KNN error rate explanation

The **error rate in K-Nearest Neighbour (KNN)** refers to how often the classifier makes a wrong prediction compared to the true class labels.

🔹 Formula for Error Rate

$$\text{Error Rate} = \frac{\text{Number of Incorrect Predictions}}{\text{Total Predictions}}$$

Accuracy is just the complement:

$$\text{Accuracy} = 1 - \text{Error Rate}$$

🔹 Factors Affecting KNN Error Rate

1. **Value of k:**
   - Small k (like $k = 1$): very sensitive to noise → high variance, low bias.
   - Large k: smoother decision boundary but may misclassify minority classes → higher bias, lower variance.
2. **Distance Metric:**
   - Euclidean, Manhattan, Minkowski, cosine similarity, etc. Choice affects classification.
3. **Feature Scaling:**
   - If features are not scaled (normalization/standardization), distance can be misleading → higher error.
4. **Data Quality:**
   - Noisy, imbalanced, or irrelevant features increase error.

🔹 Example

Suppose we test KNN on 20 samples:

- Correct predictions = 16
- Wrong predictions = 4

$$\text{Error Rate} = \frac{4}{20} = 0.20 \quad (20\%)$$

$$\text{Accuracy} = 1 - 0.20 = 0.80 \quad (80\%)$$

Great — here's a clear, **step-by-step numerical KNN example** (K = 3) with all distance arithmetic shown digit-by-digit and final error rate.

Dataset

Training points (label):

- $T_1 = (1, 1)$ — A
- $T_2 = (2, 1)$ — A
- $T_3 = (1, 2)$ — A
- $T_4 = (4, 4)$ — B
- $T_5 = (5, 4)$ — B
- $T_6 = (4, 5)$ — B

Test points (true label):

1. $X_1 = (1.5,\ 1.5)$ — true A
2. $X_2 = (3,\ 3)$ — true B
3. $X_3 = (4.2,\ 4.1)$ — true B

We use **Euclidean distance** and $k = 3$. Decision rule: majority vote among the 3 nearest neighbors.

Distance formula

For two points $P = (x_p, y_p)$ and $Q = (x_q, y_q)$:

$$d(P, Q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$$

I show full arithmetic for each test point.

Test point 1: $X_1 = (1.5,\ 1.5)$

Distances to training points:

1. To $T_1 = (1, 1)$:
- x-diff: $1.5 - 1 = 0.5$ → square: $0.5^2 = 0.25$
- y-diff: $1.5 - 1 = 0.5$ → square: $0.5^2 = 0.25$
- sum: $0.25 + 0.25 = 0.5$
- distance: $\sqrt{0.5} \approx 0.70710678$
2. To $T_2 = (2, 1)$:
- x-diff: $1.5 - 2 = -0.5$ → square: $(-0.5)^2 = 0.25$
- y-diff: $1.5 - 1 = 0.5$ → square: 0.25
- sum: $0.25 + 0.25 = 0.5$
- distance: $\sqrt{0.5} \approx 0.70710678$
3. To $T_3 = (1, 2)$:
- x-diff: $1.5 - 1 = 0.5$ → square: 0.25
- y-diff: $1.5 - 2 = -0.5$ → square: 0.25
- sum: $0.25 + 0.25 = 0.5$
- distance: $\sqrt{0.5} \approx 0.70710678$
4. To any B point (example $T_4 = (4, 4)$):
- x-diff: $1.5 - 4 = -2.5$ → square: 6.25
- y-diff: $1.5 - 4 = -2.5$ → square: 6.25
- sum: $6.25 + 6.25 = 12.5$
- distance: $\sqrt{12.5} \approx 3.53553391$

Sorted nearest 3 distances: three A points at ≈ 0.7071.

→ **Nearest 3 are all class A** → prediction: **A**.

True label: A → **correct**.

Test point 2: $X_2 = (3,\ 3)$

Distances to training points (showing those that matter):

1. To $T_4 = (4, 4)$:
- x-diff: $3 - 4 = -1$ → square: 1
- y-diff: $3 - 4 = -1$ → square: 1

- sum: $1 + 1 = 2$
- distance: $\sqrt{2} \approx 1.41421356$

2. To $T_2 = (2, 1)$:

- x-diff: $3 - 2 = 1$ → square: 1
- y-diff: $3 - 1 = 2$ → square: 4
- sum: $1 + 4 = 5$
- distance: $\sqrt{5} \approx 2.23606798$

3. To $T_3 = (1, 2)$:

- x-diff: $3 - 1 = 2$ → square: 4
- y-diff: $3 - 2 = 1$ → square: 1
- sum: $4 + 1 = 5$
- distance: $\sqrt{5} \approx 2.23606798$

4. To $T_5 = (5, 4)$:

- x-diff: $3 - 5 = -2$ → square: 4
- y-diff: $3 - 4 = -1$ → square: 1
- sum: $4 + 1 = 5$
- distance: $\sqrt{5} \approx 2.23606798$

5. To $T_1 = (1, 1)$:

- x-diff: $3 - 1 = 2$ → square: 4
- y-diff: $3 - 1 = 2$ → square: 4
- sum: $4 + 4 = 8$
- distance: $\sqrt{8} \approx 2.82842712$

Sorted by distance (small → large):

1. $T_4 = (4, 4)$ — B at ≈ 1.4142
2. $T_2 = (2, 1)$ — A at ≈ 2.2361
3. $T_3 = (1, 2)$ — A at ≈ 2.2361
   (others at same ~2.2361 but above are enough for k=3)

Nearest 3 labels: B, A, A → majority is **A** (2 votes vs 1).
→ **Predicted A**. True label: **B** → **incorrect**.

(Note: several training points tie at distance $\sqrt{5} \approx 2.2361$. We used the three smallest distances — any tie-breaking rule that picks two A neighbors and one B neighbor yields the same majority A here.)

Test point 3: $X_3 = (4.2,\ 4.1)$

Distances:

1. To $T_4 = (4, 4)$:
- x-diff: $4.2 - 4 = 0.2$ → square: 0.04
- y-diff: $4.1 - 4 = 0.1$ → square: 0.01
- sum: $0.04 + 0.01 = 0.05$
- distance: $\sqrt{0.05} \approx 0.22360680$
2. To $T_5 = (5, 4)$:
- x-diff: $4.2 - 5 = -0.8$ → square: 0.64
- y-diff: $4.1 - 4 = 0.1$ → square: 0.01
- sum: $0.64 + 0.01 = 0.65$
- distance: $\sqrt{0.65} \approx 0.80622577$
3. To $T_6 = (4, 5)$:
- x-diff: $4.2 - 4 = 0.2$ → square: 0.04
- y-diff: $4.1 - 5 = -0.9$ → square: 0.81
- sum: $0.04 + 0.81 = 0.85$
- distance: $\sqrt{0.85} \approx 0.92195445$

Other training points are much farther (> 3.8), so nearest three are T_4, T_5, T_6 — all class **B**.

→ **Predicted B**. True label: B → **correct**.

Summary of predictions

Test point	True label	Predicted	Correct?
$X_1 = (1.5, 1.5)$	A	A	✔
$X_2 = (3, 3)$	B	A	✖
$X_3 = (4.2, 4.1)$	B	B	✔

Correct = 2 / 3

Incorrect = 1 / 3

Error rate

$$\text{Error rate} = \frac{\text{Number of incorrect predictions}}{\text{Total predictions}} = \frac{1}{3} \approx 0.3333 = 33.33\%$$

Accuracy = $1 - 0.3333 = 0.6667 = 66.67\%$.

Notes & tips

- Here $k = 3$ produced one misclassification at X_2. Trying other k values can change the result:
  - $k = 1$ for X_2 would predict B (nearest neighbor T_4), so X_2 would be correct with $k = 1$.
  - Larger k could include more A points and push votes toward A — shows the bias–variance tradeoff.
- Always scale features if they have different units; otherwise distance is dominated by large-range features.
- For ties in votes, common tie-breakers: pick label of the closest neighbor among tied classes, or pick at random, or use weighted voting (weights = $1/\text{distance}$).

---

separate **KNN classification** and **KNN regression** in terms of how the **error rate (or error metric)** is calculated.

🔹 1. KNN Classification

In classification, KNN assigns a **class label** (A, B, etc.).
We evaluate using **misclassification rate** (error rate).

$$\text{Error Rate} = \frac{\text{Number of Misclassified Samples}}{\text{Total Samples}}$$

✅ Example:

- Total test points = 50

- Wrong predictions = 8

$$\text{Error Rate} = \frac{8}{50} = 0.16\ (16\%)$$

Other metrics often used in classification:

- **Accuracy = 1 − Error Rate**
- **Precision, Recall, F1-score** (if classes are imbalanced).

🔹 2. KNN Regression

In regression, KNN predicts a **numeric value** (average of neighbors).

Here, "error rate" doesn't apply directly because outputs are continuous.

Instead, we use **error metrics**:

1. **Mean Squared Error (MSE):**

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

2. **Root Mean Squared Error (RMSE):**

$$RMSE = \sqrt{MSE}$$

3. **Mean Absolute Error (MAE):**

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

✅ Example:

True values: $[5, 7, 10]$

Predicted: $[6, 8, 9]$

- Errors: $[-1, -1, 1]$
- MAE = $(1 + 1 + 1)/3 = 1$
- MSE = $(1^2 + 1^2 + 1^2)/3 = 1$
- RMSE = $\sqrt{1} = 1$

🔹 Summary

- **KNN Classification → Error Rate = Misclassification Rate.**

- **KNN Regression → Error measured using MAE, MSE, or RMSE (not % error rate).**

Nice — here's a compact, step-by-step explanation of **precision, recall, and F1-score** with a clear numeric example and notes about averaging for multi-class problems.

Definitions (binary classification)

Let positive = class we care about (e.g., "spam").

- **TP (True Positive)**: predicted positive, actually positive.
- **FP (False Positive)**: predicted positive, actually negative.
- **FN (False Negative)**: predicted negative, actually positive.
- **TN (True Negative)**: predicted negative, actually negative.

Formulas

- **Precision** = $\frac{TP}{TP + FP}$
  (Of the items predicted positive, how many were correct?)
- **Recall** (a.k.a. Sensitivity) = $\frac{TP}{TP + FN}$
  (Of the actual positive items, how many did we catch?)
- **F1-score** = harmonic mean of precision and recall:

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

Numeric example (digit-by-digit)

Suppose on 100 test items we have:

- TP = 40
- FP = 10
- FN = 5

- TN = 45

Precision:

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{40}{40 + 10} = \frac{40}{50} = 0.8 = 80\%$$

Recall:

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{40}{40 + 5} = \frac{40}{45} \approx 0.8888889 \approx 88.89\%$$

F1:

$$F1 = \frac{2 \times 0.8 \times 0.8888889}{0.8 + 0.8888889} = \frac{1.4222222}{1.6888889} \approx 0.84126984 \approx 84.13\%$$

Relationship with error rate / accuracy

- **Accuracy** = $\frac{TP + TN}{TP + FP + FN + TN} = \frac{40 + 45}{100} = 0.85 = 85\%$
- **Error rate** = $1 - \text{Accuracy} = 15\%$
  Precision/recall give class-specific performance which accuracy and error rate can hide (important when classes are imbalanced).

Multi-class averaging

If more than two classes, compute per-class Precision/Recall/F1 treating that class as "positive" and then average:

- **Macro-average**: average of per-class scores (treats all classes equally).
- **Micro-average**: total TP/FP/FN summed over classes, then compute metrics (weights by support / sample count).
- **Weighted macro**: macro average weighted by class support.

Tips

- Use **precision** if false positives are costly (e.g., spam filter that blocks real emails).
- Use **recall** if false negatives are costly (e.g., disease screening).
- Use **F1** when you want a single number balancing precision and recall.

Want me to:

- compute these for a confusion matrix you have (paste numbers), or
- give a tiny Python snippet that prints precision/recall/F1 for your results?

Got it 👍 — you want to see **precision, recall, F1 (and error rate/accuracy) calculated directly from a dataset** rather than just formulas.

Let's do a **small toy dataset** with true labels and predicted labels.

Example Dataset

True labels (y):

$$[A, A, A, B, B, C, C, C, C]$$

Predicted labels ($\hat{y}$):

$$[A, B, A, B, C, C, B, C, A]$$

Step 1. Build Confusion Matrix

True \ Pred	A	B	C
A	2	1	0
B	0	1	1
C	1	1	2

Step 2. Compute Per-class Metrics

🔹 **Class A**

- TP = 2 (A→A)
- FP = 0+1 = 1 (C→A)
- FN = 1 (A→B)
- Precision = $2/(2+1) = 0.667$
- Recall = $2/(2+1) = 0.667$
- F1 = 0.667

🔹 **Class B**

- TP = 1 (B→B)
- FP = 1 (A→B) + 1 (C→B) = 2
- FN = 1 (B→C)
- Precision = $1/(1+2) = 0.333$
- Recall = $1/(1+1) = 0.5$
- F1 = 0.4

🔹 **Class C**

- TP = 2 (C→C)
- FP = 0+1 = 1 (B→C)
- FN = 1 (C→A) + 1 (C→B) = 2
- Precision = $2/(2+1) = 0.667$
- Recall = $2/(2+2) = 0.5$
- F1 = 0.571

Step 3. Average Metrics

- **Macro-average Precision** = (0.667 + 0.333 + 0.667)/3 = 0.556
- **Macro-average Recall** = (0.667 + 0.5 + 0.5)/3 = 0.556
- **Macro-average F1** = (0.667 + 0.4 + 0.571)/3 ≈ 0.546

Step 4. Accuracy & Error Rate

Total correct predictions = $2+1+2=5$

Total = 9

$$\text{Accuracy} = 5/9 \approx 55.6\%$$

$$\text{Error Rate} = 1 - 0.556 = 44.4\%$$

✅ This shows how **precision, recall, F1, accuracy, error rate** are computed directly from a dataset's predictions.
