Nearest Neighbor methods

KNN Regression

Definition

K-Nearest Neighbors (KNN) Regression is a **non-parametric supervised learning algorithm** that predicts the value of a target variable for a new data point by looking at the values of its **K nearest neighbors** in the training data. Instead of predicting a class (as in KNN classification), it predicts a **numerical value**.

How It Works

1. **Choose K**
   - Decide the number of neighbors K.
   - Example: K = 3.
2. **Distance Metric**
   - Compute the distance (commonly **Euclidean distance**) between the new data point and all training points.

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

3. **Find Nearest Neighbors**
   - Select the K training points that are closest to the new data point.
4. **Aggregate Neighbor Values**
   - For regression, take the **average (mean)** of the target values of the K neighbors:

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

- Sometimes, a **weighted average** is used, giving closer neighbors more influence:

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

Example

Suppose you want to predict a house price based on **square footage**.

- Training data:
  - 1000 sqft → $200k
  - 1500 sqft → $250k
  - 1700 sqft → $270k
  - 3000 sqft → $500k
- Query: 1600 sqft, K=2K=2.
  - Nearest neighbors: 1500 sqft ($250k), 1700 sqft ($270k).
  - Predicted price:

$$\hat{y} = \frac{250k + 270k}{2} = 260k$$

Advantages

- **Simple & intuitive** – easy to implement.
- **No training step** – just store data.
- **Flexible** – can model non-linear relationships.

Disadvantages

- **Computationally expensive** – needs distance calculations for all points at prediction time.
- **Sensitive to choice of K**.
- **Sensitive to irrelevant features and scale of data** → usually requires normalization/standardization.

Choosing K in KNN Regression

1. Small K (e.g., 1, 2, 3)

- **Pros**: Model is very flexible, captures local patterns well.
- **Cons**: Predictions become noisy (high variance), sensitive to outliers.
- Example: If K=1, you just copy the nearest neighbor's value (can be unstable).

---

2. Large K (e.g., 20, 50, 100)

- **Pros**: Predictions are smoother, less affected by noise.
- **Cons**: Model becomes too rigid (high bias), may miss important local variations.
- Example: If K is too large, you're basically averaging over the whole dataset → prediction tends toward the global mean.

---

3. Rule of Thumb

- Start with:

  $K \approx \sqrt{N}$

  where N = number of training samples.
  - Example: If you have 100 training points, start testing with $K \approx 10$.

---

4. Cross-Validation (Best Method ✅)

- Use **k-fold cross-validation**:
  1. Try different K values (e.g., 1 → 30).

2. Measure error (e.g., Mean Squared Error, RMSE).
3. Pick the K that minimizes error on validation sets.

---

5. Weighted KNN

- Instead of worrying too much about the exact K, you can assign **weights based on distance**:
  - Closer neighbors get a higher weight.
  - This reduces the problem of choosing a “perfect” K.

Visual Intuition

- **Small K** → "wiggly" prediction curve, overfitting.
- **Large K** → very smooth, underfitting.
- **Optimal K** → balance between bias and variance.

✅ In practice:

- Try several K values using **cross-validation**.
- Prefer **odd K** in classification (to avoid ties), but in regression, odd/even doesn’t matter.
- Normalize features first (since distance is sensitive to scale).

KNN Regression VS Linear Regression

🔷 1. Nature of the Model

- **KNN Regression**:
  - **Non-parametric** (doesn’t assume any specific function form).
  - Prediction is based on **local neighbors**.
- **Linear Regression**:
  - **Parametric** (assumes a linear relationship between features and target).

- Fits a global line/plane through all the data.

🔷 2. How Prediction Works

- **KNN Regression**:
  - For a new point, look at the **K nearest neighbors** and average their target values.
  - Local approach → depends on nearby data.
- **Linear Regression**:
  - Learns coefficients β for each feature using training data.
  - Prediction uses:

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n$$

🔷 3. Training vs Testing

- **KNN Regression**:
  - No real *training phase*. Just store data.
  - **Slow at prediction** (needs distance calculation each time).
- **Linear Regression**:
  - Needs a **training phase** (fit coefficients).
  - **Fast at prediction** (just plug into equation).

🔷 4. Interpretability

- **KNN Regression**: Harder to interpret, just averages neighbors.
- **Linear Regression**: Very interpretable (coefficients tell how much each feature affects target).

🔷 5. Handling Relationships

- **KNN Regression**:

- Can capture **non-linear relationships**.
- Works well if patterns are local.

- **Linear Regression**:
  - Only models **linear relationships** (unless you add polynomial features or interactions).

🔷 6. Sensitivity

- **KNN Regression**:
  - Sensitive to irrelevant features and scaling (always normalize data!).
  - Sensitive to choice of K.
- **Linear Regression**:
  - Sensitive to outliers (one extreme point can distort the line).
  - Assumes independence, homoscedasticity, and no multicollinearity.

🔷 7. Example: Predicting House Price by Square Footage

- Suppose houses are around a curve-shaped trend (non-linear).

👉 **KNN Regression**

- Looks at nearby houses, averages their prices → adapts to local curve.

👉 **Linear Regression**

- Fits a straight line. If the relationship is not linear, predictions will be biased.

🔷 Summary Table

Aspect	KNN Regression 🟢	Linear Regression 🔵
Model type	Non-parametric	Parametric
Training phase	None	Required (fit line)
Prediction speed	Slow	Fast
Relationship handled	Non-linear ok	Only linear

Aspect	KNN Regression 🟢	Linear Regression 🔵
Interpretability	Low	High
Sensitive to	Scale, K, neighbors	Outliers, assumptions

✅ In short:

- Use **Linear Regression** if the relationship is approximately linear and interpretability matters.
- Use **KNN Regression** if the relationship is unknown/non-linear and you have enough data.

🔷 Why KNN Works Well for Non-Linear Data

- **Linear Regression** assumes:

  $y = \beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n$

  → a straight line (or hyperplane). If the data has a **curved relationship**, it struggles unless we add polynomial terms or transformations.

- **KNN Regression** makes **no assumption** about the data's shape.
  - It predicts by looking at nearby points and averaging their values.
  - This means it naturally adapts to **curves, clusters, or any non-linear patterns**.

🔷 Example: Non-Linear Relationship

Suppose the true relationship is:

$y = \sin(x) + \text{noise}$

- **Linear Regression**: Will try to fit a straight line → poor fit.
- **KNN Regression**:
  - At each x, it looks around locally and averages nearby y values.
  - The prediction curve bends and follows the sine wave shape.

🔷 Visual Intuition

Imagine plotting the data points shaped like a **U-curve (quadratic)**:

- **Linear Regression**:
  - Fits a straight diagonal line through the points.
  - High bias, misses the "U" shape.
- **KNN Regression**:
  - Each prediction is based on local neighbors.
  - Predictions curve along the data naturally, following the "U".

🔷 Trade-off (Bias-Variance)

- **Small K** → very wiggly curve, may overfit noise.
- **Large K** → very smooth, may underfit (looks too linear).
- Cross-validation helps find the sweet spot.

✅ **Key takeaway**:

KNN is especially powerful for **non-linear patterns** because it doesn't assume any global form; it adapts locally to the data.

> ⚠️ KNN works better for lower numbers of features and more nonlinear data (for d ≤ 3)
> For linear relationships and higher-dimensional data, linear regression is generally more effective and provides a better generalized model (for d ≥ 4)

KNN Classification:K-Nearest Neighbors (KNN)

🔷 What is KNN?

K-Nearest Neighbors (KNN) is a **supervised machine learning algorithm** used for **classification** and **regression**.

- It works by **comparing new data to stored training data**.
- The prediction is made based on the **K closest (nearest) neighbors** in feature space.

It's called **lazy learning** because:

- No model is built during training.
- The algorithm just stores the dataset.
- Computation happens only when making a prediction.

🔷 How KNN Works (Step by Step)

1. **Choose K** → the number of neighbors to consider.
2. **Compute Distance** → between the query point and all training points.
   - Common metric: **Euclidean distance**

     $$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$
3. **Select Nearest Neighbors** → pick the K closest points.
4. **Make Prediction**:
   - **Classification** → majority vote of neighbor labels.
   - **Regression** → average (or weighted average) of neighbor target values.

🔷 KNN for Classification

- Output = **class label** (e.g., "cat" vs "dog").
- Prediction based on **majority voting** among neighbors.

Example: K=5, neighbors = {Dog, Dog, Cat, Dog, Cat} → Prediction = **Dog**.

🔷 KNN for Regression

- Output = **numerical value**.
- Prediction = **mean (or weighted mean)** of neighbor values.

Example: Predict house price → average prices of 5 nearest houses.

🔷 Choosing K

- **Small K (e.g., 1, 3)** → sensitive to noise (high variance).
- **Large K (e.g., 20, 50)** → smoother predictions but may underfit (high bias).
- Best K is chosen using **cross-validation**.
- Rule of thumb: $K \approx \sqrt{N}$, where N = number of samples.

🔷 Pros and Cons

✅ Advantages

- Simple and intuitive.
- Works for both classification and regression.
- Non-parametric → no assumption about data distribution.
- Can model **non-linear** decision boundaries.

❌ Disadvantages

- Computationally expensive at prediction time.
- Sensitive to feature scaling (need normalization).
- Struggles in high dimensions (curse of dimensionality).
- Sensitive to noisy/irrelevant features.

🔷 Example (Classification)

Dataset: Predict whether fruit is Apple 🍎 or Orange 🍊 based on **weight** and **color**.

- Training:
  - (150g, Red) → Apple

    - (180g, Red) → Apple
    - (120g, Orange) → Orange
    - (130g, Orange) → Orange
- Query: (160g, reddish).
- Distances → nearest neighbors = 2 Apples, 1 Orange → Prediction = **Apple**.

✅ **In short**:

KNN is a **memory-based algorithm** that predicts using **neighbors**:

- **Classification** = majority vote
- **Regression** = average value

Decision Boundary

Definition

A **decision boundary** is the line (in 2D), curve, or surface (in higher dimensions) that separates different **class regions** in the feature space, based on the predictions of a model.

- On one side of the boundary, → model predicts **Class A**
- On the other side → model predicts **Class B**

It's essentially the "border" where the model is undecided.

🔷 Example in 2D

Imagine a dataset with two features:

- x_1 = weight of fruit
- x_2 = color intensity

And two classes: **Apple (red)**, **Orange (orange)**.

- The **decision boundary** is the curve/line where the model is equally likely to predict Apple or Orange.

- If you plot the data points in 2D space, the boundary “cuts” the plane into regions → each region belongs to one class.

🔷 Decision Boundaries by Algorithm

1. **Linear Models (Logistic Regression, Linear SVM)**
   - Decision boundary is a **straight line (2D)** or **hyperplane (nD)**.
   - Equation looks like:

     $$w_1x_1 + w_2x_2 + b = 0$$
   - Example: Separate apples and oranges with a straight line.

---

1. **KNN (K-Nearest Neighbors)**
   - The decision boundary is **non-linear and jagged**.
   - It depends on the local arrangement of points.
   - For K=1, every training point creates its own “region” (Voronoi diagram).
   - For larger K, the boundaries are smoother.

---

1. **Decision Trees**
   - Boundaries are **axis-aligned** (vertical or horizontal splits in 2D).
   - Looks like a series of rectangles partitioning the feature space.

---

1. **Non-linear Models (Kernel SVM, Neural Nets, KNN)**
   - Decision boundaries can be **curved and complex**.
   - Adapt better to non-linear patterns in the data.

🔷 Why It Matters

- The decision boundary shows **how the model generalizes** to unseen data.
- **Too complex a boundary** = overfitting (follows noise).
- A **too simple boundary** = underfitting (misses real patterns).

🔷 Visualization Intuition

- Imagine a map where apples 🍎 and oranges 🍊 are scattered.
- The **decision boundary** is the "fence" the algorithm builds between the apple-region and the orange-region.
- If a new fruit falls inside the apple region, the model says **Apple**.

✅ **In short**:

The **decision boundary** is the dividing line/surface in feature space that tells us which class a new point belongs to. Its **shape** depends on the algorithm:

- Straight for linear models
- Jagged for KNN
- Step-like for trees
- Curvy for non-linear methods

Error Rate in KNN

Definition

The **error rate** in KNN is the fraction of predictions the algorithm gets **wrong** compared to the true labels.

Mathematically:

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

Equivalently:

Accuracy=1−Error Rate

🔷 How Error Rate is Measured

1. **Training Error (Resubstitution Error)**
    - Run KNN on the training dataset itself.

- With K=1, training error is **0** (every point is its own nearest neighbor).
- But this doesn't mean good generalization → likely overfitting.

2. **Test Error**
   - Run KNN on a separate test set.
   - Shows how well the model generalizes to unseen data.
3. **Cross-Validation Error**
   - Split data into k folds.
   - Train on k−1 folds, test on the remaining fold, repeat.
   - Average error across folds → more reliable estimate.

🔷 Error Rate vs K (Bias-Variance Tradeoff)

- **Small K (e.g., 1)**
  - Training error ≈ 0 (perfect memorization).
  - Test error is high → overfitting, high variance.
- **Large K (e.g., 50)**
  - Model becomes too smooth.
  - Training and test error both increase → underfitting, high bias.
- **Optimal K**
  - Somewhere in between → lowest test error.
  - Found by cross-validation.

📉 If you plot **Error Rate vs K**:

- Training error starts very low, increases with K.
- Test error decreases at first, then increases again (U-shape curve).

🔷 Example

Suppose we have 100 test samples, and KNN predicts 85 correctly.

- Correct predictions = 85
- Incorrect predictions = 15

$\text{Error Rate} = \frac{15}{100} = 0.15 = 15\%$

$Accuracy = 85$

✅ **In short**:

- **Error rate in KNN** = fraction of wrong predictions.
- Depends heavily on **choice of K** and **data distribution**.
- Best K is chosen by minimizing **cross-validation error**.

🔷 Variants of KNN

1. **Weighted KNN**

- Standard KNN: all neighbors contribute equally.
- Weighted KNN: closer neighbors have **higher influence**.
- Prediction:

$$\hat{y} = \frac{\sum_{i=1}^{K} w_i y_i}{\sum_{i=1}^{K} w_i}, \quad w_i = \frac{1}{d_i}$$

- Useful when local points are much more relevant than distant ones.

---

2. **Distance-Weighted KNN with Kernel Functions**

- Instead of just 1/d, use kernel functions to weight neighbors.
- Examples:
  - Gaussian kernel
  - Epanechnikov kernel
- Smooths predictions better.

---

3. **Condensed KNN (cKNN)**

- Stores only a **subset of representative points** (prototypes).
- Reduces memory usage and speeds up prediction.

---

4. **Reduced KNN (rKNN)**

- Removes training samples that don’t affect decision boundaries.
- Smaller, faster model without losing accuracy.

---

5. **Edited KNN (eKNN)**

- Cleans noisy data points by removing mislabeled/irregular samples.
- Improves robustness against outliers.

---

6. **Fuzzy KNN**

- Instead of a hard decision (class A or B), assigns **membership probabilities**.
- Example: “70% Cat, 30% Dog” instead of just “Cat”.

---

7. **KNN with Dimension Reduction**

- Apply **PCA, LDA, t-SNE, UMAP** before KNN to fight the **curse of dimensionality**.
- Improves performance in high-dimensional datasets (e.g., text, images).

---

8. **Approximate Nearest Neighbors (ANN)**

- Exact KNN is slow for large datasets.
- ANN methods (KD-Trees, Ball Trees, Locality Sensitive Hashing) speed up neighbor search.
- Popular in recommender systems and similarity search.

---

9. **Adaptive KNN**

- Instead of a fixed K, adaptively choose K based on local density.
- In dense areas → small K.
- In sparse areas → larger K.

---

10. **Parallel / Scalable KNN**

- Distributed KNN (e.g., using Hadoop/Spark).
- Designed for big data where classic KNN is too slow.

🔷 Summary

Variant	Purpose
Weighted KNN	Give closer neighbors more influence
Kernel KNN	Smooth weighting with kernels
Condensed/Reduced/Edited KNN	Reduce dataset size & noise
Fuzzy KNN	Soft probability-based classification
Dimensionality reduction + KNN	Handle high-dimensional data
Approximate KNN	Fast lookup in large datasets
Adaptive KNN	Flexible K depending on density
Parallel/Scalable KNN	Works with very large datasets

✅ **In short**: The basic KNN idea is simple, but variants exist to tackle **efficiency, noise, interpretability, high-dimensional data, and scalability**.
