

# Data Foundation & The ML Framework

Understanding Data and How Machines Learn

Nipun Batra | IIT Gandhinagar

# Recap: What Did We See Last Week?

## AI is everywhere:

- AlphaFold solving protein folding
- AI writing code, music, art
- Self-driving cars, medical diagnosis

## The key insight:

Machine Learning = Learning patterns from DATA

Today: What IS this data? How does learning actually work?

# Part 1: The ML Framework

How Machines Learn

# The Big Question

Every AI/ML system answers **one fundamental question**:

**"Given some input, what should the output be?"**

Input	Output	System
Email text	Spam or Not Spam	Spam Filter
Image	"Cat" or "Dog"	Image Classifier
"The capital of France is"	"Paris"	Language Model
House features	Price (\$)	Price Predictor

# Traditional Programming vs ML

## Traditional Programming:

Rules + Data → Output

*Human writes explicit rules*

## Machine Learning:

Data + Desired Outputs → Rules (Model)

*Computer learns rules from examples!*

# Example: Spam Detection

## Traditional Approach:

```
if "FREE" in email:  
    return SPAM  
if "winner" in email:  
    return SPAM  
if "click here" in email:  
    return SPAM  
# ... 1000 more rules
```

*What about "Fr33" or "w1nner"?*

## ML Approach:

```
# Just give it examples!  
X = [email1, email2, ...]  
y = [spam, not_spam, ...]  
  
model.fit(X, y)  
  
# Now it learns patterns  
model.predict(new_email)
```

*It learns to generalize!*

# Three Learning Paradigms

Based on **what information you give** the model:

Paradigm	What You Provide	Goal
<b>Supervised</b>	Inputs + Correct Outputs	Learn to predict outputs
<b>Unsupervised</b>	Inputs only (no labels)	Find patterns/structure
<b>Reinforcement</b>	Environment + Rewards	Learn to maximize reward

This course focuses on **supervised learning** — it's the most common!

# Supervised Learning: Two Types

It depends on **what you're predicting**:

If output is...	Task	Example
<b>Category</b> (discrete)	Classification	Is it spam? Yes/No
<b>Number</b> (continuous)	Regression	What's the price? \$350,000

# Classification Examples

Task	Input	Output Classes
Spam Detection	Email text	Spam, Not Spam
Medical Diagnosis	Symptoms	Disease A, B, C, Healthy
Image Recognition	Photo	Cat, Dog, Bird, ...
Sentiment Analysis	Review text	Positive, Negative, Neutral

# Regression Examples

Task	Input	Output (Number)
House Pricing	Size, location, rooms	\$425,000
Stock Prediction	Historical prices	Tomorrow's price
Age Estimation	Face image	27 years
Weather	Current conditions	Temperature (°C)

# ML Tasks: The Key Question

When facing an ML problem, always ask:

What exactly am I trying to predict?

If you're predicting...	It's called...	Example
A <b>category</b>	Classification	"Is this spam?"
A <b>number</b>	Regression	"What's the price?"
A <b>sequence</b>	Seq2Seq	"Translate to French"
<b>Something new</b>	Generation	"Draw a cat"

# Computer Vision Task Hierarchy

Task	Question	Output
<b>Classification</b>	"What is this?"	One label: "Cat"
<b>Detection</b>	"What + Where?"	Boxes + labels
<b>Segmentation</b>	"Pixel-level what?"	Pixel masks
<b>Pose Estimation</b>	"Where are keypoints?"	Body skeleton

Each level adds more information!

# NLP Task Hierarchy

Task	Input → Output	Example
<b>Classification</b>	Text → Category	Review → Positive/Negative
<b>NER</b>	Text → Tagged words	"Sundar Pichai" → PERSON
<b>Seq2Seq</b>	Sequence → Sequence	English → French
<b>Generation</b>	Prompt → New text	"Write a poem..." → Poem

# Part 2: Understanding Data

The Fuel for Machine Learning

# "Data is the New Oil"

"Data is the new oil. Like oil, data is valuable, but if unrefined it cannot really be used."

— Clive Humby (2006)

Without data, ML is impossible. With bad data, ML is useless.

# What IS Data in ML?

Data = Examples we learn from

Each example has:

- **Features (X):** What we know about it
- **Label (y):** What we want to predict (in supervised learning)

```
# Example: House price prediction
Features (X) = [sqft, bedrooms, bathrooms, location]
Label (y) = price
```

# Features and Labels: House Example

sqft	beds	baths	garage	price (\$)
1500	3	2	Yes	300,000
2000	4	3	Yes	450,000
1200	2	1	No	200,000
1800	3	2	Yes	350,000

**Features (X):** sqft, beds, baths, garage

**Label (y):** price

# Features and Labels: Spam Example

has_FREE	has_winner	num_links	from_known	is_spam
1	1	5	0	Yes
0	0	1	1	No
1	0	3	0	Yes
0	0	0	1	No

**Features (X):** has\_FREE, has\_winner, num\_links, from\_known

**Label (y):** is\_spam

# Types of Features

Type	Description	Example
<b>Numerical</b>	Continuous numbers	Age: 25, Price: \$50.99
<b>Categorical</b>	Discrete categories	Color: Red/Blue/Green
<b>Binary</b>	Two values	Has garage: Yes/No
<b>Ordinal</b>	Ordered categories	Size: S < M < L < XL
<b>Text</b>	Raw text	Review: "Great product!"

# Types of Data

## Structured (Tabular)

- Rows = examples
- Columns = features
- Easy to work with
- Example: CSV files, databases

## Images

- Pixels as features
- Shape: Height  $\times$  Width  $\times$  Channels
- Example: MNIST digits

## Text

- Words/tokens as features
- Variable length sequences
- Example: Reviews, documents

## Time Series

- Ordered by time
- Temporal patterns matter
- Example: Stock prices, weather

# How Much Data Do You Need?

## Rule of thumb:

- Simple models: 100s of examples
- Complex models: 1000s-10000s of examples
- Deep learning: 100000s+ examples

More data almost always helps, but quality matters more than quantity!

# Data Quality Issues

Problem	What It Means	Impact
Missing values	Empty cells in data	Model can't learn from incomplete examples
Outliers	Extreme unusual values	Can skew model predictions
Noise	Random errors in data	Harder to find true patterns
Imbalanced classes	99% one class, 1% other	Model ignores rare class
Duplicates	Same example multiple times	Overfitting to duplicates

# The Data Lifecycle

1. COLLECT → Gather raw data from sources
2. CLEAN → Handle missing values, fix errors
3. EXPLORE → Understand patterns, visualize
4. TRANSFORM → Scale, encode, engineer features
5. SPLIT → Separate into train/test sets
6. USE → Feed into ML model

# Part 3: Train/Test Split

The Most Important Concept

# The Problem with Memorization

Imagine studying for an exam:

## Strategy A: Memorize answers

- Learn exact answers to practice questions
- Fail on any new question

## Strategy B: Learn concepts

- Understand underlying principles
- Apply to any question

We want our ML models to be like Strategy B — **generalize**, not memorize!

# What is Overfitting?

**Overfitting** = Model memorizes training data instead of learning patterns

Training Data	Test (New) Data
99% accuracy	60% accuracy

The model learned the noise, not the signal!

# How Do We Detect Overfitting?

**Solution:** Hold out some data the model never sees during training

```
from sklearn.model_selection import train_test_split

# Split: 80% train, 20% test
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

# The Train/Test Split

ALL YOUR DATA	
Training Set (80%) Model learns from this	Test (20%) Evaluate

**Training set:** Model learns patterns

**Test set:** We evaluate if it generalized

# Why This Works

1. Model trains ONLY on training data
2. Model has NEVER seen test data
3. Good performance on test = model generalized
4. Bad performance on test = model overfit

The test set simulates "real world" data the model will encounter later.

# The Golden Rule

**NEVER use test data for training or model selection!**

If you "peek" at test data:

- You'll unconsciously tune to it
- Your accuracy estimate will be wrong
- Model will fail in production

# Common Mistakes

Mistake	Why It's Bad
Training on all data	Can't detect overfitting
Looking at test data early	Contaminates your estimate
Using test data to tune hyperparameters	Same as training on it
Small test set	Unreliable accuracy estimate

# Part 4: The ML Recipe

Putting It All Together

# The Universal ML Recipe

Every ML project follows these steps:

1. GET DATA → Collect examples
2. PREPARE DATA → Clean, split into train/test
3. CHOOSE MODEL → Pick an algorithm
4. TRAIN → Model learns from training data
5. EVALUATE → Test on held-out data
6. DEPLOY → Use in the real world

# Step 1: Get Data

```
# Your data: features (X) and labels (y)
X = [[1500, 3, 2],    # House 1: sqft, beds, baths
      [2000, 4, 3],    # House 2
      [1200, 2, 1]]    # House 3

y = [300000, 450000, 200000] # Prices (labels)
```

**More data = better model** (usually!)

## Step 2: Prepare Data

```
from sklearn.model_selection import train_test_split

# Split 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
)
```

### Why split?

- Train on 80%, test on 20%
- Simulate "unseen" data
- Detect overfitting

## Step 3: Choose a Model

Task	Simple Models	Complex Models
Classification	Logistic Regression, Decision Tree	Neural Network, Random Forest
Regression	Linear Regression	XGBoost, Neural Network
Clustering	K-Means	DBSCAN, Hierarchical

**Start simple, add complexity only if needed!**

## Step 4: Train (Fit)

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train, y_train) # Learn from data!
```

### What happens inside:

- Model finds patterns in training data
- Adjusts internal parameters to minimize error
- "Learns" the mapping from X to y

## Step 5: Evaluate

```
# Make predictions on test data
predictions = model.predict(X_test)

# Compare to actual values
from sklearn.metrics import mean_squared_error
error = mean_squared_error(y_test, predictions)
print(f"Average error: {error:.0f}")
```

**Key question:** How well does it work on data it *hasn't seen*?

## Step 6: Deploy

```
# New house comes in
new_house = [[1800, 3, 2]] # sqft, beds, baths

# Predict!
predicted_price = model.predict(new_house)
print(f"Predicted price: ${predicted_price[0]:.0f}")
```

The model is now useful in the real world!

# The sklearn API Pattern

```
# This pattern works for 50+ different algorithms!

from sklearn.some_module import SomeModel

model = SomeModel()           # 1. Create
model.fit(X_train, y_train)    # 2. Train
predictions = model.predict(X_test) # 3. Predict
score = model.score(X_test, y_test) # 4. Evaluate
```

Learn this pattern once, use it everywhere!

# Quick Demo: Classification

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Load data
iris = load_iris()
X, y = iris.data, iris.target

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

# Train
model = DecisionTreeClassifier()
model.fit(X_train, y_train)

# Evaluate
accuracy = model.score(X_test, y_test)
print(f"Accuracy: {accuracy:.1%}") # ~95-97%
```

# Quick Demo: Regression

```
from sklearn.datasets import load_diabetes
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

# Load data
diabetes = load_diabetes()
X, y = diabetes.data, diabetes.target

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

# Train
model = LinearRegression()
model.fit(X_train, y_train)

# Evaluate
r2 = model.score(X_test, y_test)
print(f"R² Score: {r2:.3f}") # ~0.4-0.5
```

# Part 5: What's Next

Course Roadmap

# Your Journey Through AI

Week	Topic	Big Question
1	Introduction (Videos)	What can AI do today?
2	Data Foundation	What makes good data?
3	Supervised Learning	How do we predict?
4	Model Selection	How do we choose models?
5	Neural Networks	What makes deep learning special?
6	Computer Vision	How do machines see?
7	Language Models	How do machines understand text?
8	Generative AI	How do machines create?

# What You'll Build

Lab	What You'll Create
1 - 2	Your first ML models with sklearn
3	PyTorch basics, simple neural network
4 - 5	Build a small language model from scratch
6 - 7	Object detection with YOLO
8	Fine-tune an LLM

**By the end:** You'll understand AND build AI systems!

# Key Takeaways

1. **ML learns patterns from data** — not explicit rules
2. **Data = Features (X) + Labels (y)**
3. **Train/Test split is essential** — always hold out test data
4. **The sklearn pattern:**
  - `model.fit(X_train, y_train)` → Train
  - `model.predict(X_test)` → Predict
5. **Classification vs Regression:** Category or Number?

# Ready to Build!

Next: Supervised Learning Deep Dive

**Lab this week:** Your first ML models with sklearn

*"In God we trust. All others must bring data."*

— W. Edwards Deming

Questions?