Supervised Learning Foundations

Data split
Model evaluation
Performance vs Interpretability
Accuracy vs Generalization
Regularization (L1, L2)
Feature Engineering
Hyperparameters

Julia Lenc

Training-Test Split

Definition

A train-test split is a method to divide data into two subsets:

- Training set: to train the machine learning model (larger size).
- Test set: to evaluate the performance on unseen data.

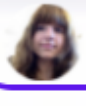
Common methods

1. **Simple split:** dividing the entire dataset.
   Common ratio is 80% training : 20% test
   If dataset size allows: 70% training : 20% validation : 10% test
   Common advanced models ratio is 90% training :10% test or 85 : 15

2. **Cross-validation:** multiple train-test splits to assess the model's performance. Most popular is k-Fold Cross-Validation.

3. **Stratified split (imbalanced datasets):** each class in the dataset is represented proportionally in both train and test sets.

4. **Time Series split (time-dependent data):** common approach is using the most recent observations for test set.


Julia Lenc

Training-Test Split

Important

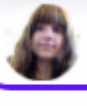
- Training set: must be always larger than test set.
- Test set: remains untouched during training (unbiased evaluation)

Scikit-Learn library makes splitting data easy using train_test_split

Replace test_size=0.2 indicated 80 : 20 split (replace if needed)

Use stratify=y to ensure proportional representation in sets

```
from sklearn.model_selection import train_test_split

# Assume X (features) and y (target) are already defined
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print("Training set size:", len(X_train))
print("Test set size:", len(X_test))
```



Julia Lenc

Evaluation: Classification Model

Confusion matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Core evalutation metrics

Accuracy : overall correctness. (TP + TN) / (Total Predictions)

Precision: % of correctly predicted positives among all positives. TP / (TP + FP)

Recall: % of correctly predicted positives among all corrects. TP / (TP + FN)

F1 score: model's performance. 2 * (Precision * Recall) / (Precision + Recall)

Secondary evalutation metrics

AUC-ROC curve: discrimination between positives and negatives.
0.5 - random guess, 1.0 - perfect discrimination.

Log Loss: correctness of prediction. 0 - perfect, ∞ - highly inaccurate.

Julia Lenc

Evaluation: Classification Model

Precision or Recall?

Precision if the cost of False Positive is high.

Examples: insurance company pay false claim, bank gives a loan to a customer with bad records, cybersecurity (false alarms disrupt operations).

Recall if capturing Positives is more important than avoiding False Positives.

Examples: fraudulent transactions detection, retail recommender system.

Scikit-Learn

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

# Predictions and true labels
y_true = [1, 0, 1, 1, 0, 1, 0, 0, 1]
y_pred = [1, 0, 1, 0, 0, 1, 1, 0, 1]

# Compute metrics
print("Accuracy:", accuracy_score(y_true, y_pred))
print("Precision:", precision_score(y_true, y_pred))
print("Recall:", recall_score(y_true, y_pred))
print("F1 Score:", f1_score(y_true, y_pred))

# Confusion matrix
matrix = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", matrix)

# Classification Report
print("Full Report:\n", classification_report(y_true, y_pred))
```

Julia Lenc

Evaluation: Regression Model

Evalutation metrics

y = target variable value n = number of observations

yi = actual value yi^ = predicted value $\bar{y_i}$ = mean of actual value

MAE (Mean Absolute Error): average absolute difference between the predicted and actual values. Most interpretable. $MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$

MSE (Mean Squared Error): average of the squared difference between the actual and predicted values. Penalizes for large errors. $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

RMSE (Root Mean Squared Error): like MSE, but expressed in the same units as the target variable. $RMSE = \sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$

MAPE (Mean Absolute Percentage Error): average of the absolute percentage differences between the actual and predicted values. $MAPE = \frac{100}{n}\sum_{i=1}^{n}|\frac{y_i - \hat{y}_i}{y_i}|$

R^2 (R-square): percentage of variance explained. $R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2}$

Adjusted R^2 : if higher than R^2, then the model is too complex (extra predictors do not add value. $Adjusted\ R^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$

Julia Lenc

Evaluation: Regression Model

Which metric is the best?

MAE or MAPE: interpretable, larger errors are acceptable as long as they cancel each other. Example: real estate (house prices).

MSE or R^2: relationship is deterministic, precision is critical, larger errors significantly impact conclusions. Example: R&D, engineering.

RMSE: larger errors must be penalized because they lead to system collapse. Example: supply chain, electricity demand forecasting.

MAPE: comparing models with different absolute values of target variable. Example: sales forecast for different countries.

Scikit-Learn

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# True values (y) and predicted values (y_pred)
y_true = [100, 200, 300, 400, 500]
y_pred = [110, 190, 310, 420, 490]

# Calculate metrics
mae = mean_absolute_error(y_true, y_pred)
mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_true, y_pred)
```

Julia Lenc

Performance vs Interpretability

Definitions

Performance: how well a model makes accurate predictions on unseen data. Key focus: **evaluation metrics**, generalization (evaluation on test set).

Interpretability: how easily humans can understand and explain how a model makes predictions. Key focus: **which factors** impact predictions and **how**.

High Interpretability (Simple Models) <-> High Performance (Complex Models)

Models by complexity

Core models

Regression: linear, non-linear, time series

Classification: logistic regression, Naive Bayes, k-NNs

Decision Trees

(tree depth: trade-off between performance and interpretability)

Support Vector Machines

Ensemble: Bagging

Ensemble: Boosting

Neural Networks


Julia Lenc

Performance or Interpretability?

Rule of thumb

Performance = Execution. **Interpretability** = Design or Regulations.

Examples

Fast Moving Consumer Goods
Interpretability - Marketing designs go-to-market campaign and needs to know how levels of media budget and its split impact profit.
Performance - Supply Chain Manager splits the forecast into weeks or days to secure stocks but avoid excessive inventory.

Fashion
Interpretability - Merchandiser aims to plan seasonal collections and needs to know which product attributes (e.g., colors, fabrics, styles) drive sales.
Performance - Recommender system for better conversions.

Banking
Interpretability - Credit risk officer wants to determine why a customer is likely to default on a loan (e.g., income instability, spending patterns).
Performance - Fraud detection model flags suspicious transactions in real time during online banking.

 Julia Lenc

Accuracy and Generalization

Definitions

Bias → Accuracy: how closely a model captures the patterns of the data.

- High Bias: The model oversimplifies, missing important patterns (**underfitting**).
- Low Bias: The model learns the patterns effectively but can become prone to minor changes in data (**overfitting**).

Variance → Generalization: how sensitive a model is to fluctuations in the data.

- High Variance: The model fits the training data very closely but fails to generalize to new data (**overfitting**).
- Low Variance: The model is more stable across datasets but might miss nuanced details (**underfitting**).

Detection

	High bias (underfit)	**High variance (overfit)**
Training error	High	Low
Test error	High	High
Learning curve	Training and Test errors are both high and close	Large gap between Training and Test errors


Julia Lenc

Fit detection: classification

F1-score comparison between Training and Test

- Underfitting: both training and test scores are low and unsatisfactory
- Overfitting: training score is high, but test score is much lower
- Balanced Model: training and test scores are similar and satisfactory

Scikit-Learn

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score

# Load data (replace 'classification_data.csv' with your file)
data = pd.read_csv('classification_data.csv')
X = data.drop(columns=['target'])  # Replace 'target' with your target column name
y = data['target']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit logistic regression
model = LogisticRegression()
model.fit(X_train, y_train)

# Calculate F1-Scores
train_f1 = f1_score(y_train, model.predict(X_train))
test_f1 = f1_score(y_test, model.predict(X_test))

print(f"Training F1-Score: {train_f1:.2f}")
print(f"Test F1-Score: {test_f1:.2f}")
```

Julia Lenc

Fit detection: regression

RMSE (or another error metric) comparison between Training and Test

- Underfitting: both training and test scores are low and unsatisfactory
- Overfitting: training score is high, but test score is much lower
- Balanced Model: training and test scores are similar and satisfactory

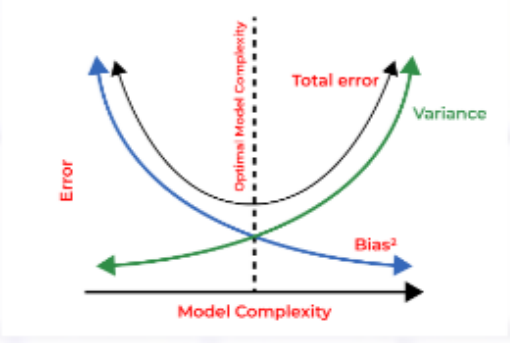
Scikit-Learn

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

# Load data (replace 'regression_data.csv' with your file)
data = pd.read_csv('regression_data.csv')
X = data[['feature']]  # Replace with the name of your feature column
y = data['target']     # Replace with the name of your target column

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Fit linear regression
model = LinearRegression()
model.fit(X_train, y_train)

# Calculate RMSE
train_rmse = np.sqrt(mean_squared_error(y_train, model.predict(X_train)))
test_rmse = np.sqrt(mean_squared_error(y_test, model.predict(X_test)))

print(f"Training RMSE: {train_rmse:.2f}")
print(f"Test RMSE: {test_rmse:.2f}")
```

Julia Lenc

Balancing Bias and Variance

1. Selecting right model complexity!!!



2. Cross-validation

Split the data multiple times to ensure the model generalizes well.

3. Regularization: L1 (Lasso), L2 (Ridge)

Penalizes overly complex models to reduce variance.

4. Other methods

- Decision Tree: pruning, max depth, min samples leaf
- Random Forest: max features, tree constraints
- Boosting: learning rate shrinkage, subsampling
- Neural Networks: dropout, batch normalization


Julia Lenc

Regularization

Definition

Regularization is a technique used to reduce overfitting by **discouraging complexity** in the model during training. It works after the initial evaluation, not preprocessing.

Types

1. L1 Regularization (Lasso)

Adds a penalty proportional to the absolute values of the model coefficients ($\sum|\text{weights}|$).Drives some weights to exactly 0, leading to sparse models (automatic feature selection).

2. L2 Regularization (Ridge)

Adds a penalty proportional to the squared values of the model coefficients ($\sum\text{weights}^2$). Shrinks coefficients smoothly, preventing extreme weight values and reducing sensitivity.


Julia Lenc

Regularization

Application

Scenario
A marketing team is conducting extended A/B testing on ad campaigns. They want to optimize for click-through rate (CTR) and engagement. They experiment with 100+ factors, including variations in layout, headlines, color schemes, audiences, etc. However, some features introduce noise and overload the model.

Problem = Generalization
Unstable or inconsistents results, when applied to new campaigns. Recommendations like *"This shade of blue in Region X leads to CTR spikes during weekdays"*.

After regularization
The team focuses on impactful patterns, like:

- Certain headlines (urgency-related) universally boost CTR.
- Specific color schemes appeal broadly.
- "Buy Now" consistently overperforms “Learn More”.

Julia Lenc

Feature Engineering

Definitions

Feature: A measurable property or characteristic of your data that helps the model predict the target outcome.
Example (sales prediction): price, advertising spend, seasonality.

Feature Engineering: the process of creating, transforming, and selecting features to improve model performance. Usually done at data preprocessing, sometimes after initial model evaluation.

Stages

1. **Creation:** deriving features from raw data.
2. **Transformation:** modifying features to make them more interpretable or linear for the model.
3. **Extraction:** seriving high-level features from complex data.
4. **Scaling:** standardizing or normalizing numerical features.

Julia Lenc

Feature Engineering

Stage	Libraries and Functions	Example output
Creation	pandas, featuretools	daily sales aggregated to monthly
Transformation	numpy, scipy.stats, sklearn.preprocessing	sales growth %, log transformed sales
Extraction	sklearn.feature_extraction, tensorflow, opencv	lag features, rolling averages
Selection	sklearn.feature_selection, Lasso, RFE	log_1 and rolling_avg_3 are most useful for future predictions
Scaling	sklearn.preprocessing (MinMaxScaler, StandardScaler)	z-score normalization (all values have similar ranges)

Julia Lenc

Hyperparameters tuning

Definitions

Hyperparameters: parameters set before training; control learning process.

Hyperparameters tuning: a process to optimize hyperparameters.

Rule of thumb

Simple model (few parameters, less sensitive)
<->
Complex model (may parameters, sensitive)

Key hyperparameters

Linear and Logistic Regressions: regularization strength

Decision Tree: complexity (max_depth, min_samples_split, min_samples_leaf)

SVM: regularization strength, kernel (type, coefficient - gamma)

Random Forest: complexity (n_estimators, max_depth)

Boosting: iterations (n_estimators), shrinking contribution (learning_rate)

Neural Networks: Learning Rate (lr), batch size, layers and neurons, dropout rate, activation functions (relu, sigmoid, tanh), optimizer (adam, sgd, rmsprop)

Julia Lenc

Did you find it useful?

Save

Share

Follow

Analytics, Market Research,
Machine Learning and AI

Julia Lenc