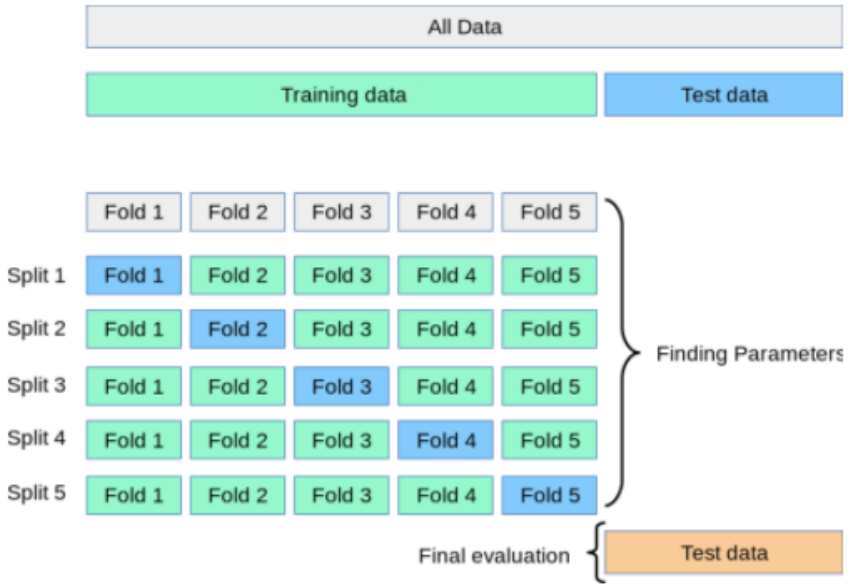


Week 8: Model Training, Evaluation, and Validation

Introduction

- We covered data preparation and model training in previous sessions. Today, we explore how to evaluate these trained models to ensure their effectiveness.

Objective of the Lesson:

Our goal is to equip you with the knowledge to assess model performance, understand key metrics, and validate models for reliable predictions.

Roadmap for the Session:

We'll delve into common evaluation metrics, explore model validation techniques, and touch on the importance of hyperparameter tuning.

Data Splitting

- Before training a model, it's crucial to divide the dataset into two subsets: a training set and a testing set.
- The training set is used to train the model, while the testing set is reserved for evaluating the model's performance on unseen data.
- Data splitting ensures the model is not overfitting (memorizing the training data) and can generalize well to new, unseen data.
- Commonly used split ratios include 70/30 or 80/20 for training/testing.

Model Training

- This phase involves feeding the training set into the chosen algorithm to enable it to learn the patterns and relationships within the data.
- The algorithm adjusts its parameters based on the input features and corresponding target values.
- The algorithm iteratively refines its parameters to minimize the difference between its predictions and the actual target values.
- The learning process involves adjusting coefficients (in regression), or decision boundaries (in decision trees).

Model Evaluation

- After training, the model's performance is assessed using the testing set, which the model has not seen before.

- Various metrics, such as accuracy, precision, recall, and F1 score, are calculated to evaluate how well the model generalizes to new data.

- Adjustments:
  - Based on the evaluation results, the model may be adjusted, hyperparameters tuned, or additional features considered to improve performance.

Key topics we'll cover

Evaluation Metrics: Understanding metrics crucial for assessing model performance.

Model Validation: Techniques to ensure models generalize well to new data.

Hyperparameter Tuning: Fine-tuning models for optimal performance.

Model Evaluation Metrics

- They allow us to gauge how well our machine learning models are performing in various aspects.

We'll explore metrics tailored to different types of tasks.

Common Evaluation Metrics:

- Classification Metrics: Accuracy, Precision, Recall, F1 Score.
- Regression Metrics: Mean Squared Error (MSE). (Self-Study)
- Receiver Operating Characteristic (ROC) Curve for Classification Models. (Self-Study)

Confusion Matrix

- A confusion matrix is a table that helps evaluate the performance of a classification model by summarizing its predictions.

- Table showing the true positive, true negative, false positive, and false negative counts.

- Provides a detailed breakdown of model performance, which are essential for making decisions about model adjustments and improvements.

Confusion Matrix Key Components

1. True Positives (TP):
- The number of instances correctly predicted as positive by the model.
- These are the cases where the model correctly identified positive instances.

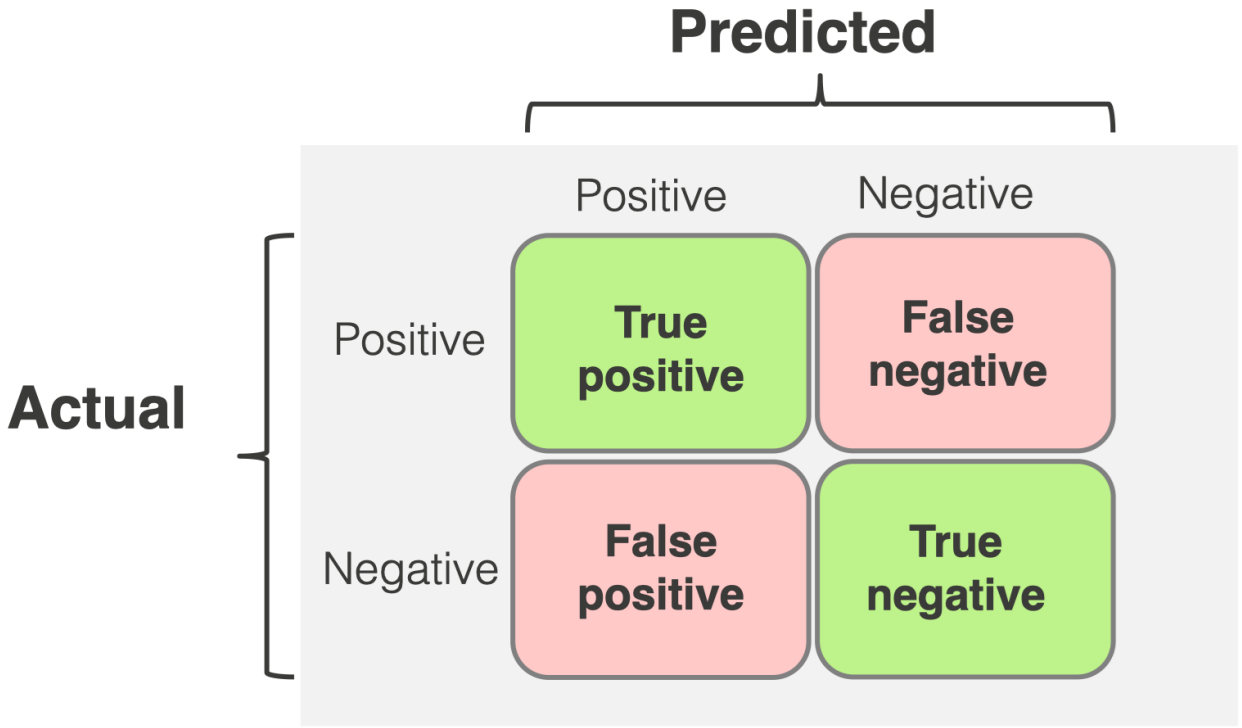
2. True Negatives (TN):
- The number of instances correctly predicted as negative by the model.
- These are the cases where the model correctly identified negative instances.

3. False Positives (FP):
- The number of instances incorrectly predicted as positive by the model.
- These are the cases where the model predicted positive, but the actual class was negative (Type I error).

4. False Negatives (FN):
- The number of instances incorrectly predicted as negative by the model.
- These are the cases where the model predicted negative, but the actual class was positive (Type II error).

Construction of Confusion Matrix

- Format: A 2x2 matrix with rows representing the actual classes and columns representing the predicted classes.



Accuracy

- Accuracy is a measure of overall correctness. It calculates the ratio of correctly predicted instances to the total instances.

Formula:

Accuracy = (True Positives + True Negatives) / Total Instances

- High accuracy suggests that the model is making correct predictions across both positive and negative classes.

Example: An accuracy of 90% means 9 out of 10 predictions are correct.

Considerations:

- Works well when classes are balanced.
- Accuracy can be misleading with imbalanced datasets. For example, in a dataset with 95% negative instances, a model predicting all negatives can still achieve 95% accuracy.

Precision

- Precision measures the accuracy of positive predictions. It calculates the ratio of true positive predictions to the total positive predictions made by the model.

Formula:

Precision = True Positives / (True Positives + False Positives)

- Out of all predicted positive instances, how many are actually positive.
- High precision indicates that when the model predicts a positive class, it is often correct.

Considerations:

- Suitability: Important when false positives are costly (e.g., spam detection).
- Limitation: Ignores false negatives.

Recall (Sensitivity or True Positive Rate)

- Recall, measures the ability of the model to capture all positive instances. It calculates the ratio of true positive predictions to the total actual positive instances.

Formula:

Recall = True Positives / (True Positives + False Negatives)

- Out of all actual positive instances, how many were correctly predicted.
- High recall indicates that the model effectively identifies a significant portion of the positive instances.

Considerations:

- Suitability: Important when false negatives are costly. (e.g., in scenarios like fraud detection, we want to minimize false negatives, or disease diagnosis).
- Limitation: Ignores false positives.

Specificity (True Negatives Rate)

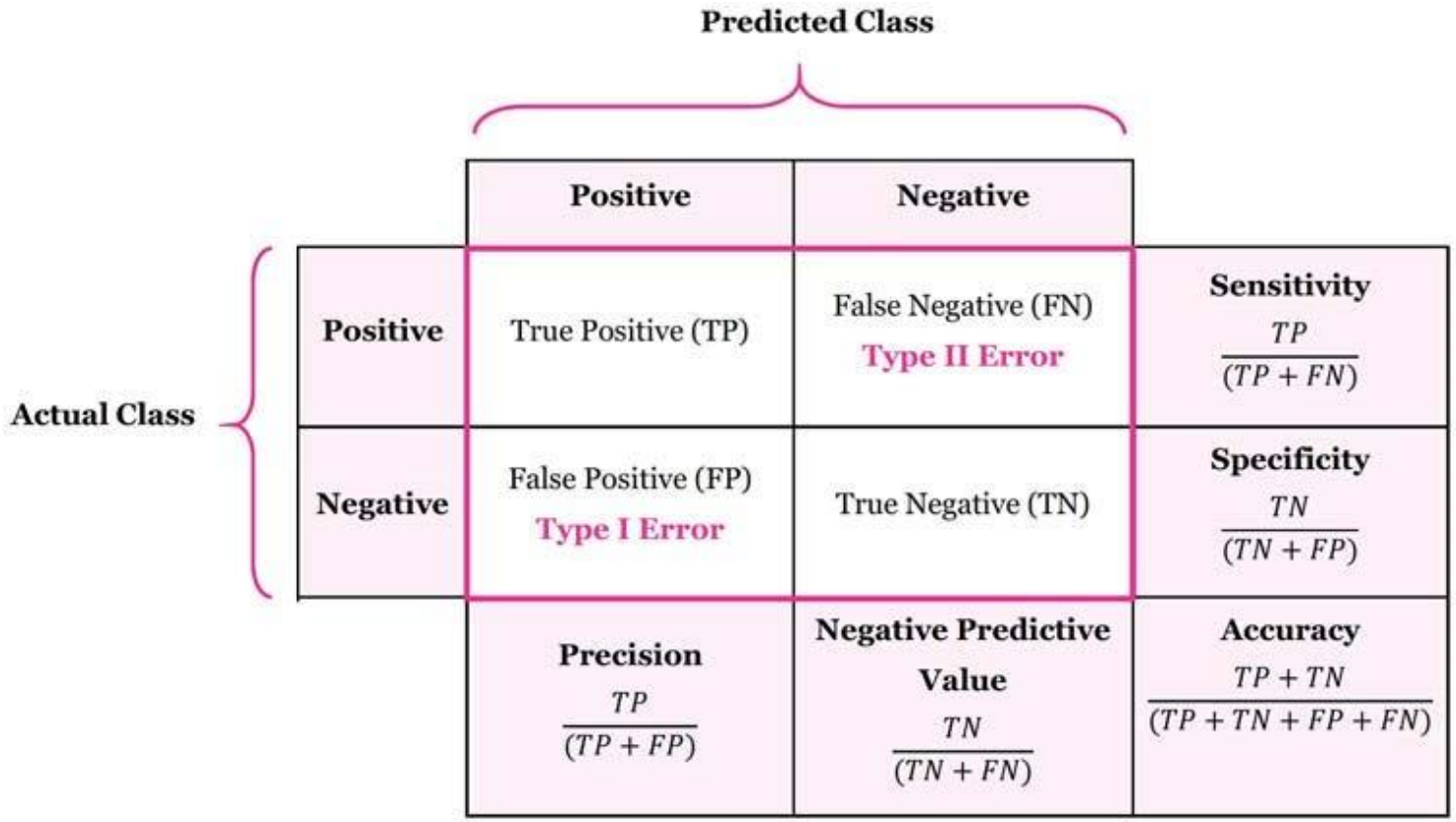
- Specificity is a measure of how well a classification model can correctly identify instances of the negative class (non-events or non-diseased cases).

Formula:

Specificity = True Negatives / (True Negatives + False Positives)

- Out of all actual negative instances, how many were correctly predicted.
- High specificity indicates a low rate of false positives, meaning the model is good at correctly identifying instances where the negative class is truly negative.
- In a medical diagnosis scenario, where the positive class represents a disease, specificity would measure how well the model identifies individuals without the disease.

Confusion Matrix



F1 Score

- The F1 Score is the harmonic mean of precision and recall. It provides a balance between precision and recall, especially in imbalanced datasets.

Formula:

F1 Score = 2 * (Precision * Recall) / (Precision + Recall)

- F1 Score combines the strengths of precision and recall. It is suitable when there's a need to balance false positives and false negatives.

Considerations:

- F1 Score is especially useful in scenarios where both precision and recall are equally important, avoiding favoring one over the other.

Evaluating the Model

Using Metrics: Accuracy, precision, recall, and F1 score.

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Evaluate accuracy
accuracy = accuracy_score(y_test, predictions)

# Evaluate precision
precision = precision_score(y_test, predictions)

# Evaluate recall
recall = recall_score(y_test, predictions)

# Evaluate F1 score
f1 = f1_score(y_test, predictions)
```

Example

- In this example, we'll use the Breast Cancer Wisconsin (Diagnostic) dataset from scikit-learn. This dataset is commonly used for binary classification tasks related to breast cancer diagnosis.

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train a Logistic Regression model
model = LogisticRegression(random_state=42)
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
```

Example

We calculate and print common evaluation metrics: accuracy, precision, recall, and F1 score.

```
# Calculate Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Calculate Evaluation Metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

# Display the results
print("Confusion Matrix:")
print(conf_matrix)
print("\nEvaluation Metrics:")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Confusion Matrix:
[[39  4]
 [ 1 70]]

Evaluation Metrics:
Accuracy: 0.9561
Precision: 0.9459
Recall: 0.9859
F1 Score: 0.9655
```

ROC Curve

- The ROC (Receiver Operating Characteristic) curve is a graphical representation of a binary classification model's ability to distinguish between the positive and negative classes.
- AUC-ROC: Area Under the ROC Curve, a single value representing the model's performance.

Components of ROC Curve:

1. True Positive Rate (TPR) or Sensitivity:

- TPR measures the proportion of actual positive instances that are correctly identified by the model.
  - **Formula: TPR = True Positives / (True Positives + False Negatives)**
- High TPR indicates that the model is good at capturing positive instances.

ROC Curve

2. False Positive Rate (FPR):

- FPR measures the proportion of actual negative instances that are incorrectly identified as positive by the model.

Formula: FPR = False Positives / (False Positives + True Negatives)

- Low FPR indicates that the model is not incorrectly labeling too many negatives as positives.

3. Thresholds:

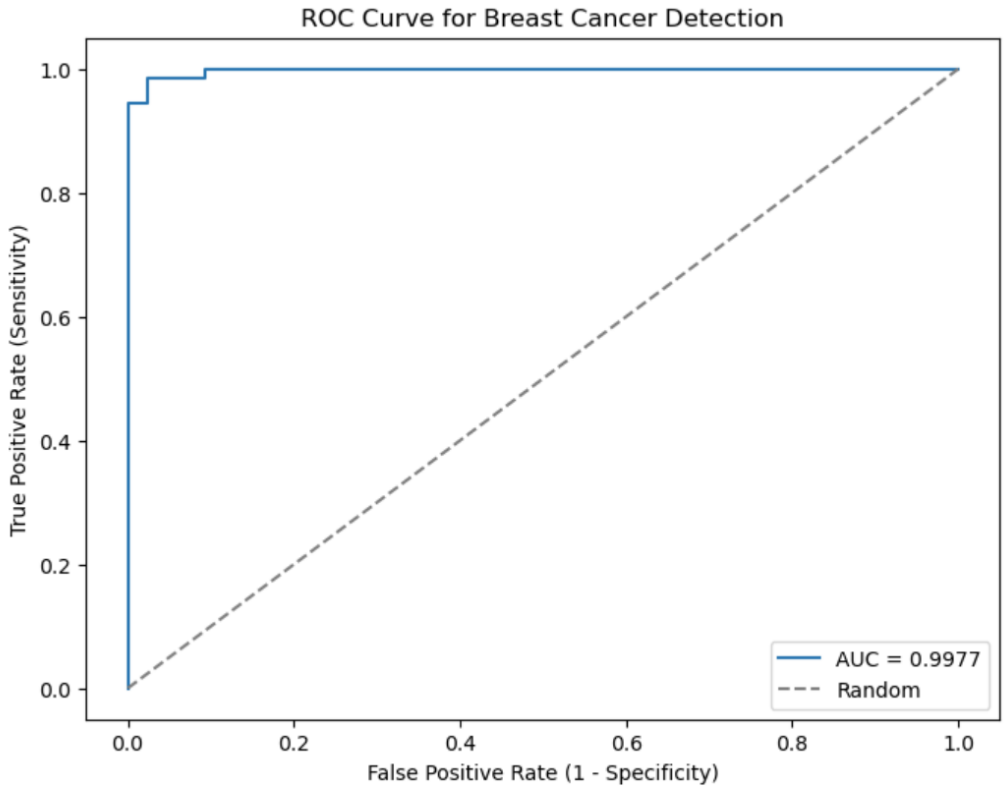
- The ROC curve is constructed by varying the decision threshold of the model, which determines the classification boundary between positive and negative instances.
- Each point on the curve corresponds to a different threshold.

Example

Finally, we plot the ROC curve to visualize the model's performance.

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Get predicted probabilities for the positive class
y_prob = model.predict_proba(X_test)[:, 1]

# Calculate ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Calculate AUC (Area Under the Curve)
roc_auc = roc_auc_score(y_test, y_prob)

# Plot ROC Curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'AUC = {roc_auc:.4f}')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Random')
plt.title('ROC Curve for Breast Cancer Detection')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.legend()
plt.show()

# Display AUC score
print(f"AUC: {roc_auc:.4f}")
```

Example




```
AUC: 0.9977
```

ROC Curve code explanation

```
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
```

- roc_curve is a function in scikit-learn used to calculate the ROC curve.
- It takes two parameters: y_test (true labels) and y_prob (predicted probabilities of the positive class).
- It returns three values:
  - fpr (False Positive Rate): The ratio of false positives to the total number of actual negatives.
  - tpr (True Positive Rate or Sensitivity): The ratio of true positives to the total number of actual positives.
  - thresholds (Thresholds): Threshold values used to calculate the points on the ROC curve.

ROC Curve explanation

```
roc_auc = roc_auc_score(y_test, y_prob)
```

- roc_auc_score is another function in scikit-learn used to calculate the Area Under the Curve (AUC) of the ROC curve.
- This function, roc_auc_score, takes the true labels (y_test) and the predicted probabilities (y_prob) and computes the AUC for the ROC curve.
- It takes two parameters: (y_test) and (y_prob)
- It returns the AUC score, which represents the area under the ROC curve.
- The AUC score is a single value that summarizes the model's ability to distinguish between positive and negative instances. A higher AUC score indicates better performance.

ROC Curve Interpretation

Perfect Model:

- In an ideal scenario, the ROC curve would reach the top-left corner, indicating a TPR of 1 and an FPR of 0.

Random Model:

- A random model would produce a diagonal line from the bottom-left to the top-right, indicating that the model's ability to distinguish between classes is no better than chance.

AUC (Area Under the Curve) Score:

- The AUC score quantifies the overall performance of the model by calculating the area under the ROC curve.
- A higher AUC score (closer to 1) indicates better discrimination between positive and negative instances.

ROC Curve Interpretation

- Visualizing the ROC curve helps us understand how the model's performance changes at different classification thresholds.
- A curve that rises steeply and approaches the top-left corner suggests a more effective model.

Model Selection:

Comparing ROC curves and AUC scores helps in selecting the best-performing model among multiple candidates.

Threshold Adjustment:

ROC curve analysis guides the selection of an appropriate threshold based on the desired balance between TPR and FPR.

Cross-Validation

- Cross-validation is a technique used to assess the performance and generalization ability of a machine learning model by dividing the dataset into multiple subsets.

Key Components:

1. Training Set:
   - The training set is the portion of the dataset used to train the model.
2. Validation Set:
   - The validation set is a subset of the dataset used to assess the model's performance during training.

Cross-Validation Techniques

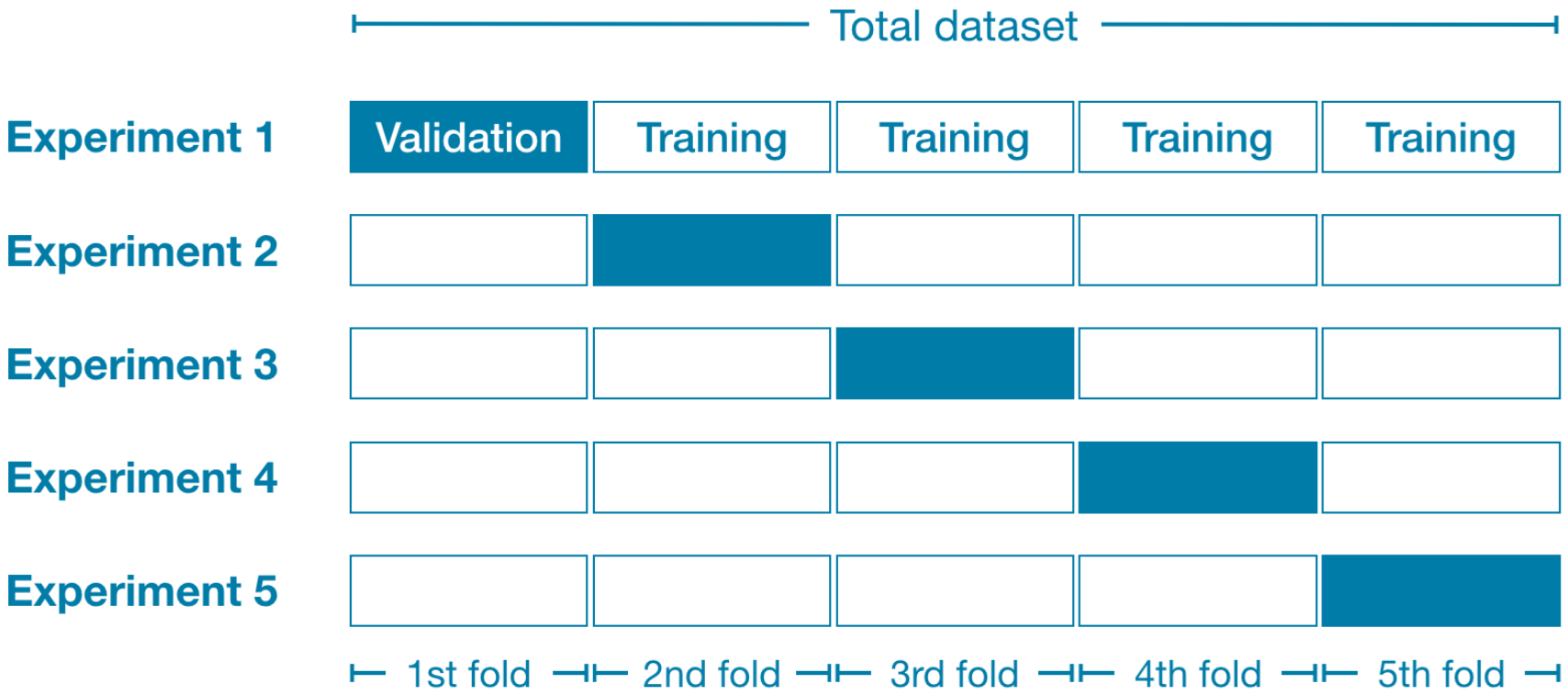
- K-fold cross-validation
- Hold-out cross-validation
- Stratified k-fold cross-validation
- Leave-p-out cross-validation

K-Fold Cross-Validation:

- The dataset is divided into k equally sized folds.
- The model is trained and validated k times, each time using a different fold as the validation set.
- All data points are used for both training and validation, ensuring comprehensive assessment.

K-Fold Cross-Validation Diagram

- The diagram illustrates how the dataset is divided into k folds, and the model is trained and validated iteratively.



Example

5-fold Cross-Validation

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import cross_val_score
from sklearn.svm import SVC

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Create an SVM classifier
model = SVC(random_state=42)

# Perform 5-fold cross-validation
cv_scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')

# Display the cross-validation scores
print("Cross-Validation Scores:")
print(cv_scores)
print("\nMean Accuracy: {:.4f}".format(cv_scores.mean()))
```

```
Cross-Validation Scores:
[0.85087719 0.89473684 0.92982456 0.94736842 0.9380531 ]

Mean Accuracy: 0.9122
```

Parameter vs. Hyperparameter:

Parameter:

Parameters are internal variables learned by the model during training.

Hyperparameter:

Hyperparameters are external settings that influence the model's learning process.

- Hyperparameters are external configuration settings that are not learned from the data but are set before the training process.

Examples: Learning rate, number of trees in a random forest, etc.

Hyperparameter Tuning

- Hyperparameter tuning is the process of finding the optimal configuration of hyperparameters for a machine learning model to improve its performance.
- Hyperparameter Tuning Strategies: Manual and Automated methods like grid search.

Manual Hyperparameter Tuning:

Advantages:

- Leverages domain knowledge and intuition.
- Allows for a targeted and efficient exploration.

Disadvantages:

- Limited in exploring a wide hyperparameter space.
- Time-consuming and may lack systematic exploration.

Hyperparameter Tuning

Automated Hyperparameter Tuning:

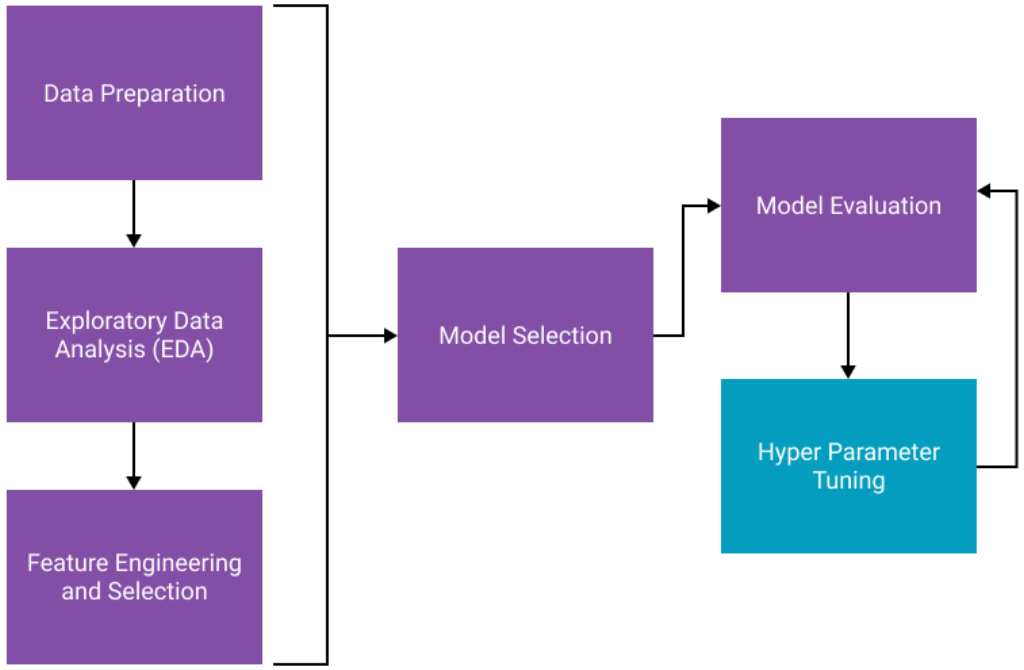
Advantages:

- Systematically explores a broad hyperparameter space.
- Efficient, especially for complex models.
- Easily reproducible and saves time.

Disadvantages:

- May lack interpretability, treating the model as a black box.
- Can be computationally expensive, especially with exhaustive searches.

Hyperparameter Tuning Diagram



Automated Tuning (Grid Search)

- Automated tools and libraries (e.g., GridSearchCV in scikit-learn) can streamline the hyperparameter tuning process.

Key Concepts:

Hyperparameters:

- Examples: Learning rate, regularization strength, number of trees in a random forest, etc.

Grid Search Space: Grid Search defines a grid of hyperparameter values to be explored.

- Example: For two hyperparameters, learning rate and number of trees, the grid might look like [(0.1, 0.01), (50, 100)].

Process of Grid Search

1. Selection of Hyperparameters:

- Choose the hyperparameters to be tuned and define their potential values to create a grid.

2. Grid Search Exploration:

- Iterations: Systematically explores all possible combinations of hyperparameter values in the defined grid.
- Each combination represents a different configuration for the model.

3. Model Training and Evaluation:

- Training: Train the model for each combination of hyperparameters.
- Evaluation: Evaluate the model's performance using a validation set or cross-validation.

4. Performance Comparison:

- Metrics: Compare models based on performance metrics.
- Selection: Choose the combination that yields the best performance.

Grid Search Consideration

Exhaustive Search:

- Grid Search performs an exhaustive search across the specified hyperparameter space, ensuring no combination is missed.

Optimal Model Configuration:

- The combination of hyperparameter values that leads to the best model performance is selected.

Computational Cost:

- Grid Search can be computationally expensive, especially with a large hyperparameter space.

Cross-Validation:

- Combining Grid Search with cross-validation provides a more robust estimate of model performance.

Example

Grid Search:

```
from sklearn.datasets import load_iris
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.neighbors import KNeighborsClassifier

# Load the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a k-Nearest Neighbors classifier
knn_model = KNeighborsClassifier()

# Define the hyperparameters and their possible values
param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'p': [1, 2],
}
```

Example

Grid Search:

```
# Create a grid search object with 5-fold cross-validation
grid_search = GridSearchCV(knn_model, param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Display the best hyperparameters and corresponding accuracy
best_params = grid_search.best_params_
best_accuracy = grid_search.best_score_
print("Best Hyperparameters:", best_params)
print("Best Cross-Validation Accuracy: {:.4f}".format(best_accuracy))

# Evaluate the model on the test set with the best hyperparameters
best_model = grid_search.best_estimator_
test_accuracy = best_model.score(X_test, y_test)
print("Test Set Accuracy with Best Hyperparameters: {:.4f}".format(test_accuracy))
```

```
Best Hyperparameters: {'n_neighbors': 3, 'p': 1, 'weights': 'uniform'}
Best Cross-Validation Accuracy: 0.9583
Test Set Accuracy with Best Hyperparameters: 1.0000
```