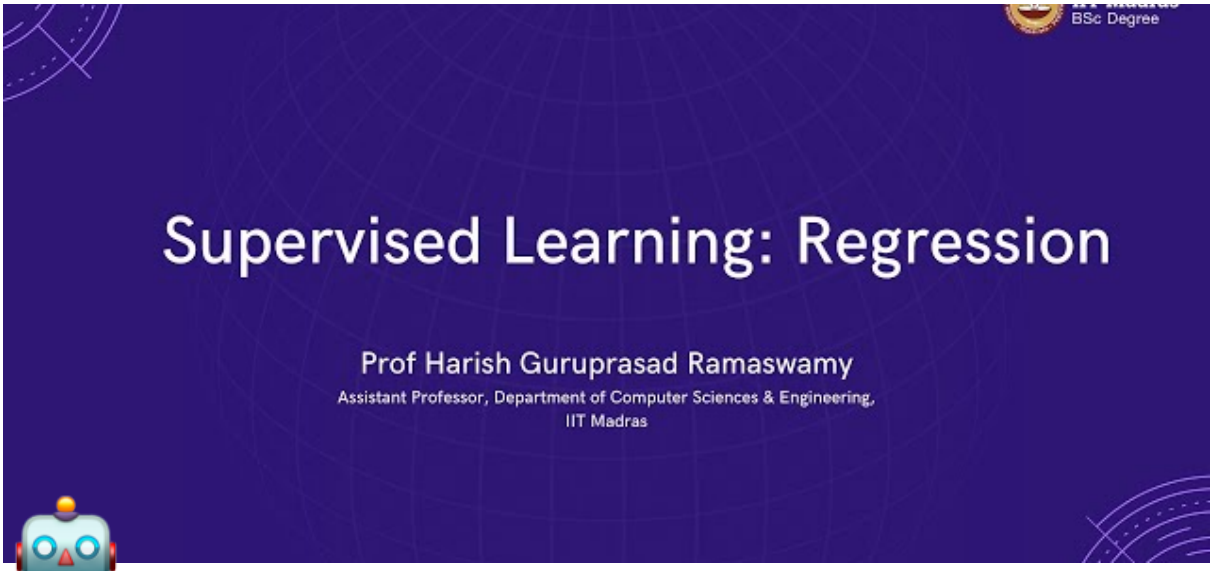


Machine Learning - Module 1: Introduction to Supervised Learning and Linear Regression

Machine learning enables computers to learn from data and make predictions without fully understanding the underlying data-generating process.

Key Ideas

Machine Learning (ML): Learn models from data to predict unseen data.

Supervised Learning: Data consists of input-label pairs (x, y). Goal: learn the mapping:

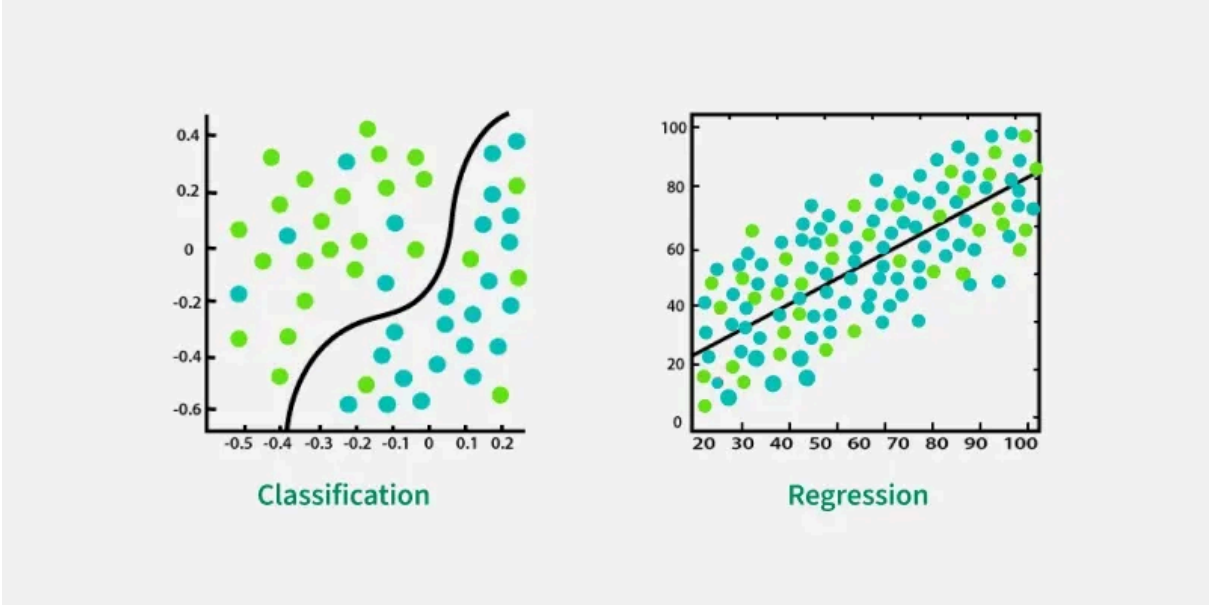
$$\hat{y} = f(\Omega, \mathbf{x})$$

where **Ω are the model parameters**.

Task Types

- **Regression**: y is continuous

    - Example: Predict house prices from area, number of rooms, location
- **Classification**: y is discrete
    - Example: Detect spam from email content



Components of the Framework

1. Inputs

Features **x** and labels **y**

Example: 10,000 clothing images 28×28, 10 class labels

2. Model

Mapping f(Ω, x)

Example: Logistic Regression, 2-layer MLP

3. Loss

J(y, ŷ) measures error between predictions and labels

Example: MSE for regression, Cross-Entropy for classification

4. Optimization

Update parameters Ω to reduce loss

Gradient Descent:

$$\Omega \leftarrow \Omega - \eta \nabla_{\Omega} J$$

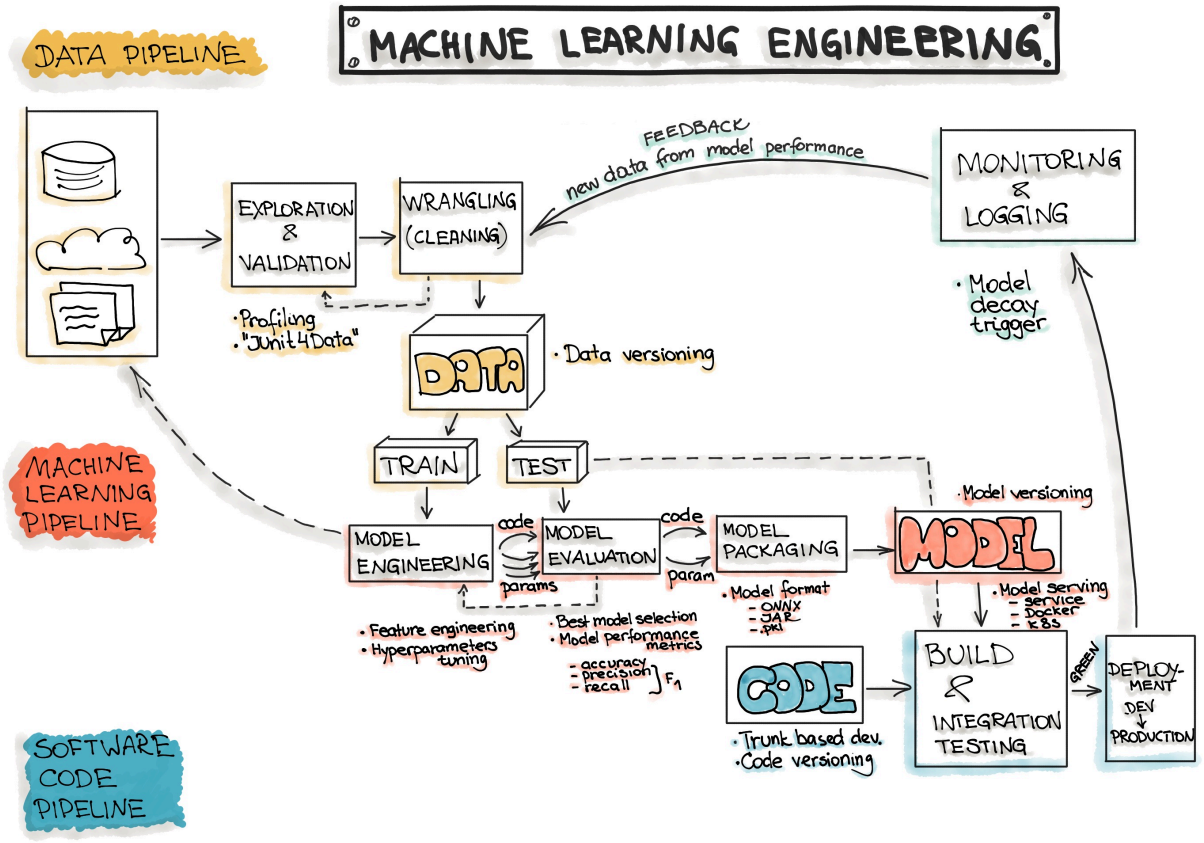
5. Evaluation

Metrics such as **MSE, Accuracy, Precision, Recall, F1-score**

6. Hyperparameters

Parameters not learned from data

Example: learning rate, number of epochs, architecture, regularization



Core Formulas

Prediction

$$\hat{y} = f(\Omega, \mathbf{x})$$

Parameters:

- $\mathbf{x} = [x_1, x_2, \ldots, x_n]$: input features

- **Ω** = [w_1, ..., w_n, b]: weights and bias

Purpose: Compute predicted value

Linear Regression Example:

$$\hat{y} = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

Loss Functions

Mean Squared Error (MSE)

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

Purpose: Measure average squared deviation between predictions and labels

Example: y=[3,5,2], ŷ=[2,7,2] ⇒ MSE ≈ 1.667

Mean Absolute Error (MAE)

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

Purpose: Measure average absolute deviation

Sigmoid Function

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Purpose: Convert logit to probability in (0,1)

Example: z=0 ⇒ σ(0)=0.5

Binary Cross-Entropy (BCE)

$$L = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log \hat{p}_i + (1 - y_i) \log (1 - \hat{p}_i) \right]$$

Purpose: Loss for binary classification

Example: y=1, p̂=0.9 ⇒ L ≈ 0.105

Softmax Function

$$\text{softmax}(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

Purpose: Probability for multiple classes

Example: z=[1,2,3] ⇒ p̂ ≈ [0.09, 0.24, 0.67]

Categorical Cross-Entropy (CCE)

$$L = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{C} y_{i,c}\log \hat{p}_{i,c}$$

Purpose: Loss for multi-class classification

Example: label [0,1,0], prediction [0.1,0.7,0.2] → L ≈ 0.357

Classification Metrics

Accuracy

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

Example: TP=40, TN=50, FP=5, FN=5 → 0.90

Precision

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

Example: TP=45, FP=15 → 0.75

Recall

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

Example: TP=40, FN=5 → 0.889

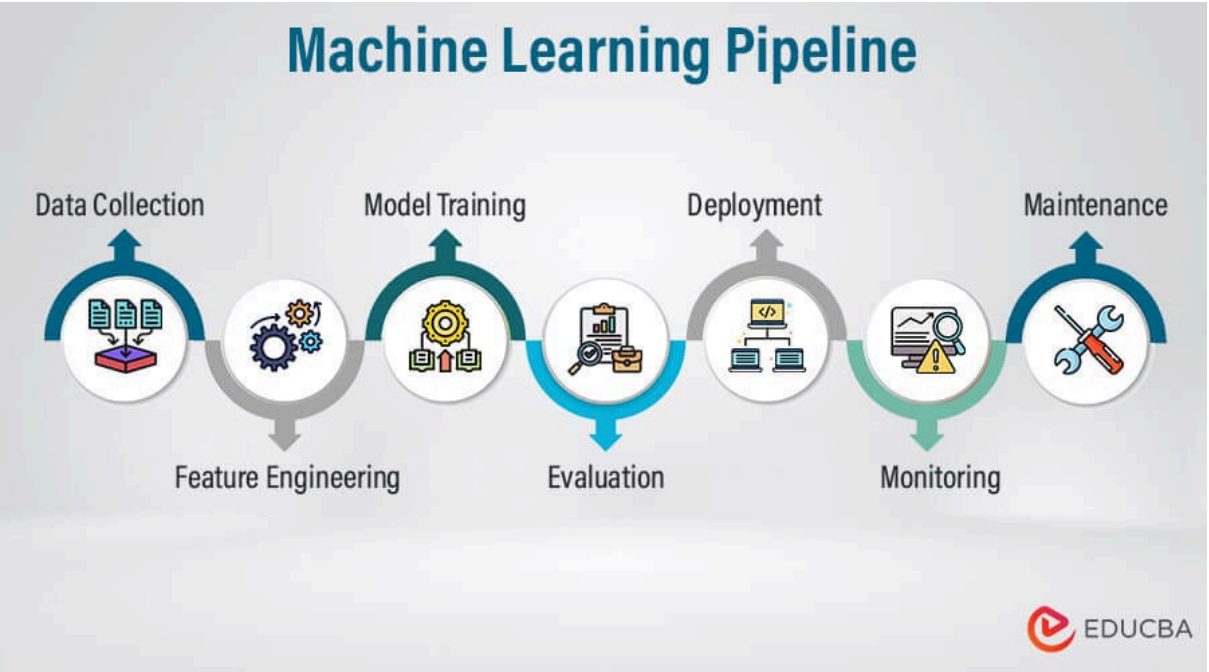
F1-score

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

Example: Precision=0.75, Recall=0.9 → 0.818

Workflow (Pipeline)

1. Collect and preprocess data
2. Split into train, validation, test
3. Feature preprocessing: scaling, encoding
4. Choose model and loss
5. **Train with an optimizer**
6. **Tune hyperparameters** (cross-validation)
7. Evaluate with metrics
8. Deploy or interpret



💡 **Example: Churn Prediction**

- Scale numeric columns, one-hot categorical columns
- Split 70% train, 15% val, 15% test
- Logistic Regression with LBFGS
- Tune C with GridSearchCV
- Evaluate AUC-ROC and F1
- Deploy as API

Hyperparameters

- **Learning rate (η)**: Step size of the optimizer
- **Batch size**: Number of samples per gradient update
- **Epochs**: Number of training iterations
- **Regularization**: L2/weight decay
- **Optimizer**: SGD, Adam, AdamW
- **Class weights & thresholding**: Handle imbalance
- **Normalization & initialization**: Normalize features, BatchNorm, Xavier/Kaiming

---

Code Examples (Python)

Regression

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.random.rand(200,3)
true_w = np.array([3.0,-2.0,1.0])
y = X @ true_w + 0.5 * np.random.randn(200)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

```
model = LinearRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Classification

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

X, y = make_classification(n_samples=500, n_features=10, n_classes=2)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

clf = LogisticRegression(max_iter=1000)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
```

PyTorch Skeleton

```
import torch
from torch import nn, optim

# dataset, model, loss, optimizer
# example:
# model = nn.Linear(input_dim, 1)
# loss_fn = nn.MSELoss()
# optimizer = optim.Adam(model.parameters(), lr=0.001)
# train loop ...
```

Quick Notes

- Feature scaling is important for Linear models, SVM
- Stratified split for classification imbalance
- Use Precision, Recall, F1, ROC-AUC for imbalanced targets
- Fix seed to reproduce results