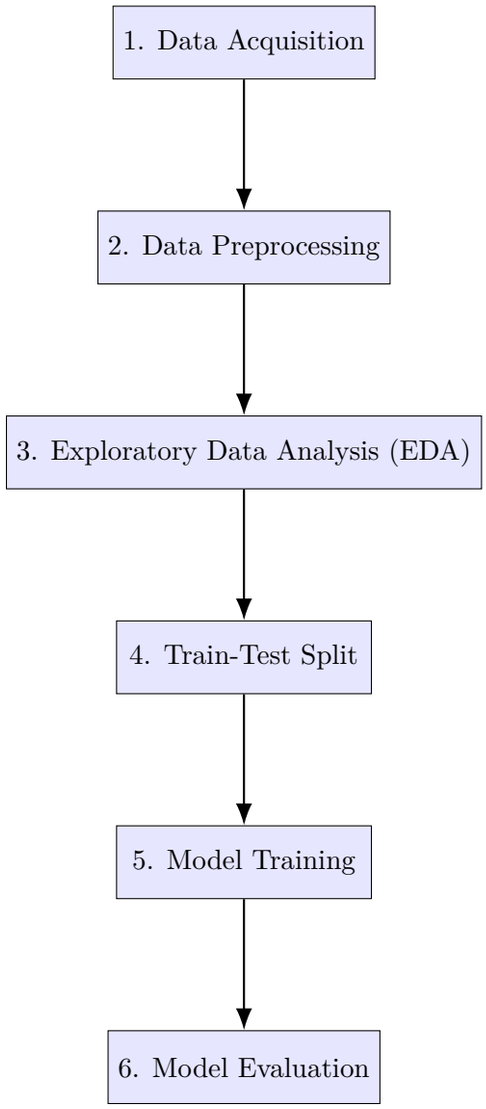
Machine Learning Project Workflow

Prepared by:

Bryan Christian Dirly Pratama

Systematic Workflow Overview



The diagram above illustrates the typical stages followed in a machine learning project. Each stage builds upon the previous, ensuring a structured and methodical approach to solving data-driven problems.

Step 1: Data Acquisition

Data acquisition is the foundational stage of any machine learning project. It involves identifying, collecting, and selecting data relevant to the problem we aim to solve. The quality, volume, and suitability of the data heavily influence the success of the final model.

Example: Consider a case where we aim to predict diabetes. The dataset may include attributes such as glucose level, blood pressure, age, and body mass index (BMI). Relevant data can be obtained from:

- Public datasets (e.g., UCI Machine Learning Repository, Kaggle)
- Government health databases
- APIs from healthcare platforms
- Hospital or institutional records
- Peer-reviewed academic publications

Key Considerations in Data Acquisition:

- **Relevance:** The data must be directly related to the prediction task.
- **Volume and Diversity:** Sufficient and representative samples are crucial.
- **Accuracy and Validity:** Ensure data sources are reliable and verifiable.
- **Privacy and Ethics:** Handle sensitive data responsibly (e.g., GDPR, HIPAA).

Types of Data:

- **Structured Data:** Clearly defined fields and types, like rows in spreadsheets or databases.
- **Unstructured Data:** Includes text, images, audio, or video. Requires additional processing techniques.

Common Data Formats:

- CSV (Comma-Separated Values)
- Excel (.xlsx)
- JSON (JavaScript Object Notation)
- SQL Databases

Once the dataset is obtained, it's important to conduct an initial inspection—also known as a data sanity check. This includes examining the number of rows and columns, identifying missing values, understanding variable types, and reviewing summary statistics. These steps lay the groundwork for the preprocessing phase.

"Better data beats fancier algorithms." – Andrew Ng

Step 2: Data Preprocessing

After acquiring the dataset, the next step is to prepare the data for modeling. Raw data is often incomplete, inconsistent, or in a format unsuitable for machine learning algorithms. Therefore, data preprocessing is essential to clean, transform, and structure the data appropriately before analysis and modeling.

Common Issues in Raw Data:

- Missing values in one or more columns
- Inconsistent or duplicate entries
- Categorical variables that need encoding
- Features with different scales
- Presence of outliers

Key Steps in Data Preprocessing:

- **Handling Missing Values:** Rows or columns with missing values may be dropped, or the missing entries can be imputed using techniques such as mean, median, or predictive models.
- **Encoding Categorical Variables:** Machine learning models typically require numerical input. Categorical data must be converted using label encoding or one-hot encoding.
- **Feature Scaling:** Variables with different units or ranges can bias the model. Common scaling methods include normalization (Min-Max Scaling) and standardization (Z-score).
- **Removing Duplicates and Outliers:** Duplicate entries add noise, and outliers can skew learning. Outlier detection may be done using statistical thresholds or domain knowledge.
- **Data Type Conversion:** Ensure each column has the appropriate data type (e.g., integers, floats, datetime).

Example: In the diabetes prediction dataset, some patients might have missing blood pressure readings. These can be replaced by the mean value of the column or estimated using related features like BMI and age. Similarly, if a categorical column such as "smoking status" is present, it must be encoded into numerical format before feeding it to the model.

A well-preprocessed dataset improves model performance, reduces training time, and ensures that the algorithm can extract meaningful patterns. It is one of the most important steps in any machine learning pipeline and often takes more time than the modeling itself.

"Data preprocessing is not glamorous, but it's where great models begin."

Step 3: Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is the process of analyzing datasets visually and statistically to uncover patterns, detect anomalies, and test assumptions. This step serves as a bridge between raw data and modeling, helping analysts understand which features are most relevant to the target variable.

Objectives of EDA:

- Understand the distribution of each feature
- Identify correlations between variables
- Detect missing values, outliers, or skewed distributions
- Determine the relationships between input features and the target

Common EDA Techniques:

- **Summary Statistics:** Mean, median, mode, standard deviation, min/max values.
- **Univariate Analysis:** Histograms or boxplots to observe individual feature distributions.
- **Bivariate Analysis:** Scatter plots, bar plots, and cross-tabulations for exploring relationships between two variables.
- **Correlation Matrix:** A heatmap showing correlation coefficients between features to detect multicollinearity.
- **Target Comparison:** Visualizations comparing feature values across different classes of the target variable.

Example: In the diabetes dataset, we might plot the distribution of glucose levels and observe whether diabetic patients tend to have higher glucose levels. Similarly, a correlation matrix may reveal that glucose and BMI are positively correlated with the outcome variable, suggesting their potential predictive value.

Feature Importance Consideration: EDA also helps assess which features are most influential. While this may be finalized later using model-based methods, early insight from EDA allows better preprocessing and selection.

EDA is both an art and a science. It combines visual intuition with statistical evidence, enabling the analyst to form hypotheses, identify necessary transformations, and avoid pitfalls in the modeling stage.

"If you torture the data long enough, it will confess to anything." – Ronald Coase

Step 4: Train-Test Split

Before training a machine learning model, it is essential to divide the dataset into separate subsets for training and testing. This allows us to evaluate how well the model generalizes to new, unseen data and prevents overfitting.

Why Split the Data?

- To train the model on a portion of the data.
- To evaluate its performance using data it has never seen before.
- To simulate real-world scenarios where the model must make predictions on new inputs.

Typical Split Ratios:

- **80/20:** 80% for training, 20% for testing — a common standard.
- **70/30 or 60/40:** Used when more test data is preferred for evaluation.
- **Validation Set:** Sometimes, the data is split into three sets: training, validation, and testing. The validation set is used for tuning hyperparameters.

Stratified Splitting: When dealing with classification problems, especially imbalanced classes (e.g., 90% negative, 10% positive cases), a stratified split ensures the proportion of classes remains consistent across both training and testing sets.

Example: Suppose our diabetes dataset contains 1,000 entries. Using an 80/20 split, we assign 800 entries for training and 200 for testing. If the dataset contains 35% diabetic patients, stratified splitting will preserve that same ratio in both subsets.

Implementation Note: In most ML libraries (such as scikit-learn in Python), train-test splitting is supported with built-in functions like `train_test_split()` which can randomize the split and support stratification.

Careful and consistent splitting is crucial for trustworthy evaluation. A model that performs well on training data but poorly on testing data may be overfitting and not generalizing — a key issue to identify before deployment.

"Never trust a model until you've tested it on data it hasn't seen."

Step 5: Model Training

Once the data has been preprocessed and split, the next stage is to train the machine learning model. This is where the actual "learning" takes place — the model tries to identify mathematical patterns in the data that best explain the relationship between inputs and outputs.

What is Model Training?

Model training refers to the process of feeding the training dataset into a machine learning algorithm, allowing it to iteratively learn from the input features and corresponding output labels (in supervised learning). The model adjusts its internal parameters to minimize the prediction error.

Common Algorithms for Classification and Regression:

- **Logistic Regression** – Simple yet powerful for binary classification.
- **Linear Regression** – Models linear relationships for regression tasks.
- **Decision Trees and Random Forests** – Intuitive models that split data using rules.
- **Support Vector Machines (SVM)** – Maximizes the margin between data classes.
- **K-Nearest Neighbors (KNN)** – Makes predictions based on similarity to neighbors.
- **Naive Bayes** – Assumes independence between features; fast and effective for text.
- **Gradient Boosting Models (e.g., XGBoost, LightGBM)** – Ensemble models with strong performance.
- **Neural Networks** – Flexible models inspired by the human brain, effective for complex patterns.

Core Concepts in Model Training:

1. Loss Function: The loss function quantifies how well the model's predictions align with the true labels. It guides the optimization process. Examples include:

- **Mean Squared Error (MSE)** – For regression.
- **Cross-Entropy Loss** – For classification.
- **Hinge Loss** – For SVMs.

2. Optimization and Gradient Descent: Gradient descent is an optimization algorithm used to minimize the loss function. The model calculates the gradient of the loss with respect to its parameters and updates them iteratively:

$$\theta \leftarrow \theta - \alpha \cdot \nabla L(\theta)$$

where:

- θ are the model parameters,
- α is the learning rate,
- $\nabla L(\theta)$ is the gradient of the loss function.

Gradient descent has several variants:

- **Batch Gradient Descent** – Uses the entire training set per update.

- **Stochastic Gradient Descent (SGD)** – Updates per single sample.
- **Mini-batch Gradient Descent** – A balance between batch and SGD.

Step 5: Model Training (Continued)

3. Epochs and Iterations: An epoch refers to one full pass through the entire training dataset. The model is typically trained for multiple epochs. Within each epoch, the data may be divided into mini-batches, and each batch update is one iteration.

4. Underfitting vs Overfitting:

- **Underfitting:** The model is too simple and fails to capture the underlying pattern.
- **Overfitting:** The model performs well on training data but poorly on unseen data. It memorizes rather than generalizes.

5. Regularization: Regularization helps prevent overfitting by adding a penalty term to the loss function. Examples:

- **L1 Regularization (Lasso)** – Promotes sparsity.
- **L2 Regularization (Ridge)** – Shrinks coefficients smoothly.

Example: In our diabetes prediction task, we might use logistic regression as a baseline model. The model will use features like glucose level, age, and BMI to predict whether a patient is diabetic. The learning algorithm updates the weights associated with each feature until the loss (difference between predicted and actual values) is minimized.

Training Code Example (Conceptual in Python):

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, y_train)
```

This command trains the logistic regression model using the training inputs (`X_train`) and outputs (`y_train`).

The model training process may require experimentation with different algorithms, hyperparameters, or even feature sets. Model performance is not just about raw accuracy, but also about how well it generalizes, which will be tested in the next step: evaluation.

"The more you sweat in training, the less you bleed in production."

Step 6: Model Evaluation

I. Introduction

Model evaluation is a critical phase in a machine learning workflow. It ensures that the model not only learns from training data but also generalizes well to unseen data. Evaluation answers the question: *How good is my model?*

Why Evaluate?

- To quantify prediction accuracy or error
- To compare different models or tuning strategies
- To avoid overfitting or underfitting
- To select the best model for deployment

When to Evaluate?

1. **Before training:** Split dataset into training, validation, and test sets.
2. **During training:** Use validation metrics to tune models.
3. **After training:** Evaluate final model on holdout test set.

What to Evaluate?

- **Classification:** Predicting categories (e.g., spam vs. not spam)
- **Regression:** Predicting continuous quantities (e.g., temperature, price)

Each type of problem requires different metrics:

- Classification uses confusion matrix-based metrics and probabilistic scores
- Regression uses error-based metrics like MAE, MSE, and R^2

This chapter explores standard evaluation metrics for both classification and regression, as well as advanced techniques like cross-validation and model selection principles.

II.A. Classification Evaluation: Confusion Matrix and Core Metrics

Classification models predict discrete class labels. A widely used tool for evaluating classification performance is the **confusion matrix**, which summarizes the number of correct and incorrect predictions in a tabular format.

Binary Confusion Matrix:

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

Derived Metrics:

- **Accuracy** — overall proportion of correct predictions:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Precision** (Positive Predictive Value) — proportion of positive predictions that are correct:

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Recall** (Sensitivity, True Positive Rate) — proportion of actual positives correctly predicted:

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **F1 Score** — harmonic mean of precision and recall:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Specificity** (True Negative Rate) — proportion of actual negatives correctly predicted:

$$\text{Specificity} = \frac{TN}{TN + FP}$$

- **Balanced Accuracy** — mean of recall and specificity:

$$\text{Balanced Accuracy} = \frac{\text{Recall} + \text{Specificity}}{2}$$

Interpretation Note: When the dataset is imbalanced (e.g., rare positive cases), relying solely on accuracy can be misleading. Precision and recall provide a deeper understanding of how well the classifier handles minority classes.

F1 score is often used when seeking a balance between precision and recall, especially under class imbalance conditions.

II.A.2. Regression Evaluation: Core Metrics

Regression models predict continuous numeric outcomes. The performance of such models is evaluated based on the magnitude of prediction errors — how far predictions are from true values.

Let:

- y_i: true value for observation i
- $\hat{y}_i$: predicted value for observation i
- n: total number of observations

1. Mean Absolute Error (MAE):

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

MAE measures the average magnitude of the errors without considering their direction. It is simple and interpretable in the same unit as the output.

2. Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

MSE penalizes larger errors more severely than MAE due to squaring. It is useful when large errors are especially undesirable.

3. Root Mean Squared Error (RMSE):

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

RMSE is the square root of MSE and thus in the same unit as the output variable. It is more sensitive to outliers than MAE.

4. Coefficient of Determination (R^2 Score):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

This score represents the proportion of variance in the dependent variable explained by the model.

- $R^2 = 1$: perfect prediction
- $R^2 = 0$: model predicts no better than the mean
- $R^2 < 0$: model performs worse than a constant predictor

Interpretation: MAE and RMSE provide direct measures of prediction accuracy, while R^2 offers insight into how well the model captures the overall variance structure in the data.

Choosing the best metric depends on the application's sensitivity to error size, unit interpretability, and robustness to outliers.

II.B. Classification Evaluation: Advanced Metrics (Part 1)

Beyond basic classification metrics, we can use probability-based and ranking-based measures to better understand model performance—especially when thresholds or probability estimates are involved.

1. ROC Curve and AUC (Area Under the Curve)

The ROC (Receiver Operating Characteristic) curve shows how a classifier's **true positive rate** (sensitivity) and **false positive rate** change as we vary the decision threshold.

- **True Positive Rate (TPR)**:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} \quad \text{(a.k.a. Recall)}$$

- **False Positive Rate (FPR)**:

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

The ROC curve plots TPR vs. FPR at multiple thresholds (e.g., from 0 to 1). A good classifier "hugs" the top-left corner of the graph.

AUC (Area Under Curve) quantifies the total area under the ROC curve:

$$0.5 \leq \text{AUC} \leq 1.0$$

Interpretation:

- AUC = 1.0: Perfect prediction ranking
- AUC = 0.5: Model ranks no better than random guessing

2. Logarithmic Loss (LogLoss)

While ROC evaluates *ranking*, LogLoss evaluates *probability quality*. LogLoss penalizes false confidence in wrong predictions.

Let:

- $y_i \in \{0, 1\}$: true class label
- $p_i \in [0, 1]$: predicted probability that $y_i = 1$

Then:

$$\text{LogLoss} = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Interpretation:

- Smaller values mean better calibration
- A prediction of 1.0 when true class is 0 gives infinite penalty
- Use LogLoss when probability accuracy is critical (e.g., medicine, finance)

Example: If the true label is 1 and the predicted probability is 0.9, LogLoss is:

$$-[1 \cdot \log(0.9) + 0 \cdot \log(0.1)] \approx 0.105$$

Note: LogLoss is undefined for probabilities 0 or 1; models often clip probabilities near 0 and 1 to avoid infinities.

II.B. Classification Evaluation: Advanced Metrics (Part 2)

3. Matthews Correlation Coefficient (MCC)

MCC is a correlation coefficient between predicted and actual binary classes. It works well even with imbalanced datasets.

$$\text{MCC} = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Value range: -1 to $+1$

- $+1$: Perfect prediction
- 0: No better than random
- -1: Total disagreement between prediction and ground truth

Why MCC matters:

- Balanced for all four elements of confusion matrix
- More informative than accuracy in imbalanced datasets

4. Cohen's Kappa (κ)

Cohen's Kappa measures agreement between prediction and actual label *adjusted for chance agreement.*

$$\kappa = \frac{P_o - P_e}{1 - P_e}$$

Where:

- P_o: Observed accuracy $= \frac{TP+TN}{n}$
- P_e: Expected agreement by chance:

$$P_e = \left(\frac{TP + FP}{n} \cdot \frac{TP + FN}{n}\right) + \left(\frac{TN + FN}{n} \cdot \frac{TN + FP}{n}\right)$$

Interpretation:

κ Range	Strength of Agreement
> 0.80	Almost Perfect
$0.60 - 0.80$	Substantial
$0.40 - 0.60$	Moderate
$0.20 - 0.40$	Fair
$0.00 - 0.20$	Slight
< 0.00	Poor

Summary:

These advanced metrics give deeper insight into classifier behavior:

- ROC/AUC: How well the model ranks positive cases

- LogLoss: How well the model calibrates its confidence
- MCC: Overall correlation-like quality
- Kappa: Agreement beyond chance

III. Regression Evaluation: Advanced Metrics

While MAE, MSE, RMSE, and R^2 are commonly used, some contexts demand alternative metrics for interpretability, percentage error analysis, or accounting for model complexity.

1. Adjusted R^2

The standard R^2 increases with more features, even if they add no real predictive power. The adjusted R^2 corrects for this by penalizing model complexity.

$$\text{Adjusted } R^2 = 1 - \left(\frac{(1 - R^2)(n - 1)}{n - k - 1} \right)$$

Where:

- n: number of observations
- k: number of predictors/features

Use When: comparing models with different numbers of input features.

2. Mean Absolute Percentage Error (MAPE)

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

Pros:

- Easy to interpret as a percentage error

Cons:

- Undefined if $y_i = 0$
- Heavily penalizes small actual values

3. Symmetric Mean Absolute Percentage Error (SMAPE)

To reduce MAPE's bias on small values:

$$\text{SMAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{\left(|y_i| + |\hat{y}_i| \right) / 2}$$

Properties:

- Always defined unless both y_i and $\hat{y}_i$ are 0
- Normalized error between 0

4. Explained Variance Score

$$\text{Explained Variance} = 1 - \frac{\text{Var}(y - \hat{y})}{\text{Var}(y)}$$

Measures how much of the variance in the target is captured by the model. Similar to R^2, but focuses on prediction variance, not total error.

Interpretation:

- 1.0: perfect predictions
- < 0: model adds more error than using the mean

When to Use These Metrics:

- Use **MAPE** or **SMAPE** when stakeholders expect percentage-based performance
- Use **Adjusted** R^2 when comparing models with different feature sets
- Use **Explained Variance** to assess consistency in prediction quality

IV. Cross-Validation

Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent dataset.

1. K-Fold Cross-Validation

- Data is split into k equally sized folds.
- The model is trained on $k-1$ folds and tested on the remaining fold.
- This is repeated k times, each fold serving once as the test set.

$$\text{CV Score} = \frac{1}{k}\sum_{i=1}^{k}\text{Score}_i$$

Typical values: $k = 5$ or 10. Larger k offers less bias but higher variance and computation time.

2. Stratified K-Fold (Classification Only)

Ensures that the proportion of classes in each fold matches the overall dataset distribution — useful in imbalanced classification.

3. Leave-One-Out (LOO) CV

A special case of K-Fold where $k = n$. Very thorough but computationally expensive.

4. Repeated Cross-Validation

Repeat K-Fold multiple times with different random splits. Useful to obtain stable metric estimates.

V. Model Selection Principles

Once multiple models are trained, we must choose one based on generalization, simplicity, and metric trade-offs.

Key Considerations:

- **Performance:** Use cross-validated scores (e.g., mean F1, RMSE)
- **Bias–Variance Trade-Off:** Balance underfitting and overfitting
- **Interpretability:** May prefer a slightly worse but more explainable model
- **Computation:** Consider training time and inference speed

Selection Techniques:

- Compare models using validation metrics on same folds
- Use visualization: e.g., ROC curves, residual plots
- Use model ensembling if multiple models perform equally well

Common Traps to Avoid:

- Tuning hyperparameters using test data
- Using accuracy as sole metric in imbalanced classification
- Ignoring domain constraints (e.g., interpretability, cost)

Final Advice: Use both quantitative metrics and qualitative considerations to guide final deployment decisions. No metric is universally best — context is key.