Sri Lanka Institute of Information Technology

Faculty of Computing

IT2011 - Artificial Intelligence and Machine Learning

Dr.Lakmini Abeywardhana

Year 02 and Semester 01

Lecture 4

Machine Learning Pipeline

Learning Objectives

- Understand the key steps in a Machine Learning project
- Apply various preprocessing techniques on data
- Evaluate ML models using appropriate metrics

Overview of Machine Learning Pipeline

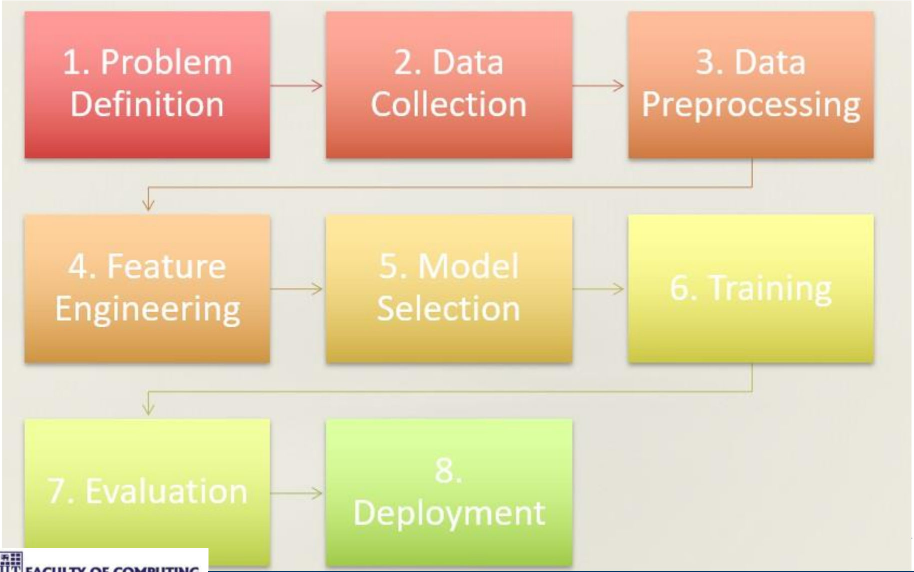
Definition and importance

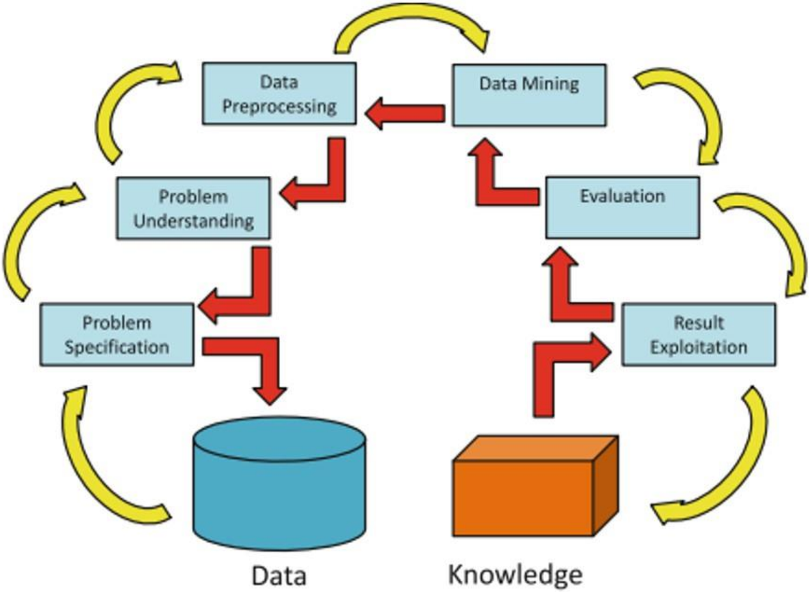
- A **Machine Learning Pipeline** is a sequence of steps that transforms raw data into valuable predictions using machine learning techniques.
- It helps automate and streamline:
    - **Data flow**
    - **Model building**
    - **Evaluation**
    - **Deployment**

Key Stages in the ML Pipeline

Stage	What Happens
1. Data Collection	Gather raw data from sources like sensors, websites, databases, files
2. Data Preprocessing	Clean, fix, and format the data (remove nulls, encode text, scale numbers)
3. Feature Engineering	Create and transform features that help the model learn better
4. Model Selection	Choose the right algorithm (e.g., decision tree, logistic regression)
5. Model Training	Teach the model to learn from the data using the selected algorithm
6. Model Evaluation	Measure how well the model performs using test data (accuracy, error metrics)
7. Model Deployment	Make the model available in the real world (e.g., in an app or website)

Steps in a Machine Learning Project




Data Preprocessing
Data Mining
Problem Understanding
Evaluation
Problem Specification
Result Exploitation
Data
Knowledge

1. Problem Definition

UNDERSTANDING DOMAIN

IDENTIFYING BUSINESS OBJECTIVE

DETERMINING ML TASK TYPE

2. Data Collection

Structured vs. Unstructured Data

- **Structured**: tabular format, easily searchable (e.g., databases, CSV files)
- **Unstructured**: free-form text, images, audio, etc. (e.g., social media, emails)

Sources of Data

- APIs (e.g., Twitter API, OpenWeatherMap)
- Databases (SQL, NoSQL)
- Files (CSV, Excel, JSON, XML)

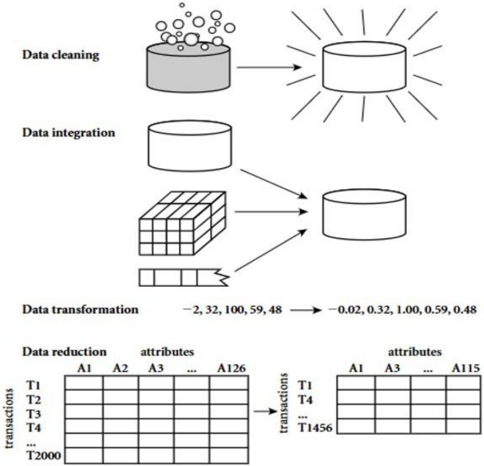
Data Acquisition Tools

- Web scraping: BeautifulSoup, Scrapy
- API clients: Postman, Python Requests
- ETL tools: Apache NiFi, Talend, Airbyte

3. Data Preprocessing

- Raw data is often **messy**. Before using it in a machine learning model, we need to **clean, fix, and prepare** it. This process is called **data preprocessing**.
- Cleaning, Handling missing values
- Encoding categorical features
- Normalization/Standardization

3. Data Preprocessing



Source: https://hanj.cs.illinois.edu/cs412/bk3/03.pdf#page=4.25

Steps in Data Preprocessing

1. **Handling Missing Data**
   - Sometimes a value may be missing (e.g., temperature not recorded)

Solutions:

- Fill with average (mean)
- Fill with zero or ”unknown”
- Drop the row (if too many values missing)

2. **Converting Categorical Data**
   - Machines work with **numbers**, not words.
   - We need to convert categories like "Yes"/"No" or "Male"/"Female" into numbers.

Common Methods:

- **Label Encoding**: Yes → 1, No → 0
- **One-Hot Encoding**: Country → UK, India → separate columns
- **Target Encoding**: Replace categories with the **mean of the target** variable for each category

Encoding Types

Encoding Type	How It Works	Best Used For	Pros	Cons / Caution
Label Encoding	Assigns a unique number to each category (e.g., Yes → 1, No → 0)	Ordinal data (e.g., Low, Medium, High)	Simple and memory-efficient	Implies order even when not present — ❗ use only if order matters
One-Hot Encoding	Creates a new column for each category with binary 0/1	Nominal data (unordered, e.g., Country, Color)	No false ranking, easy to understand	Can create many columns with high-cardinality features
Target Encoding	Replaces category with average value of the target variable for that category	High-cardinality features (e.g., Product ID)	Efficient for large feature sets	Risk of overfitting — use regularization or cross-validation

3. Scaling the Data

- Imagine "Age" ranges from 10–80 but "Marks" range from 0–10. The machine may give more attention to **bigger numbers**.
- Scaling ensures all features are treated equally.

Common Methods:

- **Min-Max Scaling**: Rescales values between 0 and 1
- **Standardization (Z-score)**: Makes data follow a standard scale

Data Cleaning

- Data in the Real World Is Dirty: Lots of potentially incorrect data, e.g., instrument faulty, human or computer error, transmission error
  - incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data
    - e.g., Occupation=" " (missing data)
  - noisy: containing noise, errors, or outliers
    - e.g., Salary="10" (an error)
  - inconsistent: containing discrepancies in codes or names, e.g.,
    - Age="42", Birthday="03/07/2010"
    - Was rating "1, 2, 3", now rating "A, B, C"
    - discrepancy between duplicate records
  - Intentional (e.g., disguised missing data)
    - Jan. 1 as everyone's birthday?

4. Feature Engineering

- The process of transforming raw data into meaningful features that improve the performance of machine learning models.
- Good features can make simple models powerful, while poor features can render complex models useless.
- **Role in ML Pipeline**: Comes after data preprocessing and before model training.

What is Feature Engineering?

Imagine you're building a machine that learns to predict whether a student will pass or fail based on their study habits, sleep hours, and attendance. The information you use to help the machine make this decision is called **features**.

Feature engineering means:

- Picking the right information (features)
- Improving it
- Removing the unnecessary parts

- **Good feature engineering = smarter machine learning!**

What Are Features?

- The information we give to the machine.
- Features are the **columns in your dataset**

Example (Student Data):

Hours Studied	Attendance (%)	Sleeps Before Exam	Result
5	90	8 hours	Pass
0	50	2 hours	Fail

Features: Hours Studied, Attendance, Sleep Hours
Target: Result (this is what we want to predict)

Why Do We Need Feature Engineering?

- Machines don’t understand raw data as we do.
- Some features are not helpful, some are **confusing**, and some are hidden and need to be created.
- We want to make our data **clean, useful, and easy to understand** for the machine.

Steps in Feature Engineering

Cleaning and Preparing Data

- Fill missing values (e.g., if sleep hours are missing, maybe use the average)
- Convert text to numbers (e.g., "Pass" → 1, "Fail" → 0)

Choosing the Best Features

Feature Selection – Picking the Most Helpful Features

- Not all information is useful. Some may be repeated, or have little effect.
- Simple Methods:
  - Variance Threshold
  - Mutual Information
  - SelectKBest
  - Model-Based Selection

Variance Threshold

Purpose:

- Removes all features with low variance (i.e., features that don't vary much across samples and thus carry little information).

Key Idea:

- A feature with zero or near-zero variance is likely constant or almost constant and not helpful for prediction. **If a column has almost the same value for everyone (e.g., all students slept 8 hours), it's not useful.**

How It Works:

- Calculates variance of each feature.
- Removes features where variance $<$ a user-defined threshold (default = 0).

Use Case:

- Preprocessing step for high-dimensional data
- Suitable for **numerical data**

Mutual Information

Purpose:

- Measures the amount of information shared between a feature and the target variable. High mutual information means knowing the feature helps predict the target.

Key Idea:

- **Helps find out which features are most related to the target (e.g., Attendance might be strongly linked to passing).** Unlike correlation (which captures only linear relationships), **mutual information** captures **any kind of dependency** (linear or non-linear).

How It Works:

- Estimate how much information a feature gives about the target.
- Scores range from 0 (no info) to 1 (perfect info).

Use Case:

- Works for **both categorical and numerical** data
- Very useful when target is **non-linear** with input features

SelectKBest / SelectPercentile

Purpose:

- These are general-purpose selection wrappers that choose top K features (or top X percentile) based on a scoring function.

Key Idea:

- This method picks the **top 'K' best features** — like choosing your top 5 ingredients for a recipe!
  Plug-and-play with any scoring function: chi-square, mutual information, ANOVA F-value, etc.

How It Works:

- Calculate a univariate score for each feature using a statistical test
- Select top K features (or top X%)

Use Case:

- Quick filtering based on relevance
- Works with **classification or regression**

Model-Based Selection (e.g., Random Forest)

Purpose:

- Uses a machine learning model to determine the importance of features, then selects based on importance scores.

Key Idea:

- Some models (like decision trees and ensembles) **internally compute feature importance** during training.
  Some models can **indicate which features were most helpful (much like a teacher identifying which exam questions contributed the most to determining** grades).

How It Works:

- Train a model (e.g., Random Forest)
- Access .feature_importances_ attribute
- Remove features with low importance

Methods of feature selection

- Feature selection methods are generally categorized into **three** main types:
  - Filter Methods
  - Wrapper Methods
  - Embedded Methods

Filter Methods

Approach: Select features based on statistical tests and intrinsic properties of the data — independent of any machine learning algorithm.

Key Characteristics:

- Fast and computationally efficient
- Useful for preprocessing large datasets
- Not tailored to a specific model

Technique	**Description**	**Suitable For**
Variance Threshold	Removes features with low variance	Numerical features
Correlation Coefficient	Select features based on correlation with target	Linear relationships
Chi-Square Test	Measures independence between categorical variables and target	Classification (categorical)
ANOVA (F-test)	Compares variance between groups	Classification problems

Wrapper Methods

Approach: Evaluate different combinations of features using a predictive model. The performance (e.g., accuracy, RMSE) determines which features to keep.

Key Characteristics:

- More accurate than filter methods
- Computationally expensive
- Prone to overfitting if not carefully tuned

Technique	Description
Forward Selection	Start with no features, add one at a time based on performance
Backward Elimination	Start with all features, remove one at a time
Recursive Feature Elimination (RFE)	Recursively fits model and removes least important feature

Embedded Methods

Approach: Feature selection is **integrated within** the model training process. The model itself selects the important features during learning.

Key Characteristics:

- Balance between performance and computational cost
- Model-specific
- Automatically selects features during training

Technique	Description	Applicable Models
Lasso Regression (L1 regularization)	Shrinks some coefficients to zero	Linear Models
Decision Tree Feature Importance	Selects features based on impurity reduction (e.g., Gini)	Trees, Random Forest
Elastic Net	Combines L1 and L2 regularization	Linear Models

Filter vs Wrapper vs Embedded

Type	Uses Model?	Speed	Accuracy	Risk of Overfitting
Filter	❌ No	✅ Fast	⚠️ May ignore interactions	Low
Wrapper	✅ Yes	❌ Slow	✅ High (model-driven)	High
Embedded	✅ Yes	⚖️ Medium	✅ High	Medium

Dimensionality Reduction

When we have **too many features**, it's like giving the machine too many things to think about. Some may even confuse it.

Dimensionality reduction means:

- **Combining or removing features**
- Keeping only what's **essential**

Example Technique: PCA (Principal Component Analysis)

- Think of it as squeezing a big photo into a smaller one, but still keeping the important parts visible.

Feature Selection vs Dimensionality Reduction

Aspect	Feature Selection	Dimensionality Reduction
Goal	Select a subset of the **original features**	Create **new features** by combining existing ones
Output	Original features (fewer in number)	Transformed features (compressed form)
Interpretability	High (original feature names are retained)	Low (new features are combinations of originals, e.g., PC1)
How it works	Keeps features based on importance or relevance	Projects features into a new space (e.g., using PCA)
Examples	Variance Threshold, Mutual Info, SelectKBest, RFE	PCA (Principal Component Analysis), Autoencoders, t-SNE
Data type	Works well with any data	Mostly used for numerical data
Use case	When you want to keep features understandable	When model speed or visual clarity is more important
Pros	Simple, interpretable, useful for all ML models	Reduces overfitting, better visualization
Cons	Might still retain correlated or redundant features	Harder to interpret transformed features

What is a Machine Learning Model?

- A **model** is like a recipe or formula the computer learns to make predictions.
- It ”learns” from the data during the training process

How to Select a Model?

Factors to consider:

- What is the **problem type?** (Classification, Regression, Clustering)
- How much data do you have?
- Do you need the model to be **explainable**?
- Do you prefer accuracy or speed?

Problem Type	Model Type Example	Purpose
Predicting a category (Pass/Fail)	**Classification** – Logistic Regression, Decision Tree	Classification
Predicting a number (Marks)	**Regression** – Linear Regression	Numerical prediction
Grouping data without labels	**Clustering** – K-Means	Grouping similar things

6. Model Training – Let the Learning Begin

Step-by-Step: From Data to Model

1. **Input**: Cleaned and engineered data
2. **Split the data** into:
   - **Training set**: For learning (e.g., 80%)
   - **Test set**: For checking performance (e.g., 20%)
3. **Train the model**:
   - What happens here?
     The model tries to understand patterns between the features and the target.

Hyperparameter Tuning

- What Are Hyperparameters?
  - A **parameter** is learned by the model from the data (e.g., weights in linear regression)
  - A **hyperparameter** is a setting that **you define before training** the model
- Think of hyperparameters as the "settings" or "knobs" you can tune to improve model performance.

Examples of Hyperparameters

Model	Hyperparameter Examples
Decision Tree	max_depth, min_samples_split
K-Nearest Neighbors	n_neighbors, metric
SVM	kernel, C, gamma
Random Forest	n_estimators, max_features

Why Tune Hyperparameters?

- To improve model performance
- To avoid overfitting or underfitting
- To make the model faster and more efficient
- The right hyperparameter values can significantly improve your results without changing the model.

Common Hyperparameter Tuning Techniques

Method	**Description**	**Best Used When...**	**Example**
Manual Tuning	Try combinations by hand and observe performance	You are experimenting or learning, with few parameters	Try n_neighbors = 3, then 5, then 7 in KNN and see which gives best accuracy
Grid Search	Tries **all possible combinations** of provided hyperparameter values	You have **limited combinations** and want thorough results	Search over n_neighbors = [3, 5, 7] and weights = ['uniform', 'distance'] for KNN
Random Search	Samples random combinations from a defined range	You have **many possible combinations** and want to reduce computation time	Randomly test max_depth from 2–10 and min_samples_split from 2–10 in a Decision Tree

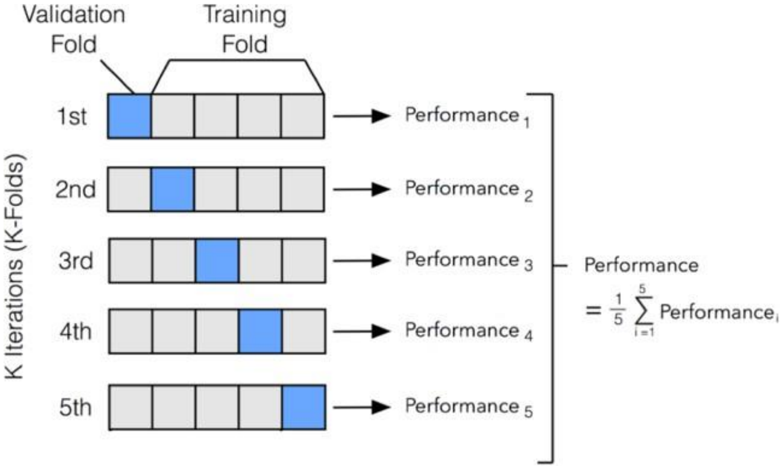
Cross Validation

- What is Cross-Validation?
- Why Cross-Validation?
  - A single train/test split might give a lucky or unlucky result.
- **Cross-validation** helps check if your model performs consistently.
- It splits data into and tests the model more than once.

How It Works – K-Fold Cross-Validation

- Split data into **K equal parts** (called ”folds”)
- Train on **K-1 folds**, test on the 1 remaining
- Repeat **K times**, each time changing the test fold
- Calculate **average score** = more reliable performance estimate

- Example: 5-Fold cross validation Train on 80%, test on 20%, repeat 5 times with different splits



Source: https://www.kaggle.com/discussions/general/204878

Benefits of Cross-Validation

Feature	**Benefit**
Multiple Tests	More **stable & fair evaluation**
Works on small data	Makes the most of limited data
Reduces **Overfitting**	Prevents tuning to one test set

What is overfitting?

7. Model Evaluation

A model that performs well on training data might **fail** on new data.That's why we evaluate using unseen test data.

That's why we evaluate using **unseen test data**. **Evaluation helps us answer:**

- Is the model accurate?
- Is it making too many mistakes?
- Does it work well for **all types of data**, not just training examples?

Common metrices for model evaluation

Type	**Metric**	**Interpretation**
Classification	Accuracy	Overall correctness
	Precision	Correctness of positive predictions
	Recall	Coverage of actual positives
	F1 Score	Balance between P and R
Regression	MAE	Avg. absolute error
	MSE	Avg. squared error
	R^2 Score	% variance explained

Overfitting and Underfitting

A good model:

- Learns patterns from training data
- Generalizes well to **new (unseen)** data
- Makes accurate predictions — not just memorizing

What Is Overfitting?

- **Overfitting** = The model **learns too much**, including noise and outliers in the training data
- **Too complex** — tries to fit everything perfectly
- Performs well on **training data**, but poorly on **test data**

Signs of Overfitting:

	Training Accuracy	Test Accuracy
Overfitted	Very High (~100%)	Low (~60–70%)

What Is Underfitting?

- **Underfitting** = The model d**oesn't learn enough** from the training data
- **Too simple** — can't capture patterns
- Performs poorly on both training and test data

Signs of Underfitting:

	Training Accuracy	**Test Accuracy**
Underfitted	Low (~60%)	Low (~60%)

How to Fix Underfitting

Technique	Description
Use more complex model	Try decision trees, ensembles, etc.
Add features (feature engineering)	Give more useful info to the model
Reduce regularization	Too much penalty can cause underfitting
Train longer	Let model learn more patterns

How to Fix Overfitting

Technique	Description
Use simpler model	Avoid high-complexity algorithms
Regularization	Adds a penalty for large weights
More training data	Helps generalize better
Early stopping	Stop training before it overlearns
Cross-validation	Helps detect overfitting early

8. Deployment

- **Model deployment** is the process of taking a machine learning model that has been trained and tested, and making it available for real-world use.

Once your model makes accurate predictions, you want it to:

- Run outside your notebook
- Be used by other people (via a website app, or system)

Components of Model Deployment

Component	Role
Trained model file	The .pkl, .joblib, or .h5 model file
Prediction script	A Python script that loads the model & runs predictions
API/Interface	Allows users to interact (e.g., web form, chatbot)
Host environment	Where the model runs — server, cloud, browser

8. Post-Deployment: Monitor & Maintain

Deployment isn't the end — you must:

- Monitor model performance (accuracy may drop over time)
- Handle user feedback and bugs
- Update model with new data (retraining)

Cross-Validation Techniques

- K-Fold
- Stratified K-Fold
- Leave-One-Out (LOO)

Further Reading and Resources

- Hands-On ML with Scikit-Learn & TensorFlow – Aurélien Géron
- Pattern Recognition and Machine Learning – Christopher Bishop
- Online resources and ML repositories

Thank you