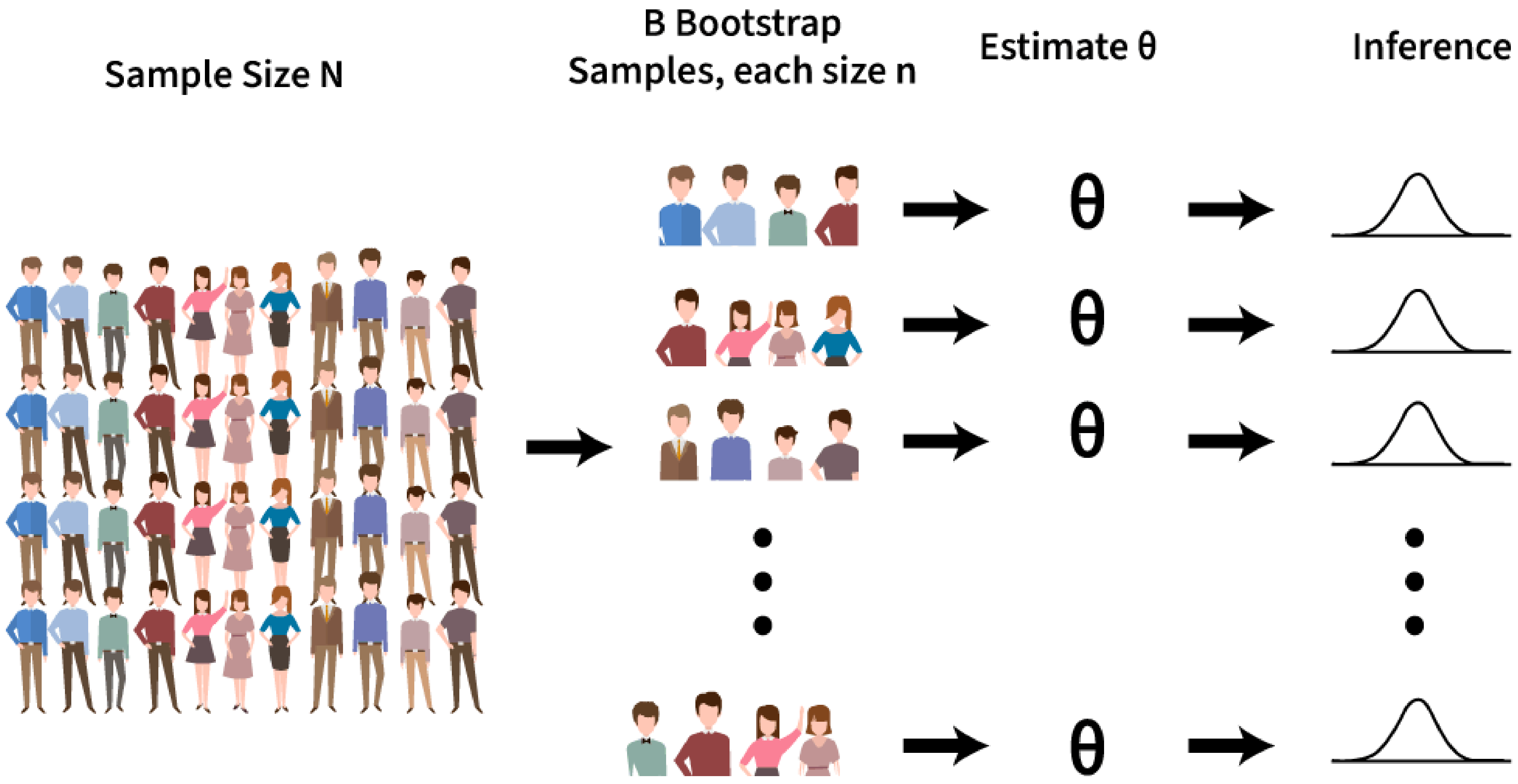
Repeat Many Times: This process is repeated many times (e.g., hundreds or thousands of times) to create a large number of different bootstrap samples, each of the same size as the original dataset.

Calculate the Statistic of Interest: For each of these bootstrap samples, you calculate the statistic you are interested in. In machine learning, this could be:

- The accuracy of a classification model trained on that bootstrap sample.
- The value of a regression coefficient.
- The mean or median of a set of values.

Estimate the Distribution: You now have a large collection of calculated statistics (e.g., a list of 1,000 accuracy scores). This collection forms an empirical distribution of your statistic, from which you can estimate properties like the standard error, confidence intervals, or bias.

Bootstrapping Statistics



Measuring Error

Measuring Error: Techniques to quantify the error of a learning algorithm. In machine learning, measuring error is the process of quantifying how inaccurate a model's predictions are compared to the actual outcomes. The specific techniques used depend on whether the model is for a regression or a classification task.

General Techniques to Quantify Error

The choice of error metric is tied to the type of prediction being made.

For Regression Problems (Predicting Continuous Values)

Mean Absolute Error (MAE): This is the average of the absolute differences between the predicted values and the actual values. It gives an idea of the magnitude of the error on average.

Mean Squared Error (MSE): This metric calculates the average of the squared differences between predicted and actual values. By squaring the errors, it penalizes larger errors more heavily than smaller ones.

Measuring Error

Measuring Error: Techniques to quantify the error of a learning algorithm. In machine learning, measuring error is the process of quantifying how inaccurate a model's predictions are compared to the actual outcomes. The specific techniques used depend on whether the model is for a regression or a classification task.

General Techniques to Quantify Error

The choice of error metric is tied to the type of prediction being made.

For Regression Problems (Predicting Continuous Values)

Mean Absolute Error (MAE): This is the average of the absolute differences between the predicted values and the actual values. It gives an idea of the magnitude of the error on average.

Mean Squared Error (MSE): This metric calculates the average of the squared differences between predicted and actual values. By squaring the errors, it penalizes larger errors more heavily than smaller ones.

Measuring Error

Root Mean Squared Error (RMSE): As the square root of the MSE, RMSE is useful because its value is in the same units as the target variable, making it easier to interpret the scale of the error.

For Classification Problems (Predicting Categories)

Error in classification is often analyzed using a Confusion Matrix, which is a table that summarizes the performance of a classifier by showing the number of correct and incorrect predictions for each class. The matrix contains four key values:

- **True Positives (TP):** The model correctly predicted the positive class.
- **True Negatives (TN):** The model correctly predicted the negative class.
- **False Positives (FP) / Type I Error:** The model incorrectly predicted the positive class.
- **False Negatives (FN) / Type II Error:** The model incorrectly predicted the negative class.

Metrics derived from the confusion matrix include Accuracy, Precision, Recall, and F1 Score.

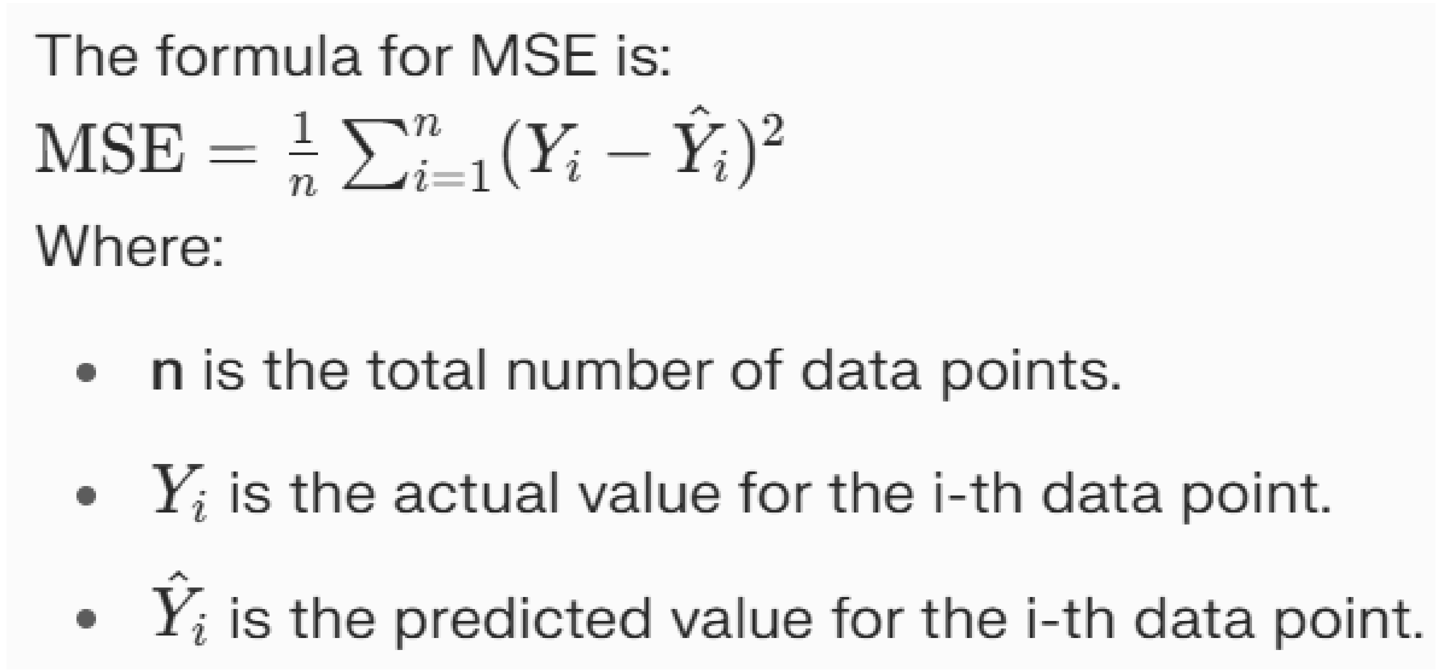
Measuring Error

Mean Squared Error (MSE) is a fundamental and widely used metric for evaluating the performance of a regression model. It quantifies the average of the squared differences between the predicted values and the actual, or true, values.

How MSE is Calculated

The calculation of MSE involves three main steps:

1. Calculate the error
2. Square each error
3. Calculate the average



The formula for MSE is:

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

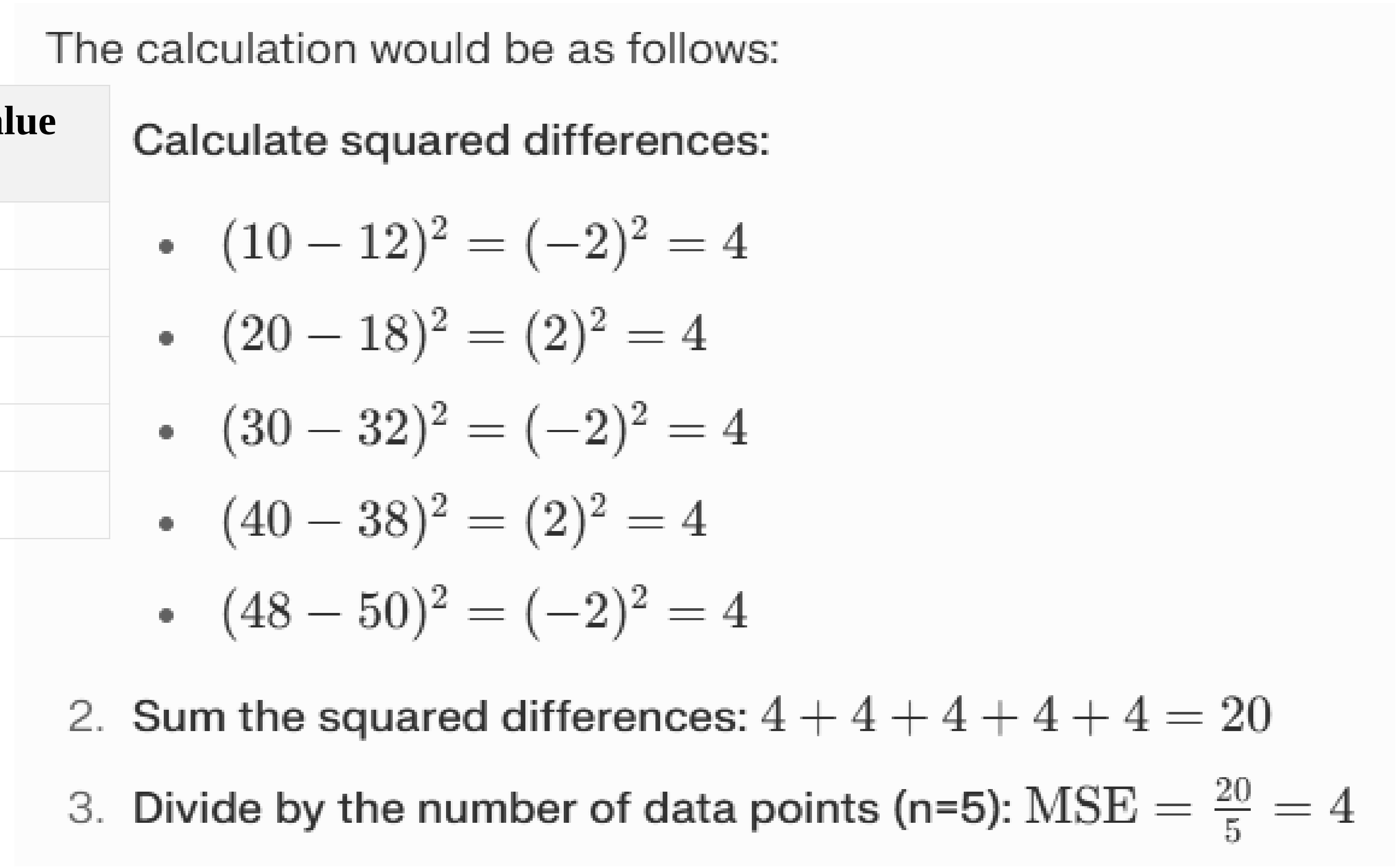
Where:

- **n** is the total number of data points.
- Y_i is the actual value for the i-th data point.
- $\hat{Y}_i$ is the predicted value for the i-th data point.

Measuring Error

Example Calculation

Data Point	Actual Value (Y)	Predicted Value (Y^)
1	10	12
2	20	18
3	30	32
4	40	38
5	48	50



The calculation would be as follows:

Calculate squared differences:

- $(10-12)^2=(-2)^2=4$
- $(20-18)^2=(2)^2=4$
- $(30-32)^2=(-2)^2=4$
- $(40-38)^2=(2)^2=4$
- $(48-50)^2=(-2)^2=4$

2. Sum the squared differences: $4+4+4+4+4=20$
3. Divide by the number of data points (n=5): $\mathrm{MSE}=\frac{20}{5}=4$

Measuring Error

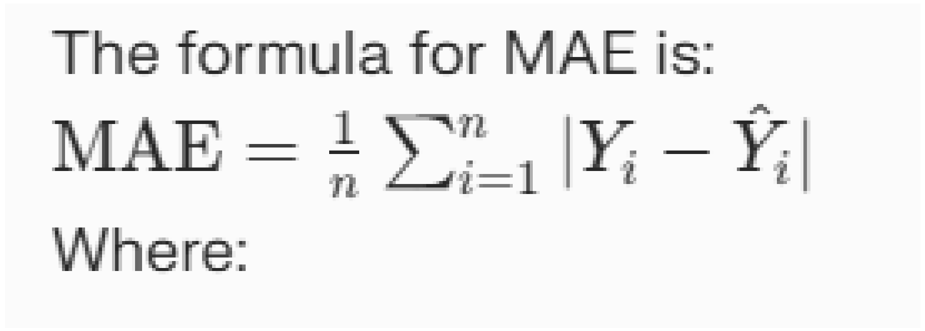
Mean Absolute Error (MAE) is a metric used to evaluate the performance of a regression model by measuring the average magnitude of the errors in a set of predictions, without considering their direction. It provides a straightforward and easily interpretable score for how accurate a model's predictions are.

How MAE is Calculated

- The calculation for MAE is direct and involves three simple steps:

1. **Calculate the error** for each data point by finding the difference between the actual value (Yi) and the predicted value (*Y*^*i*).
2. **Take the absolute value** of each error
3. **Calculate the average** of these absolute errors.

Measuring Error



The formula for MAE is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$$

Where:

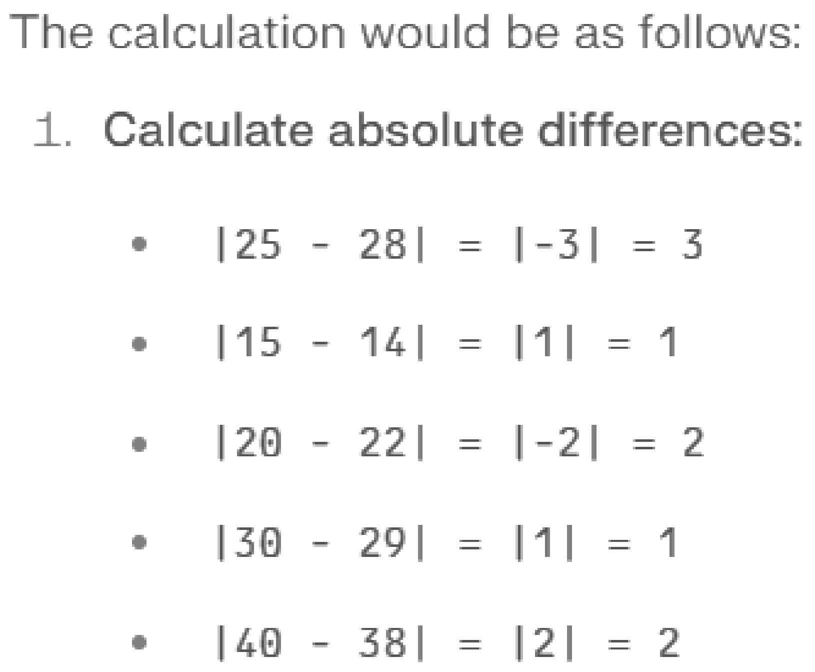
- n is the total number of data points.
- *Yi* is the actual value for the i-th data point
- Y^*i* is the predicted value for the i-th data point.

Measuring Error

Example Calculation

Using the same data as the MSE example:

Data Point	Actual Value (Y)	Predicted Value (Y^)
1	25	28
2	15	14
3	20	22
4	30	29
5	40	38



The calculation would be as follows:

1. Calculate absolute differences:

- `|25 - 28| = |-3| = 3`
- `|15 - 14| = |1| = 1`
- `|20 - 22| = |-2| = 2`
- `|30 - 29| = |1| = 1`
- `|40 - 38| = |2| = 2`

2. Sum the absolute differences: 3 + 1 + 2 + 1 + 2 = 9
3. Divide by the number of data points (n=5): MAE = 9 / 5 = 1.8

Measuring Error

Root Mean Squared Error (RMSE): As the square root of the MSE, RMSE is useful because its value is in the same units as the target variable, making it easier to interpret the scale of the error.

For Classification Problems (Predicting Categories)

Error in classification is often analyzed using a Confusion Matrix, which is a table that summarizes the performance of a classifier by showing the number of correct and incorrect predictions for each class. The matrix contains four key values:

- **True Positives (TP):** The model correctly predicted the positive class.
- **True Negatives (TN):** The model correctly predicted the negative class.
- **False Positives (FP) / Type I Error:** The model incorrectly predicted the positive class.
- **False Negatives (FN) / Type II Error:** The model incorrectly predicted the negative class.

Metrics derived from the confusion matrix include Accuracy, Precision, Recall, and F1 Score.

Measuring Error

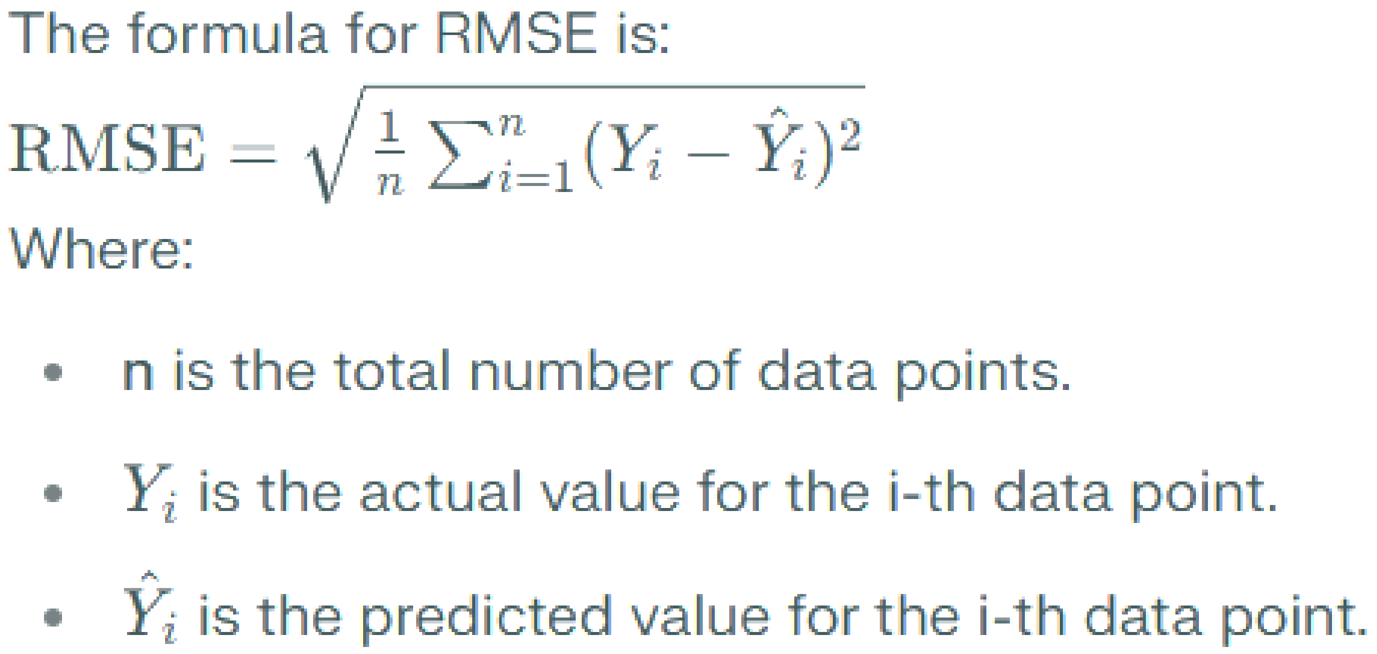
Root Mean Squared Error (RMSE) is a standard way to measure the error of a model in predicting quantitative data, making it one of the most common evaluation metrics for regression tasks. It represents the standard deviation of the prediction errors (residuals), giving an idea of how spread out these errors are.

How RMSE is Calculated

RMSE is derived directly from the Mean Squared Error (MSE). The calculation follows these steps:

- **Calculate the error** (residual) for each data point by finding the difference between the actual value (*Yi*) and the predicted value (*Y*^*i*).
- **Square each residual.**
- **Calculate the average** of these squared residuals to get the MSE.
- **Take the square root** of the MSE to get the RMSE.

Measuring Error



The formula for RMSE is:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}$$

Where:

- n is the total number of data points.
- Y_i is the actual value for the i-th data point.
- $\hat{Y}_i$ is the predicted value for the i-th data point.

Binomial Test to Assess Classification Algorithm

The Binomial Test is a specific statistical test used to evaluate a classification algorithm by determining whether its accuracy is statistically significant or if it could simply be the result of random chance.

The test frames the evaluation as a series of Bernoulli trials, where each prediction is either correct (a "success") or incorrect (a "failure").

Binomial Test to Assess Classification Algorithm

Steps Involved

1. Hypothesis Formulation:

- **Null Hypothesis (H_0):** The classifier has no skill and is equivalent to random guessing. For a balanced two-class problem, this means the probability of a correct prediction is 0.5.
- **Alternative Hypothesis (H_1):** The classifier's performance is significantly better than random guessing.

2. Data Collection: The algorithm's accuracy is observed on a test set. For instance, a model makes k correct predictions out of n total instances.

3. P-value Calculation: The binomial test calculates the probability of achieving k or more successes in n trials, assuming the null hypothesis is true. This probability is the p-value.

4. Conclusion: The p-value is compared to a significance level (alpha, typically 0.05). If the p-value is smaller than alpha, the null hypothesis is rejected, suggesting the classifier's performance is statistically significant and not just due to luck.

Comparison of Algorithms

Methods for comparing two different classification algorithms, such as To compare two different classification algorithms, it's not enough to simply look at their accuracy scores. Statistical tests are needed to determine if the observed performance difference is significant or if it could have occurred by chance.

McNemar's Test

McNemar's test is a statistical test used for comparing two classification models. It is particularly well-suited for situations where the two models are evaluated on the same dataset. The test focuses specifically on the instances where the two models' predictions disagree.

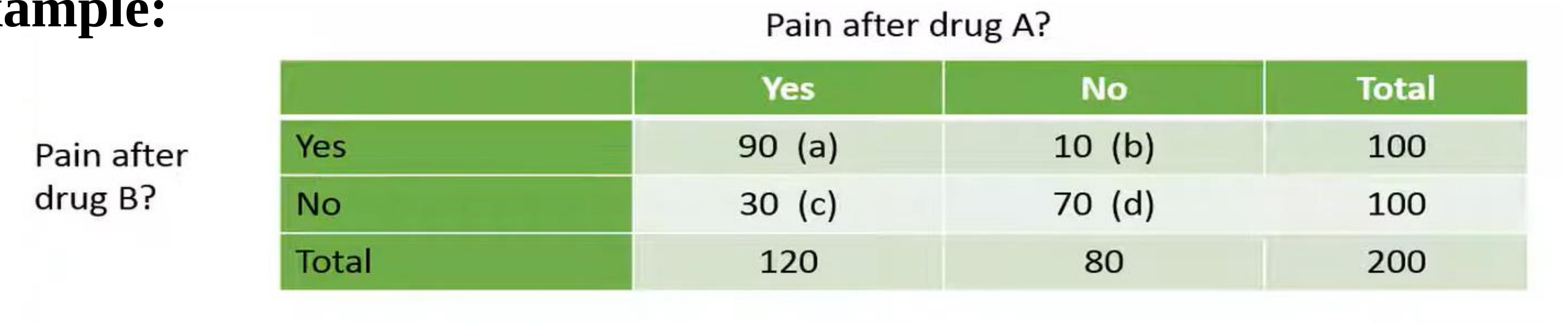
Comparison of Algorithms

How it works:

1. **Run both models on the same test dataset.**
2. **Create a 2x2 contingency table that categorizes the predictions:**
    - Cell a: Number of instances where both models were correct.
    - Cell b: Number of instances where Model 1 was correct and Model 2 was incorrect.
    - Cell c: Number of instances where Model 1 was incorrect and Model 2 was correct.
    - Cell d: Number of instances where both models were incorrect.
3. **Analyze the disagreements:** The test's null hypothesis is that the two models have the same error rate. If this were true, you would expect the number of disagreements in cell b and cell c to be roughly equal.
4. **Calculate the test statistic:** The test determines if the difference between b and c is statistically significant. A significant result (a low p-value) indicates that one model makes significantly fewer errors than the other on the points where they disagree.

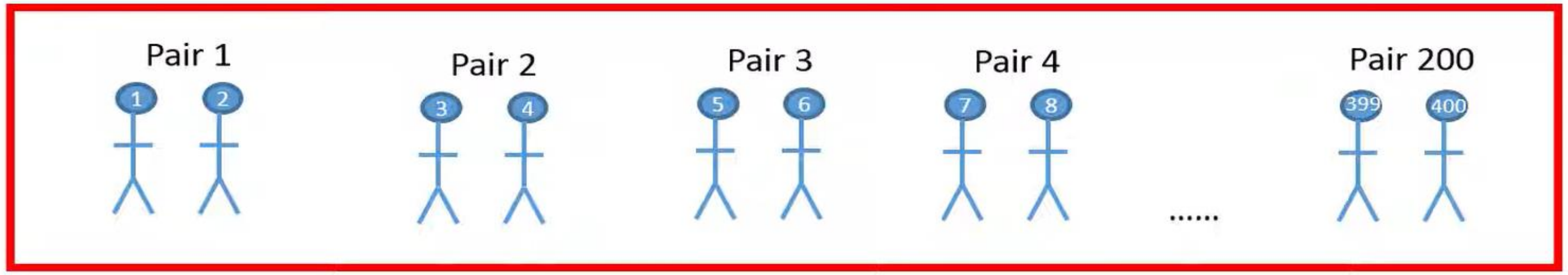
Comparison of Algorithms

Example:

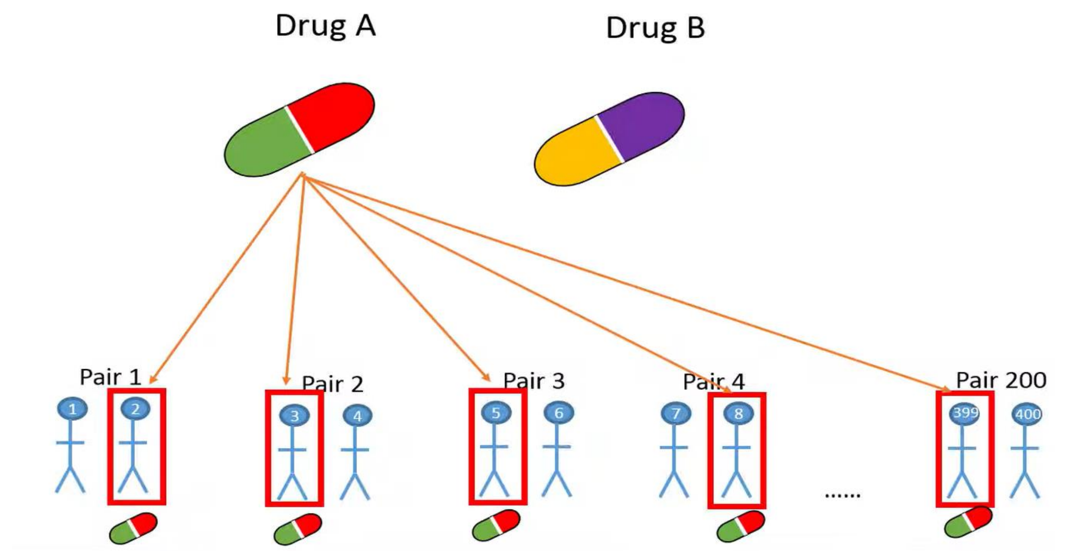


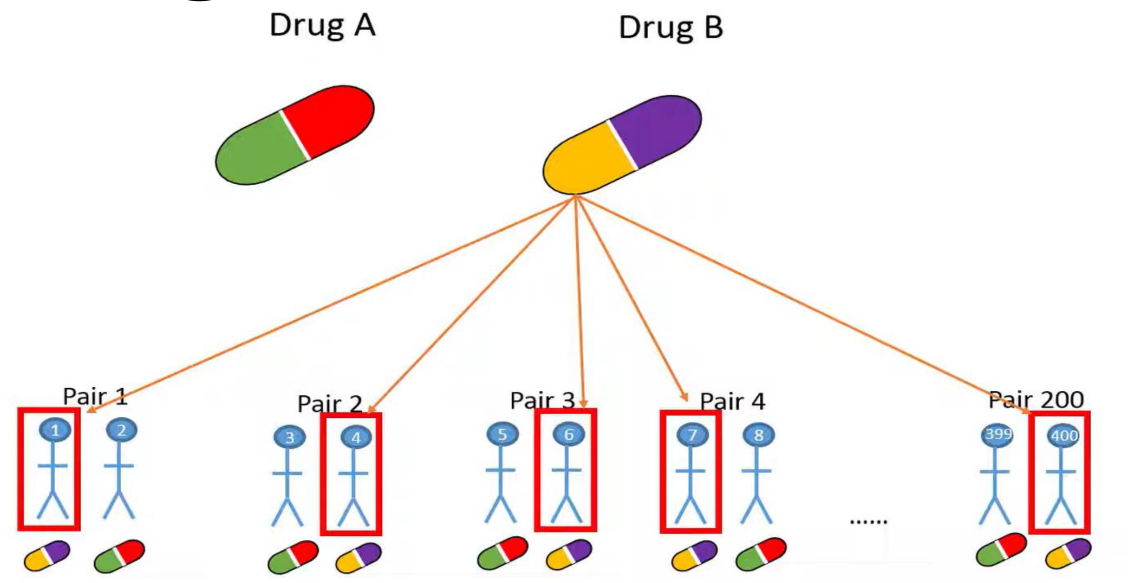
Pain after drug A?

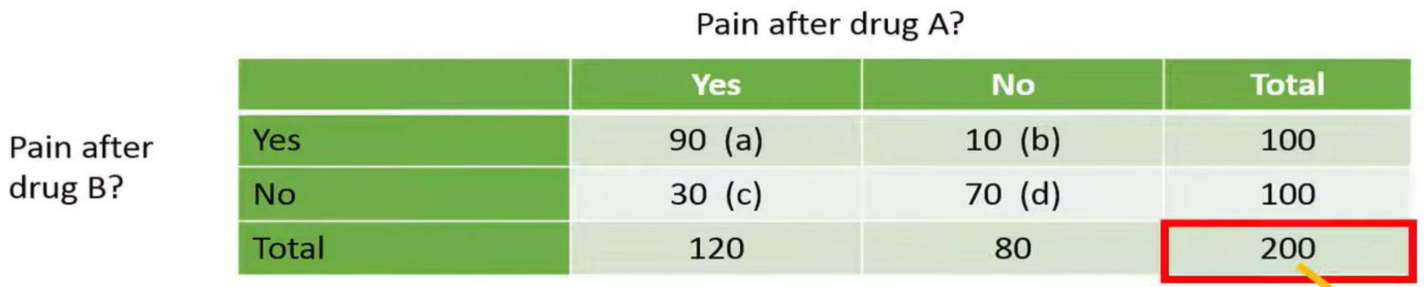
		Yes	No	Total
Pain after drug B?	Yes	90 (a)	10 (b)	100
	No	30 (c)	70 (d)	100
	Total	120	80	200



Comparison of Algorithms



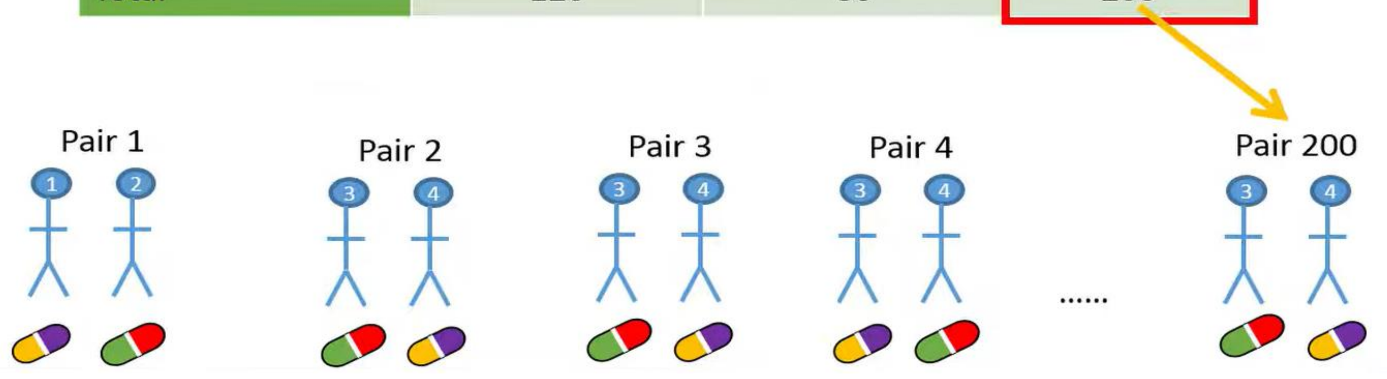




Pain after drug A?

Pain after drug B?

	Yes	No	Total
Yes	90 (a)	10 (b)	100
No	30 (c)	70 (d)	100
Total	120	80	200



Pain after drug A?

Pain after drug B?

	Yes	No	Total
Yes	90 (a)	10 (b)	100
No	30 (c)	70 (d)	100
Total	120	80	200

$$\chi^2 = \frac{(b-c)^2}{b+c}$$

$$\chi^2 = \frac{(10-30)^2}{10+30} = \frac{400}{40} = \boxed{10}$$

$$H0: p_b = p_c$$

$$\boxed{H1: p_b \neq p_c}$$

Binomial Test Appropriate Versus Mcnemar's Test

Feature	Binomial Test	McNemar's Test
Primary Use Case	To assess if a single classifier performs significantly better than a baseline (e.g., random chance) .	To compare the performance oftwo different classifierson the same dataset to see if there is a significant difference in their error rates .
Input Data	The number of correct predictions (k) out of the total predictions (n).	A 2x2 contingency table summarizing the instances where the two classifiers agree and disagree .
Null Hypothesis (H_0)	The classifier's performance is equal to a baseline probability (e.g. = 0.5for random guessing).	The two classifiers have the same error rate (the number of times Classifier A is right and B is wrong equals the number of times B is right and A is wrong) .

Ensemble Methods

Ensemble methods or Ensemble learning are techniques in machine learning that combine the predictions of multiple individual models (often called "base learners" or "weak learners") to produce a single, more accurate, and robust predictive model. The core idea is that by aggregating the "opinions" of several models, the final prediction will be better than the prediction of any single model alone.

Ensemble methods are effective because they help manage the bias-variance tradeoff. Different ensemble techniques can reduce either bias or variance, leading to improved overall performance and better generalization to new, unseen data.

Ensemble Methods

Ensemble Methods or learning

- Improve accuracy and stability
- Reduce overfitting
- Handle bias and variance better
- More robust to noisy data

Types of Ensemble Methods

1. Bagging (Bootstrap Aggregating)
2. Boosting
3. Stacking

Ensemble Methods

Bagging (Bootstrap Aggregating)

- **Bagging** Train multiple models **independently** on **randomly sampled subsets** of the data (with replacement).
- The final prediction is usually made by **averaging** (for regression) or **voting** (for classification).
- **Bagging** Reduce variance and prevent overfitting.
- **Example:** *Random Forest* (uses bagging on decision trees)

Different models trained on slightly different datasets → average their results → more stable predictions.

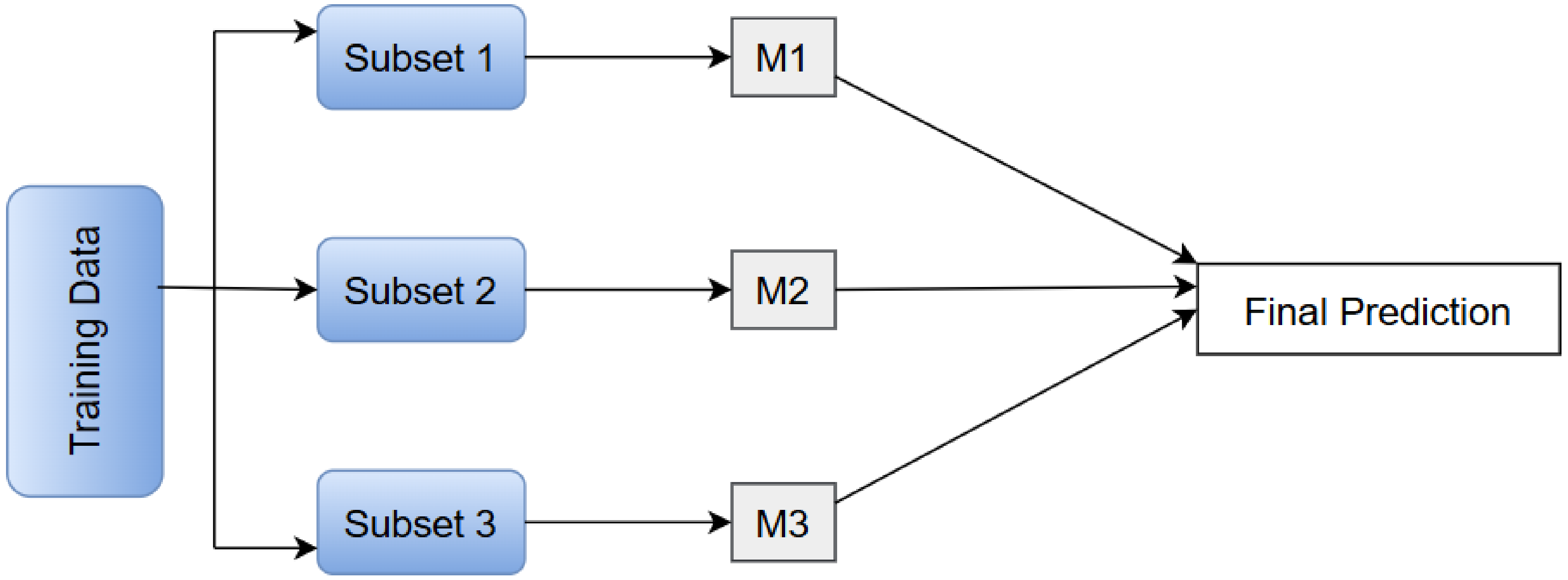
Ensemble Methods

How Bagging Works:

- **Bootstrap Sampling** Multiple subsets are created from the original dataset through random sampling with replacement. This means that some data points may appear multiple times in a single subset, while others may not be included at all.
- **Independent Model** Training A separate base model is trained on each of these subsets in parallel.
- **Aggregation** The predictions from all the individual models are combined. For regression tasks, the predictions are averaged. For classification tasks, the final prediction is determined by a majority vote.

A popular example of a bagging method is the Random Forest algorithm, which trains multiple decision trees on different subsets of the data.

Ensemble Methods



Ensemble Methods

Boosting

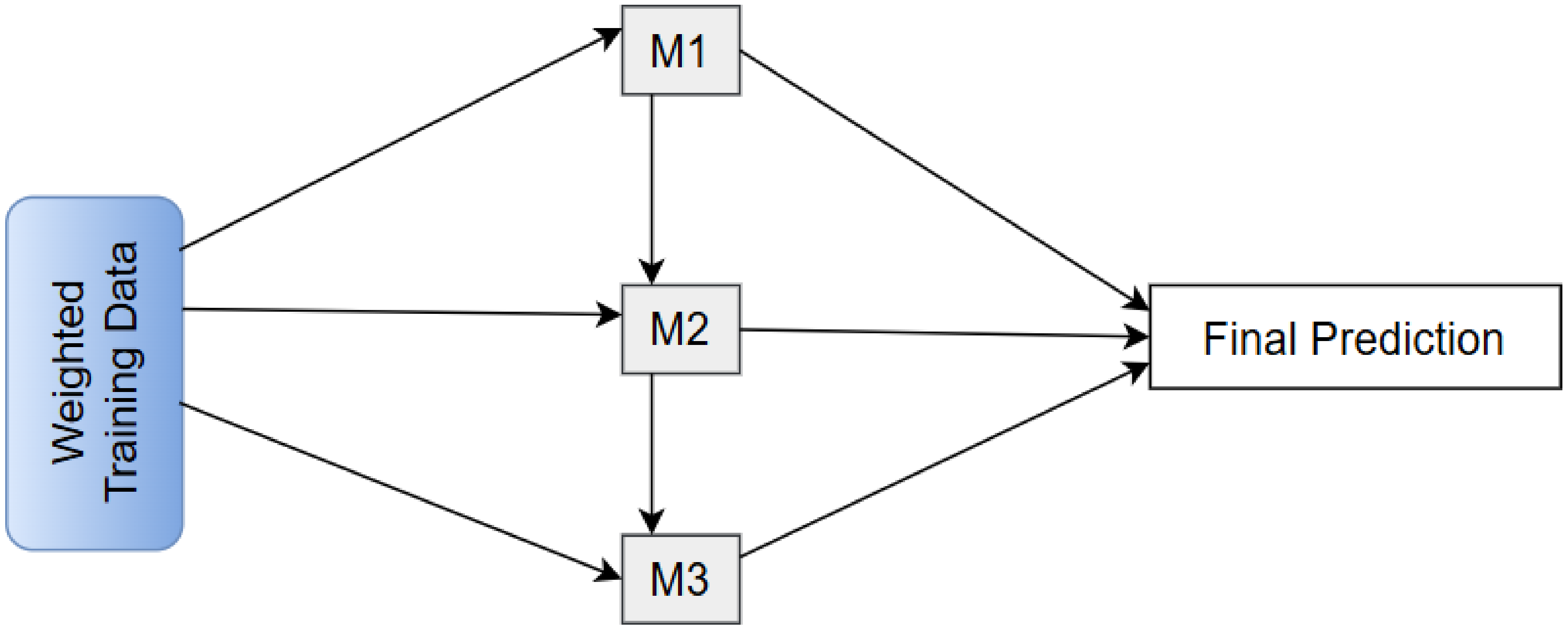
- **Boosting** Models are trained **sequentially**, where each new model focuses on **correcting the errors** made by the previous ones.
- **Boosting** Reduce bias and build a strong model from weak learners.
- **Common Algorithms:**
  - AdaBoost (Adaptive Boosting)
  - Gradient Boosting
  - XGBoost, LightGBM, CatBoost
- This models “boost” the performance by improving where earlier models failed.

Ensemble Methods

How Boosting Works:

- **Sequential Training** The models are trained in a sequence. The first model is trained on the original dataset.
- **Weight Adjustment** After each model is trained, the algorithm increases the weights of the data points that were misclassified. This forces the next model in the sequence to focus more on these difficult-to-predict instances.
- **Weighted Aggregation** The final prediction is a weighted sum of the predictions from all the individual models, where models that performed better are given more influence.

Ensemble Methods



Ensemble Methods

Stacking (Stacked Generalization)

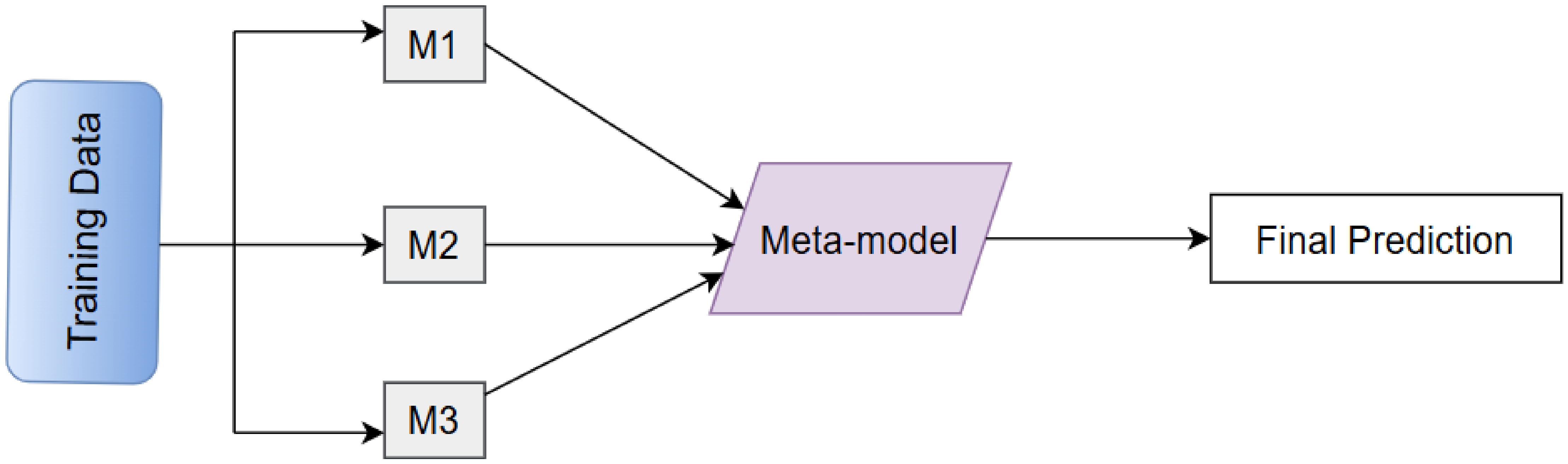
- **Stacking** Combines predictions of **different types of models** (e.g., decision tree, SVM, logistic regression) using a **meta-model**.
- The **meta-model** learns how to best combine the predictions from base models.
- **Stacking** Leverage the strengths of different algorithms.
- **Example:** Use Random Forest + SVM + Logistic Regression → combine their outputs using another model.

Ensemble Methods

How Stacking Works:

- **Level-0 Models (Base Models)** Several different base models (e.g., logistic regression, decision tree, SVM) are trained on the same training data. Unlike bagging, these models are typically of different types to ensure diversity.
- **Level-1 Model (Meta-Model)** The predictions generated by the base models are then used as input features to train a new model, called the meta-model or combiner algorithm. This meta-model learns to optimally combine the base model predictions to make the final prediction.
- **Final Prediction** The meta-model's output is the final prediction. Stacking often yields better performance than any single one of the trained models.

Ensemble Methods


Training Data
M1
M2
M3
Meta-model
Final Prediction