SUPERVISED MACHINE LEARNING: CLASSIFICATION

MODULE 2:

K NEAREST NEIGHBORS

TABLE OF CONTENTS

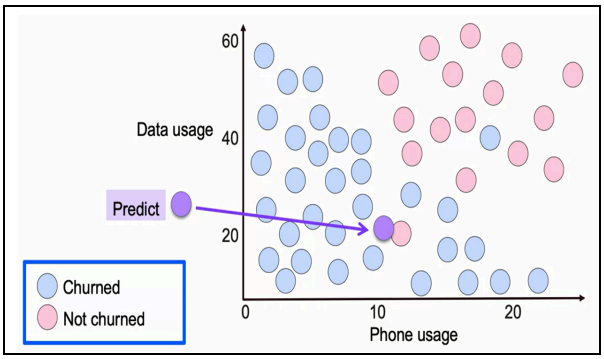
Introduction

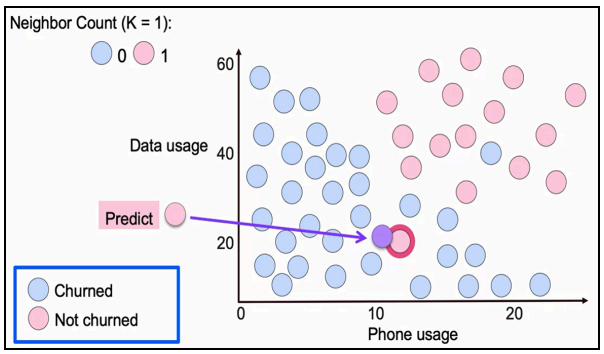
- **K-Nearest Neighbors (KNN)** is a **non-parametric** and **instance-based** supervised learning algorithm.
- It predicts the class of a new data point based on the *majority vote* of its **K nearest neighbors**.
- Works for both **classification** and **regression**, depending on whether the target variable is categorical or continuous.

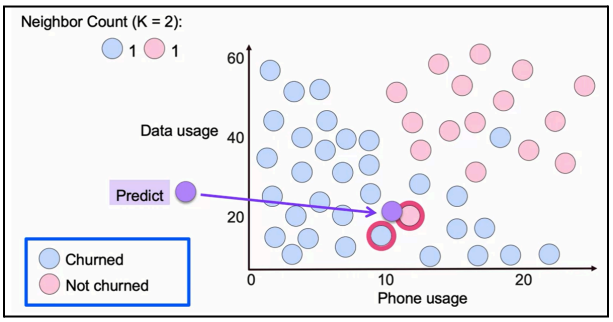
How KNN Works

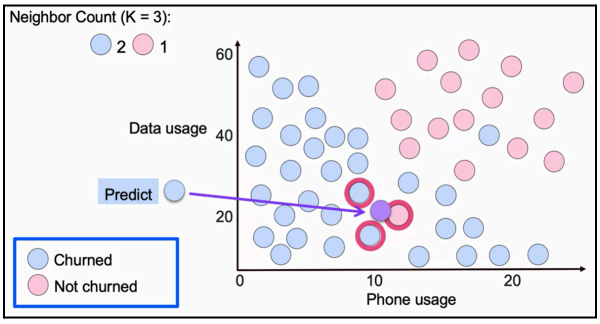
- Each data point is represented in a **feature space (không gian đặc trưng)**.
- When predicting a new sample:

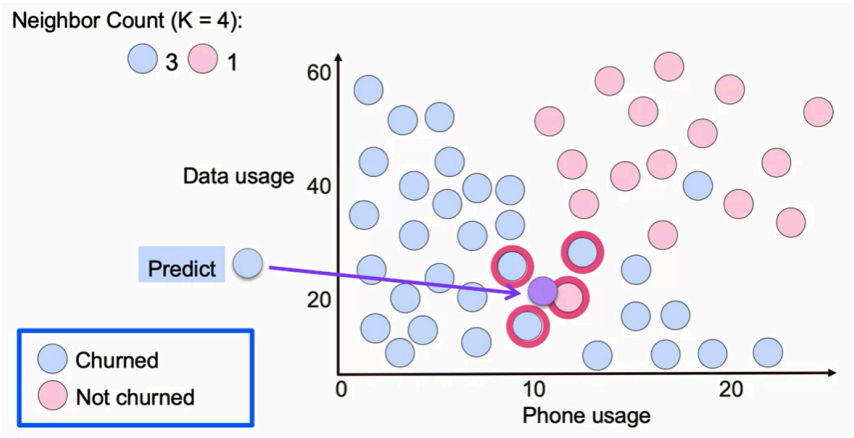
1. Step 1 − choose the value of K i.e. the nearest data points. K can be any integer.
2. Step 2 − For each point in the test data do the following:
    a. 2.1 − Calculate the distance between test data and each row of training data.
    b. 2.2 − Based on the distance value, sort them in ascending order.
    c. 2.3 − Choose the top K rows from the sorted array.
    d. 2.4 − Assign a class to the test point based on the most frequent class of these rows.
3. Step 3 − End









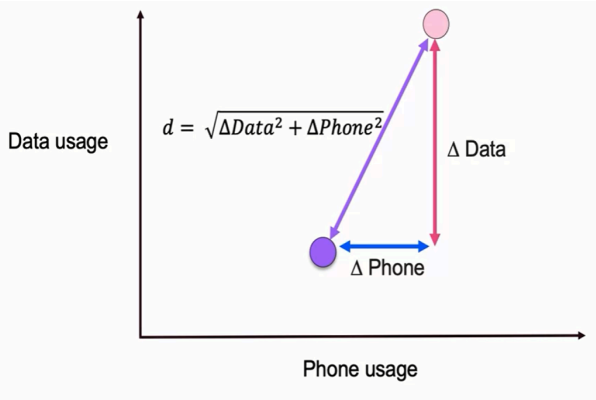


Choosing K (Hyperparameter)

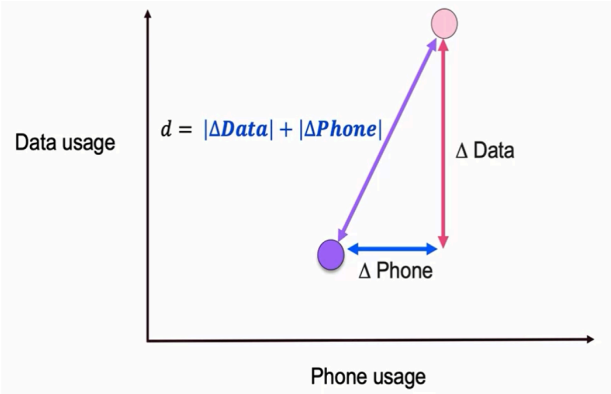
- **K** controls the bias–variance trade-off:
  - Small K → high variance (overfitting).
  - Large K → high bias (underfitting).
- Use **Elbow Method (phương pháp khuỷu tay)** to select optimal K:
  - Plot error rate vs. K and find where improvement stops increasing.

- Choice of **error metric** depends on business goal:
  - Prioritize **Recall** for catching positives.
  - Prioritize **Precision** for minimizing false alarms.
  - Or balance both with **F1 Score**.

Distance Metric



Euclidean distance



Manhattan distance

- **Euclidean distance (khoảng cách Euclid)** — default, geometric distance:

$$d(p, q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- **Manhattan distance (L1)** — absolute difference, "city block" path:

$$d(p, q) = \sum_i |p_i - q_i|$$

- Choice affects the model's shape of **decision boundary (ranh giới quyết định)**

Feature Scaling (Chuẩn hóa đặc trưng)

- KNN is **distance-based**, so features must be **on the same scale**.
- Two common scaling methods:
  - **Min–Max Scaling:**

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

  - **Standard Scaling:**

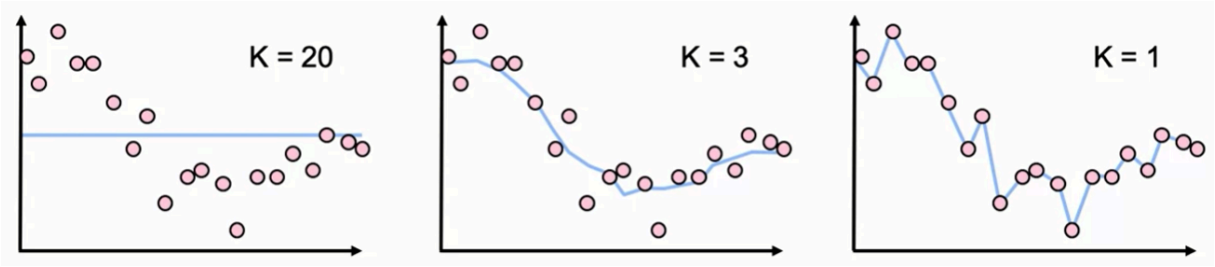
$$x' = \frac{x - \mu}{\sigma}$$

- Ensures no feature dominates others in distance calculation.

KNN for Multi-Class Classification

- Works naturally with more than two classes.
- The predicted class = **majority vote** among K neighbors.
- For multiple labels (e.g., 3 classes), K is often chosen as a **multiple of the number of classes + 1** to avoid ties.

KNN for Regression



- Instead of majority vote, the prediction is the **mean** of K nearest values:

$$\hat{y} = \frac{1}{K} \sum_{i=1}^{K} y_i$$

- Acts like a **smoothing function**, reducing noise with larger K.

Implementation in Python (Scikit-learn)

```
from sklearn.neighbors import
KNeighborsClassifier

# Initialize model
knn = KNeighborsClassifier(n_neighbors=3)

# Train model
knn.fit(X_train, y_train)

# Predict
y_pred = knn.predict(X_test)
```

- `n_neighbors`: number of K values.
- Default distance = Euclidean.
- Default weight = uniform (all neighbors equally weighted).

- For regression, use:

```
from sklearn.neighbors import
KNeighborsRegressor
```

Pros and Cons

- **Advantages**
  - Simple to implement and intuitive.
  - No need to estimate parameters.
  - Adapts quickly to new data (no retraining).
  - High **interpretability (dễ hiểu)** — easy to explain to business users.
- **Disadvantages**
  - **Slow prediction** — must compute distance to all training points.
  - **Memory intensive** — stores all data points.
  - **No explicit model** (unlike logistic regression).
  - **Sensitive to feature scaling** and **high-dimensional data** (curse of dimensionality).

Comparison with Linear/Logistic Regression

Aspect	Linear/Logistic Regression	KNN
Model Fitting	Slow (parameter estimation)	Fast (just store data)
Prediction	Fast	Slow (distance computation)
Memory	Efficient (Store coefficients)	High (store all data)
Interpretability	High (coefficients)	Moderate (Proximity logic)

Key Takeaways

- **KNN** predicts by voting among nearest data points in feature space.
- **K** is a **hyperparameter** controlling bias–variance balance.
- **Distance measure** and **feature scaling** crucially affect performance.
- **Elbow method** helps identify optimal K.
- Simple, interpretable, but computationally expensive for large datasets.
- Implemented easily with **KNeighborsClassifier** in scikit-learn.