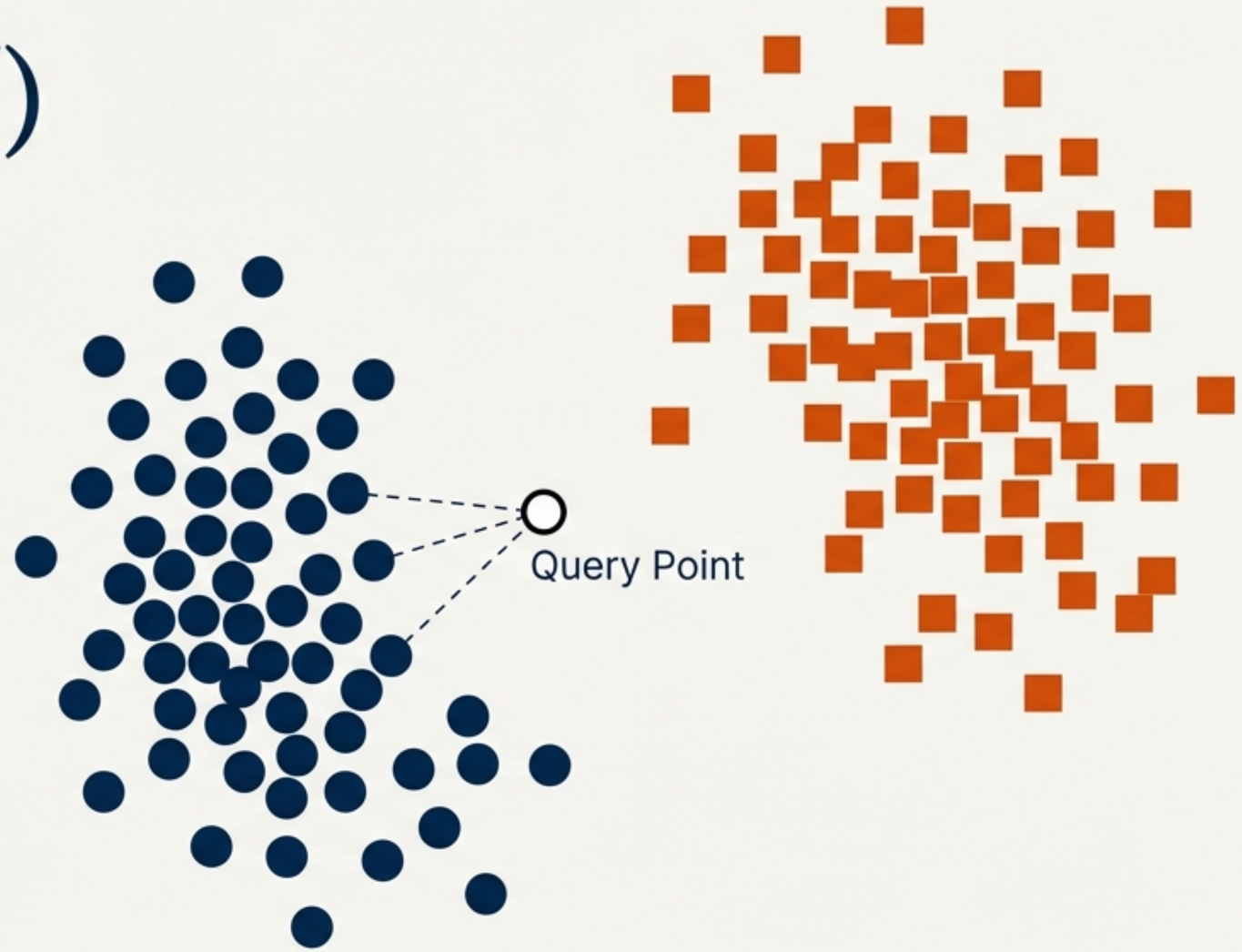
The K-Nearest Neighbors (KNN) Algorithm

From Intuition to Implementation

A comprehensive guide to the logic, mathematics, and Python implementation of KNN for classification tasks.

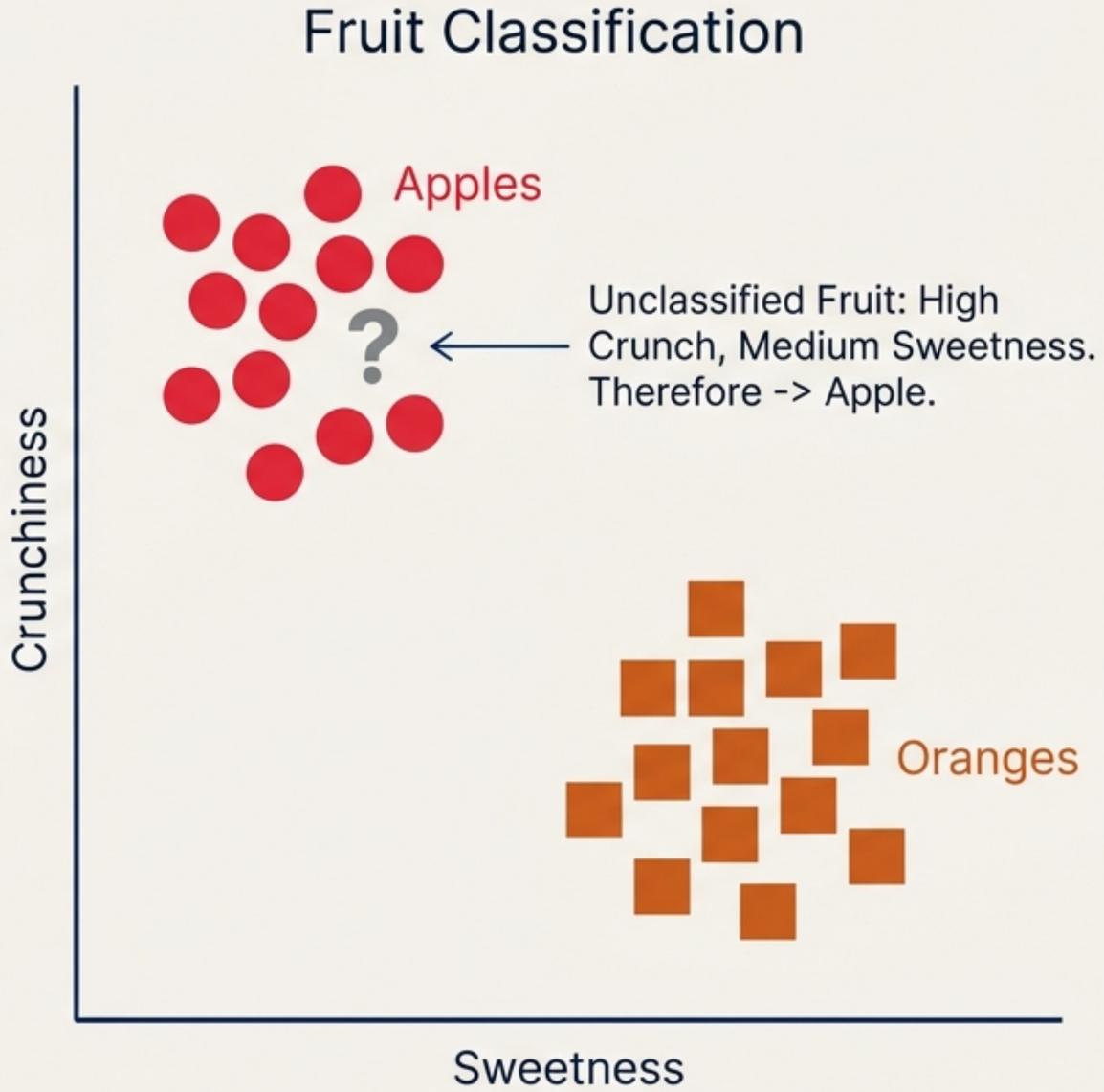


Prepared for Data Science Review

The Intuition: "Show me your neighbors, and I'll tell you who you are."

This is Classification by Proximity.

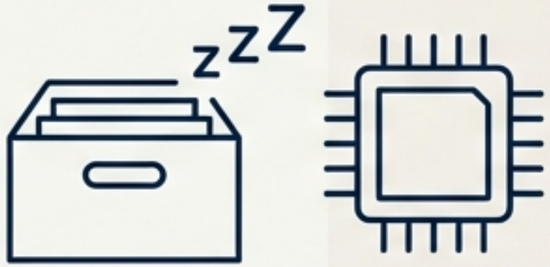
KNN operates on a simple assumption: similar things exist in **close proximity**. Just as you might judge a person by the company they keep, the algorithm judges a data point by the features of its **nearest neighbors**.



Three Pillars of the KNN Architecture

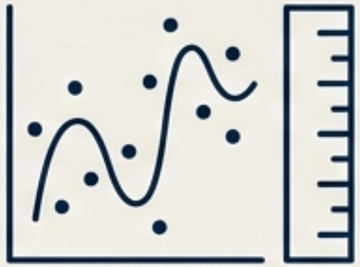
Supervised Learning

The algorithm requires a labeled dataset to function. It needs a ‘teacher’—we must already know which historical inputs correspond to which classes (e.g., ‘Cancer’ vs. ‘Benign’) to use them as a reference.



Lazy Learner

KNN does not build a model during the training phase. It simply memorizes the entire dataset. It defers all computation until the exact moment a prediction is requested, making training fast but prediction slower.



Non-Parametric

The algorithm makes no assumptions about the underlying data distribution. Unlike Linear Regression, which assumes a straight line, KNN adapts to the actual, often irregular shape of the data boundaries.

The Execution Roadmap: A 4-Step Process

Choose K

Select the number of neighbors to evaluate.

Calculate Distance

Measure distance between the new point and all training points.

Find Neighbors

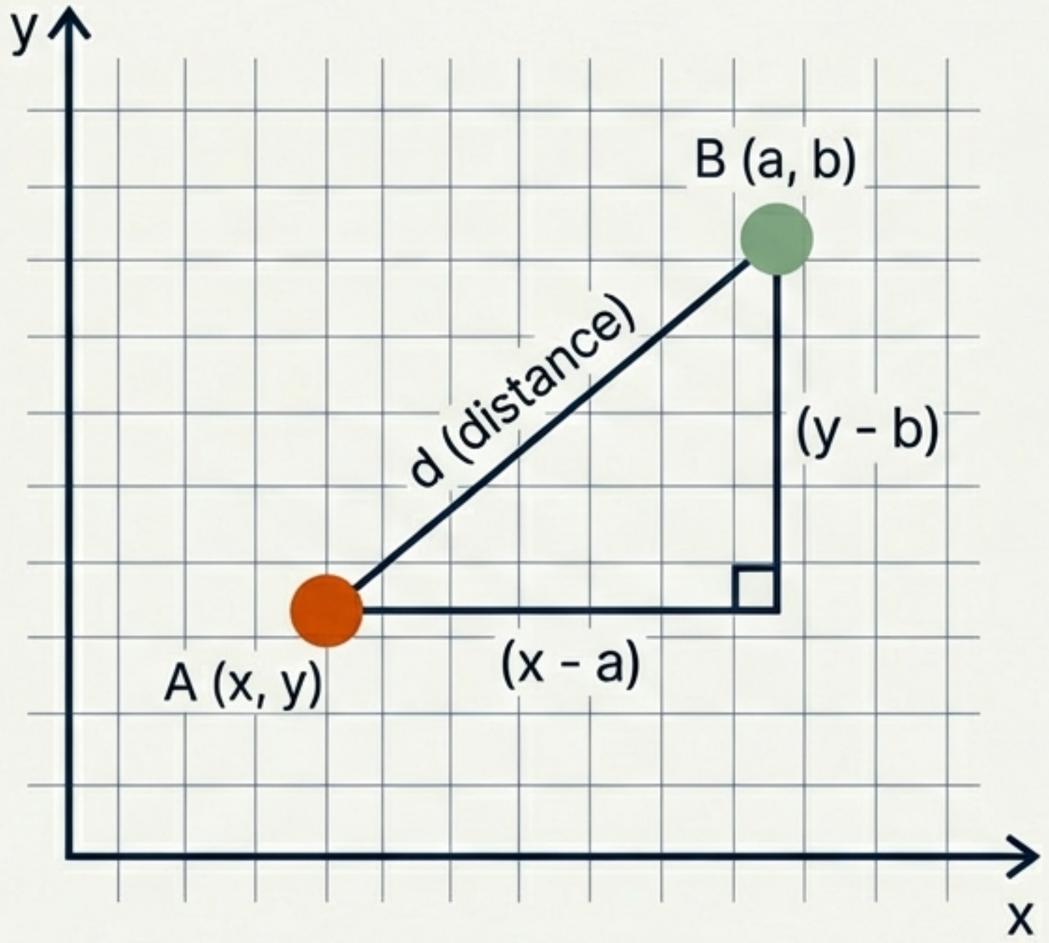
Identify the “K” points with the smallest distance values.

Vote & Classify

Assign the category that is most frequent among the neighbors.

This logic applies to both classification (majority vote) and regression (averaging values).

Defining ‘Near’: The Mathematics of Euclidean Distance



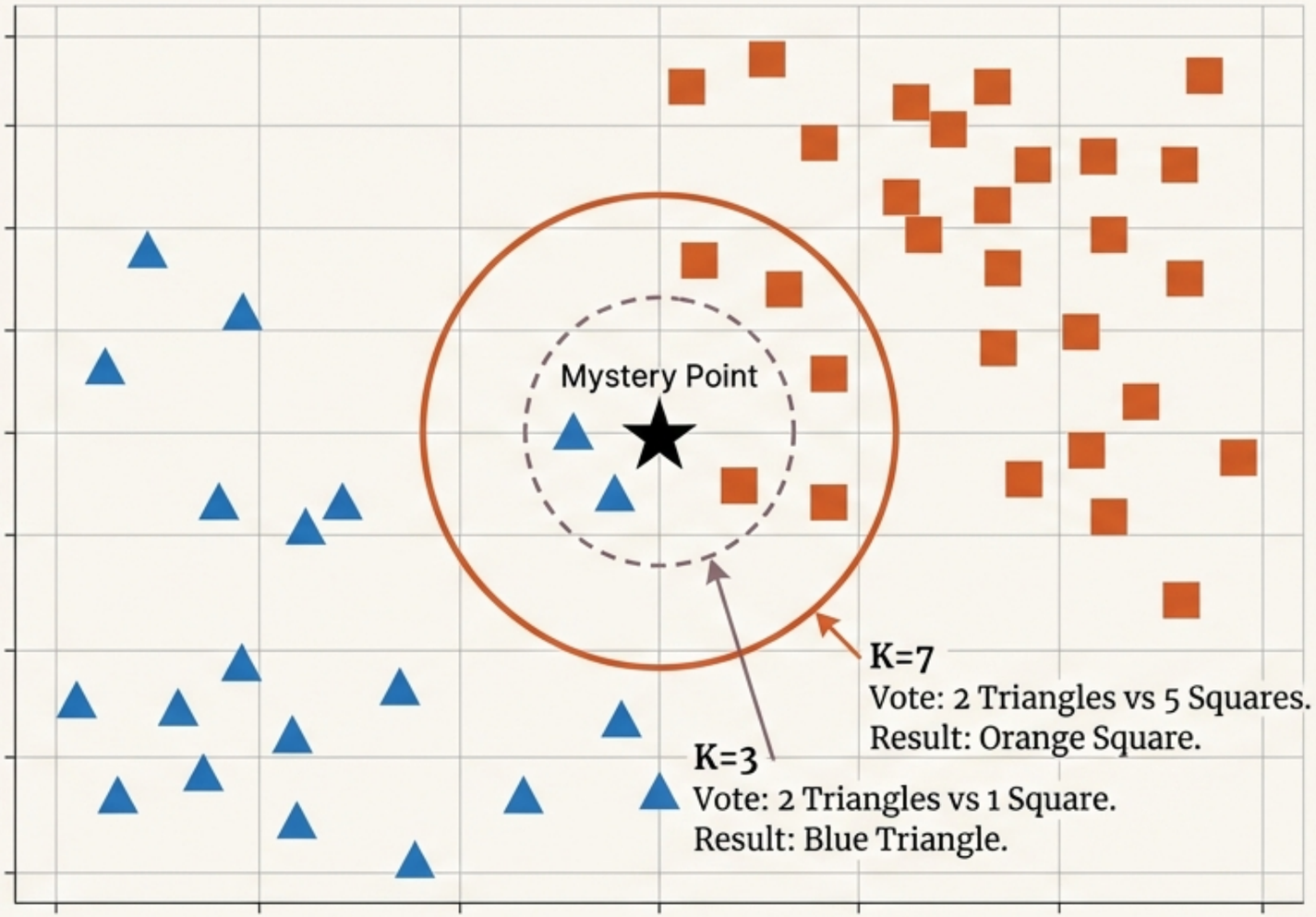
$$d = \sqrt{(x - a)^2 + (y - b)^2}$$

To find the nearest neighbor, the machine calculates the straight-line distance between points. This is an application of the Pythagorean theorem.

While other metrics exist (Manhattan, Mahalanobis), Euclidean distance is the industry standard for KNN.

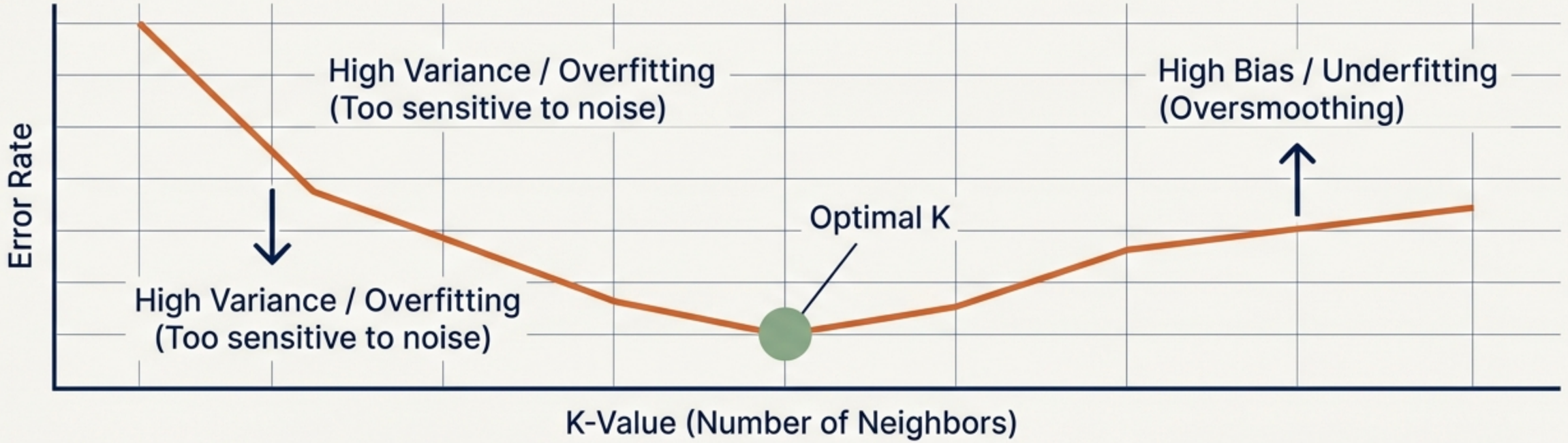
Context: For a dataset with patient height and weight, the algorithm computes this value for every single training record to find the smallest “d”.

The 'K' Factor: How Neighbor Count Dictates the Verdict



The definition of “truth” flips based on how wide we cast our net. Small K looks at local detail; Large K looks at broader averages.

Tuning the Algorithm: Finding the Optimal K



Rule of Thumb 1:

Start with $K = \sqrt{N}$ (Square root of total data points).

Rule of Thumb 2:

Always choose an ODD number for K. This prevents voting ties in binary classification (you can't have a 50/50 split with 5 voters).

Real-World Application: High-Stakes Medical Diagnosis

We will build a Python model to classify patients based on the Pima Indians Diabetes Dataset.

Input Features

- Glucose Level
- Blood Pressure
- Insulin
- BMI
- Age

The Goal

Healthy (Negative)

Diabetic (Positive)

Challenges: Medical data is often ‘noisy’ and contains missing values (e.g., placeholders where Insulin = 0). We must clean this before classification.

Python Implementation Step 1: Cleaning the Data

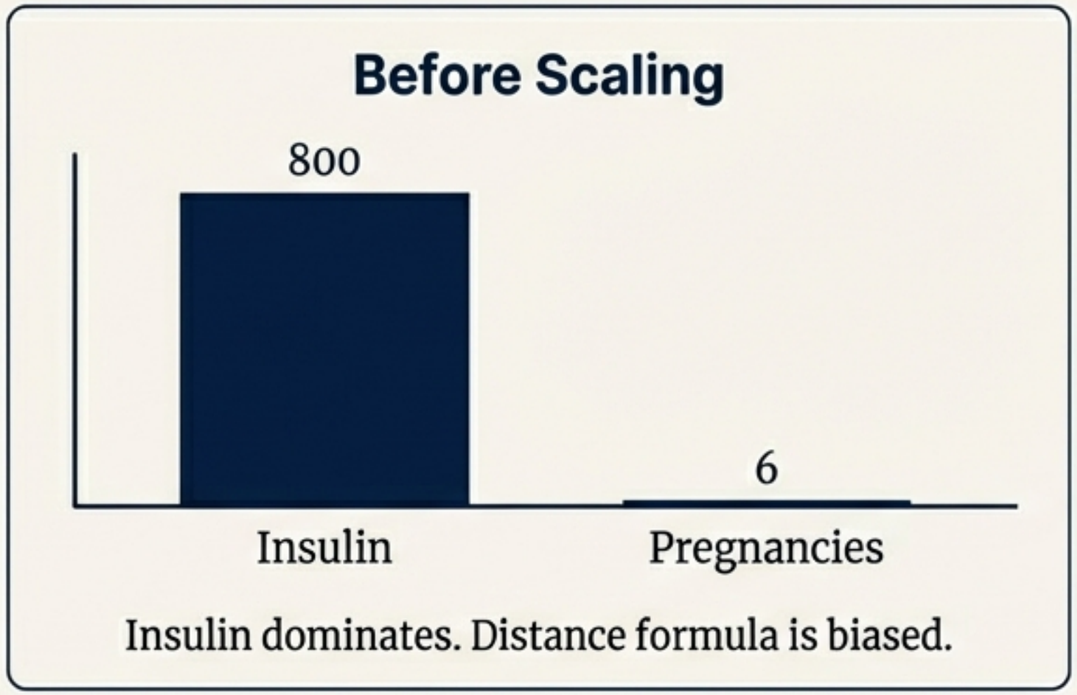
- Problem: The dataset contains impossible ‘0’ values for Glucose and Blood Pressure, indicating missing data.
- Solution: We impute these values. We cannot simply delete rows (data is precious), so we replace zeros with the column Mean.

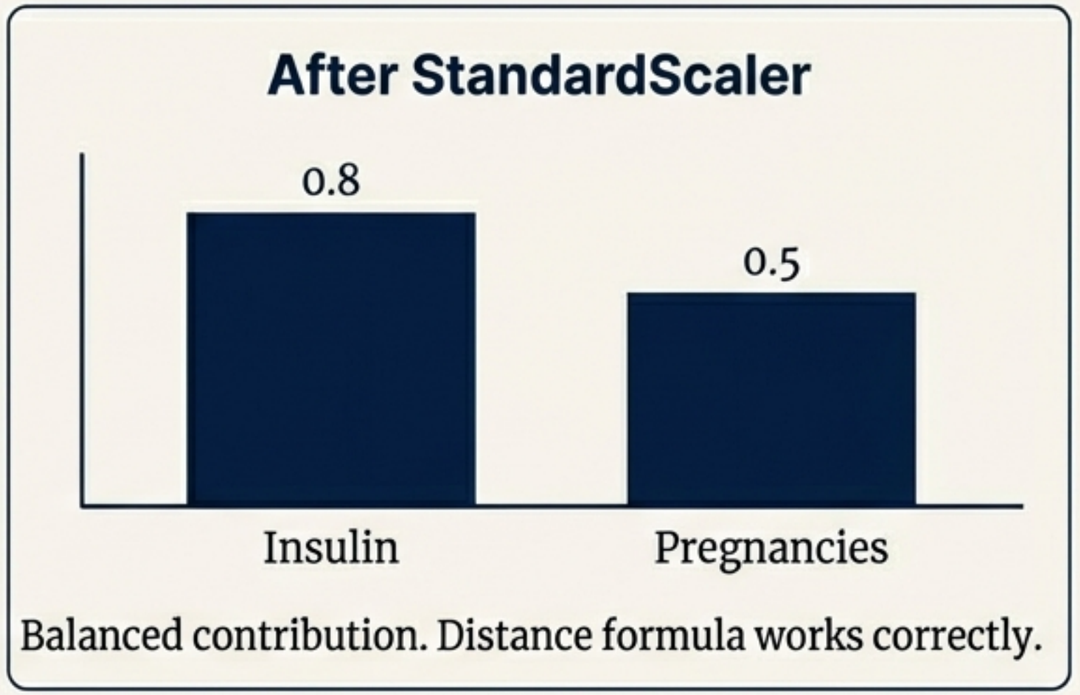
```
# Replace zeros with NaN (Not a Number)
dataset[columns_to_fix] =
dataset[columns_to_fix].replace(0,np.NaN)

# Fill NaN values with the mean of that
column
dataset[columns_to_fix].fillna(dataset[co
lumns_to_fix].mean(), inplace=True)

# Verify the fix
print(dataset.head())
```

Fills gaps with the average value to preserve data integrity.

Python Implementation Step 2: Feature Scaling

KNN relies on distance. If features have different scales, the larger numbers will dominate the calculation.



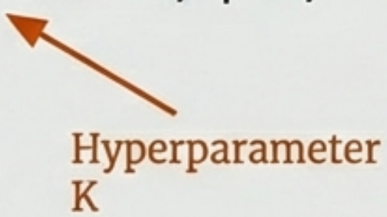



```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Python Implementation Step 3: Training & Prediction

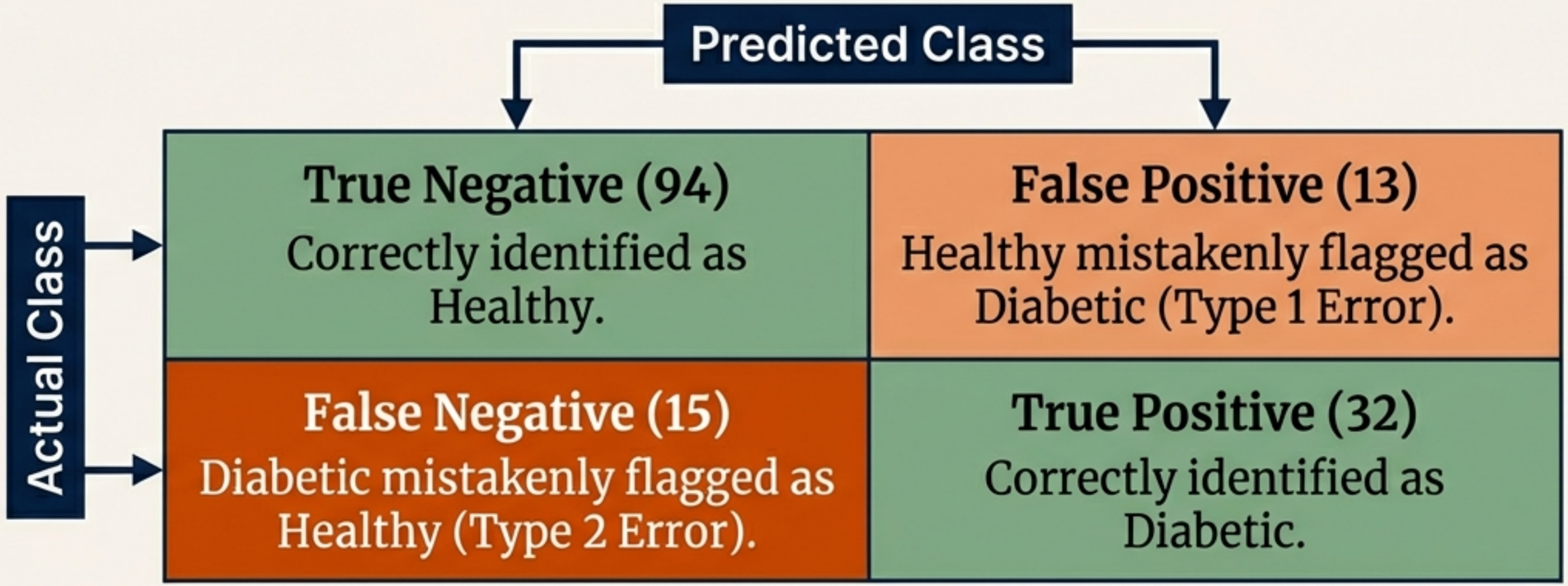
- We initialize the classifier with K=11.
- Why 11? N=153 (Test set). $\sqrt{153} \approx 12.3$. We choose 11 (odd number close to square root) to avoid ties.
- Note the 'Lazy' execution: .fit() only stores data; .predict() performs the calculation.

```
# Define the model: 11 Neighbors, Euclidean Distance
(p=2)
classifier = KNeighborsClassifier(n_neighbors=11, p=2,
      metric='euclidean')

# Fit the model (Store the training data)
classifier.fit(X_train, y_train)

# Predict the test set results
y_pred = classifier.predict(X_test)
```



Evaluation: The Confusion Matrix

Accuracy can be misleading. We need to see exactly *how* the model failed.




```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

Beyond Accuracy: Precision, Recall, & F1 Score

Precision

$$\frac{TP}{TP + FP}$$

Of all the patients we *labeled* diabetic, how many actually were?

0.71

Recall (Sensitivity)

$$\frac{TP}{TP + FN}$$

Of all the *actually* diabetic patients, how many did we find?

CRITICAL IN HEALTHCARE: A low recall means we missed sick patients (False Negatives).

0.68

F1 Score

$$2 * \frac{Precision * Recall}{Precision + Recall}$$

The harmonic mean. A balance between Precision and Recall.

0.69

In this use case, we might lower K or adjust threshold to improve Recall, even if Precision drops.

Summary: The Pros & Cons of KNN

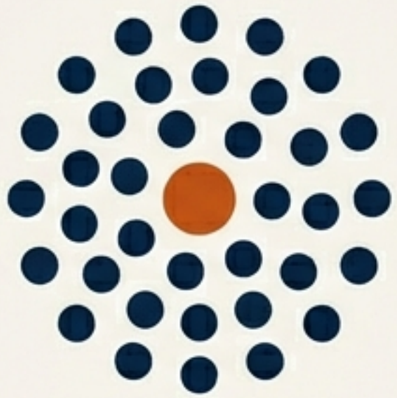
Advantages (When to use)

- Simple & Intuitive: Easy to explain and implement.
- No Training Time: Ideal for streaming data where models need to be updated instantly.
- Versatile: Naturally handles multi-class cases without modification.

Disadvantages (When to avoid)

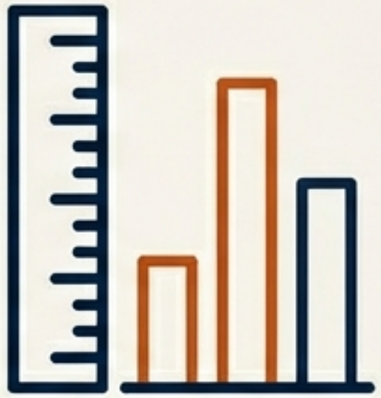
- Computationally Expensive: Slow with large datasets as it calculates distance for every point.
- Memory Intensive: Must store the entire training dataset.
- Curse of Dimensionality: Performance degrades drastically as the number of features increases.

Key Takeaways



Similarity is Identity

KNN classifies based on the features of nearest neighbors. It assumes similar data points cluster together.



Context Matters

Data must be scaled (StandardScaler) and ‘K’ must be tuned carefully (Rule of Thumb: $\sqrt{N}$) to balance noise and bias.

Verify Carefully

In critical fields like healthcare, Accuracy is not enough. Always check the Confusion Matrix and Recall to understand the cost of errors.

Presentation complete.