

Lecture 3: kNN Implementation & Real Datasets

ECE 2410 – Introduction to Machine Learning

Farzad Farnoud

University of Virginia

Spring 2026

Outline

- 1 Review & Recap
- 2 Training and Test Sets
- 3 Data Normalization
- 4 Evaluating Classification
- 5 Images as Vectors
- 6 Summary

Key concepts from L02:

- k-Nearest Neighbors algorithm
- Distance metrics (L2, L1, Lp)
- `argmin` – finding the minimizer
- NumPy basics for ML

The kNN idea:

- 1 Find the k closest training points
- 2 Take a majority vote
- 3 Predict that class!

Today's Goals

Moving from Theory to Practice

- 1 **Train/Test Split:** Why we need separate data for evaluation
- 2 **Data Normalization:** Scaling features fairly
- 3 **Evaluation Metrics:** Accuracy, precision, and recall
- 4 **Images as Vectors:** How to apply kNN to MNIST digits



Notebook: Hands-on implementation in Python

kNN: Mathematical Formulation

1-Nearest Neighbor (1-NN)

Given test point \mathbf{x}_{test} and training data $\{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$:

$$\hat{i} = \underset{i \in \{1, \dots, N\}}{\operatorname{argmin}} \|\mathbf{x}_{test} - \mathbf{x}_i\|, \quad \hat{y} = y_{\hat{i}}$$

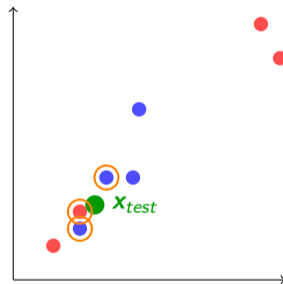
k-Nearest Neighbors (k-NN)

- 1 Find k indices with smallest distances, $\hat{i}_1, \dots, \hat{i}_k$

- 2 Take **majority vote**:

$$\hat{y} = \operatorname{mode}(y_{\hat{i}_1}, \dots, y_{\hat{i}_k})$$

Example: $k = 3$



Votes: 2 blue, 1 red
Predict: Blue

How do We Evaluate Performance?

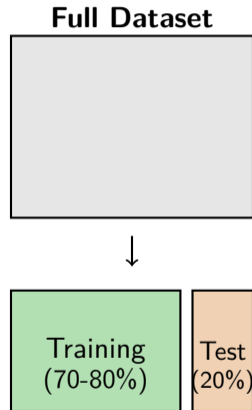
- 🤔 Why do we need to evaluate performance?
- 🤔 Can we evaluate on the same data we trained on?

The Problem:

- If we evaluate on the **same data** we trained on...
 - That's like having exam problems exactly the same as in-class examples
 - 1-NN will **always** get 100% accuracy! (Why?)
 - This doesn't tell us how well we'll do on **new data**

The Solution:

- Split data into **training** and **test** sets
- Train on one, evaluate on the other



The Golden Rule

Never Use Test Data During Training!

- The test set simulates **unseen, real-world data**
- If you peek at the test set, your accuracy estimate will be overly optimistic

Training Phase

Use training data only
Learn patterns & fit model

Evaluation Phase

Apply to test data
Measure real performance

Activity 1: Python Concepts & Data Preparation



Open the Notebook

L03-2026-01-kNN-Implementation.ipynb

Complete these sections:

① Python Concepts You'll Need Today

- Random seed, multiple returns, axis, reshape, views vs copies

② Section 1.1: Load the NBA dataset

③ Section 1.2: Data preprocessing

④ Section 1.3: Train/test split



Take about 10 minutes, then we'll continue with normalization.

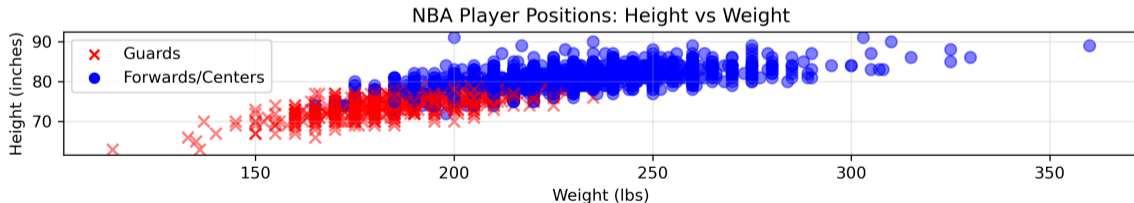
Why Normalize?

The Problem:

- Features can have very different scales
- **NBA Example:**
 - Height: 70–85 inches
 - Weight: 180–280 lbs
- Weight differences dominate distance!

Solution: Normalize features

- Scale all features to similar ranges
- Each feature contributes fairly



Normalization Methods

1. Min-Max Normalization

Scale to $[0, 1]$:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

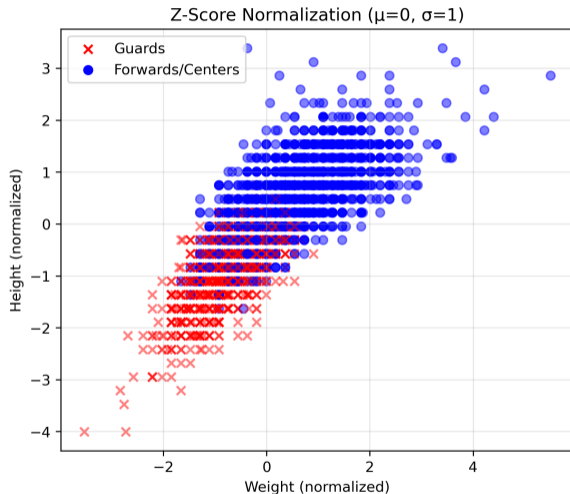
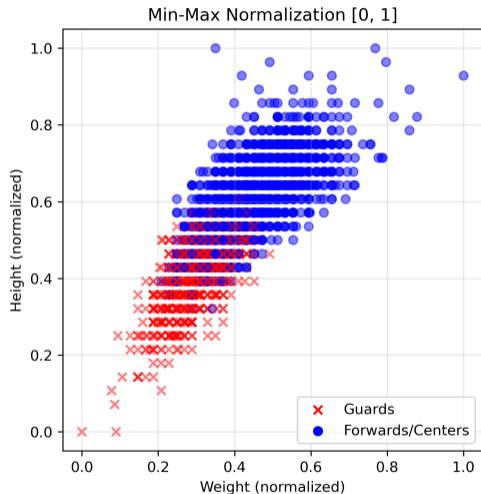
2. Z-Score (Standardization)

Scale to mean 0, std 1:

$$x' = \frac{x - \mu}{\sigma}, \quad \mu = \frac{1}{N} \sum_{i=1}^N x_i, \quad \sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$$

- **Min-Max:** Good when you know the range
- **Z-Score:** More robust to outliers. Replace $x - \mu$ with $x - x_{\min}$ to keep values positive.

Effect of Normalization on NBA Data



Both methods: features now on **comparable scales**

Important: Fit on Training Data Only!

! Golden Rule Corollary

Compute normalization parameters (μ , σ , min, max) from **training data only**!

Why?

- Test data simulates unseen data
- In production, you won't have test statistics
- Using test stats = data leakage

Correct Workflow

- 1 Compute μ_{train} , σ_{train} from training set
- 2 Normalize training set using these values
- 3 Normalize test set using **the same** μ_{train} , σ_{train}

Measuring Performance: Accuracy

Definition

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Example:

- 100 test samples
- kNN predicts 85 correctly
- Accuracy = 85%

Limitations

- Accuracy can be misleading with **imbalanced classes!**
(99% of data is class A → predicting “A” always gives 99% accuracy)
- Not all mistakes are the same!

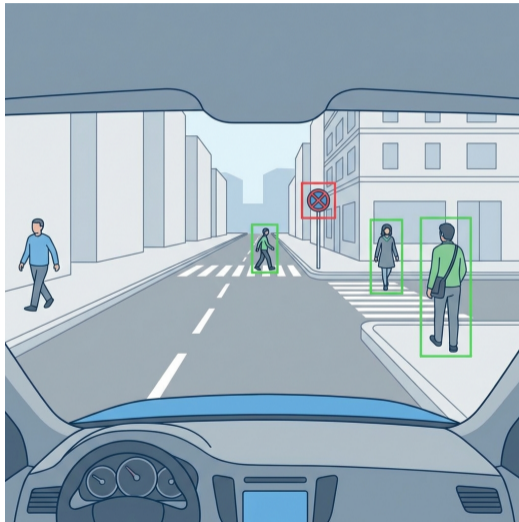
Not All Mistakes Are the Same: Pedestrian Detection

Self-Driving Car Task:

- Detect pedestrians to avoid collisions
- **Positive** = “Pedestrian detected” → **brake**
- **Negative** = “No pedestrian” → continue driving

Four Possible Outcomes:

- **TP**: Pedestrian present, detected
- **FP**: No pedestrian, but detected
 - False alarm: bad but not terrible
- **TN**: No pedestrian, not detected
- **FN**: Pedestrian present, **not detected**
 - **Extremely dangerous** 🦴



Precision and Recall

Why Accuracy Isn't Enough

- Less harmful to have false alarms than to miss pedestrians
- Unbalanced classes still a problem: What happens if most images don't have pedestrians?

Precision

Of all **predicted positives**, how many are correct?

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

High precision = few false alarms

Recall (Sensitivity)

Of all **actual positives**, how many did we find?

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

High recall = few missed pedestrians

⚠ For safety-critical applications: **recall** is often more important!

TO COMPLETE YOUR REGISTRATION, PLEASE TELL US
WHETHER OR NOT THIS IMAGE CONTAINS A STOP SIGN:



NO YES

ANSWER QUICKLY—OUR SELF-DRIVING
CAR IS ALMOST AT THE INTERSECTION.

SO MUCH OF "AI" IS JUST FIGURING OUT WAYS
TO OFFLOAD WORK ONTO RANDOM STRANGERS.

Source: xkcd.com/1897

Activity 2: Normalization & kNN on NBA Data



Continue in the Notebook

L03-2026-01-kNN-Implementation.ipynb

Complete these sections:

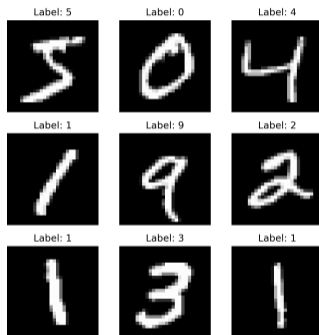
- 1 **Section 1.4:** Feature normalization (Z-score)
- 2 **Section 1.5:** k-NN Implementation
- 3 **Section 1.6:** Compute accuracy, precision, and recall



Take about 15 minutes, then we'll discuss images.

Can kNN Classify Images?

Question: Can we use kNN to recognize handwritten digits?



Let's think about it... What are some reasons it could (not) work?...
What about other types of images?

Can kNN Classify Images?

Question: Can we use kNN to recognize handwritten digits?

Why It Might Work (for MNIST)

- Similar digits *should* look similar
- MNIST is **well-controlled**:
 - Same size (28×28)
 - Centered, normalized
 - Clean grayscale
- Limited variation in styles

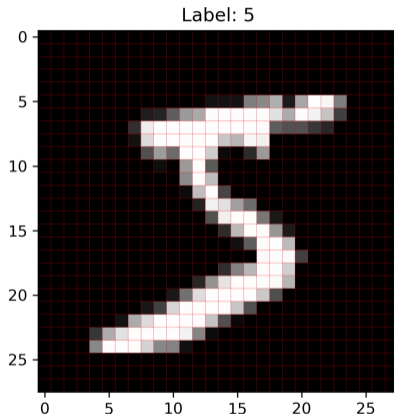
Challenges & Limitations

- Variation in handwriting:
 - Different slants, thickness
 - Small shifts break similarity
- **Other images?** Much harder!
 - Faces: pose, lighting, expression
 - Objects: scale, background, occlusion
- Slow: compare to *all* training images

How Computers See Images

Grayscale Image:

- 2D array of pixel intensities
- Values range from 0 (black) to 255 (white)
- MNIST: 28×28 pixels



A single MNIST digit (28×28)

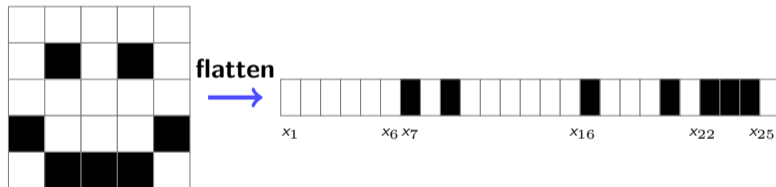
Vectorization: Flattening Images

Key Idea

To use kNN on images, we need to represent each image as a **vector**.

5×5 Image

25-element Vector



- Read pixels row by row \rightarrow create 1D vector
- MNIST: $28 \times 28 = 784$ dimensions
- Each image becomes a point in \mathbb{R}^{784}

Feature Matrix

Organizing All Images

Stack all flattened images as **rows** of a matrix:

$$\mathbf{X} = \begin{bmatrix} - & \mathbf{x}_1^T & - \\ - & \mathbf{x}_2^T & - \\ & \vdots & \\ - & \mathbf{x}_N^T & - \end{bmatrix} \in \mathbb{R}^{N \times 784}$$

- N = number of images (samples)
- 784 = number of features (pixels)
- Row i is the flattened representation of image i



In Python: `X[i, :]` gives the i -th image as a vector

Activity 3: MNIST Digit Classification



Continue in the Notebook

L03-2026-01-kNN-Implementation.ipynb

Complete these sections:

- 1 **Section 2.1:** Visualize MNIST digits (vector \rightarrow image)
- 2 **Section 2.2:** Apply k-NN to MNIST
- 3 **Section 2.3:** View misclassified examples



See which digits confuse the classifier!

Complete and submit online by 11:59 PM today!

Key Takeaways

- 1 **Train/Test Split:** Essential for honest evaluation
 - Never leak test data into training!
- 2 **Normalization:** Scale features to comparable ranges
 - Compute stats from training set only
- 3 **Images → Vectors:** Flatten 2D images to 1D vectors
 - $28 \times 28 = 784$ dimensions for MNIST
- 4 **Precision & Recall:** Better than accuracy for imbalanced classes

Next Time: Bias-Variance Tradeoff & Cross-Validation