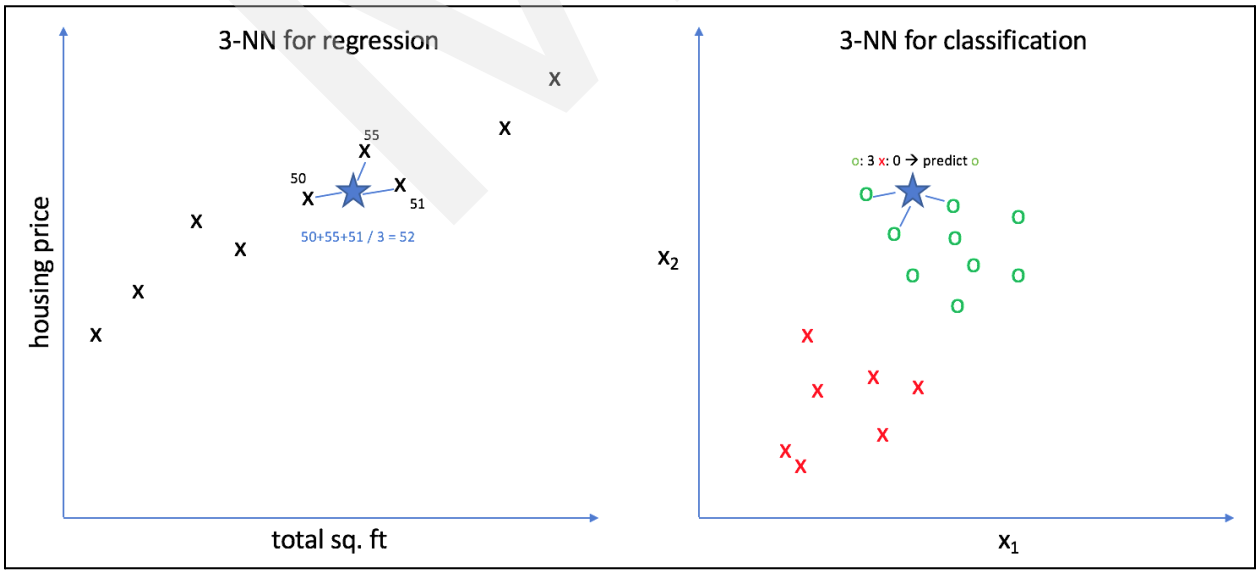
KNN (K-Nearest Neighbors)

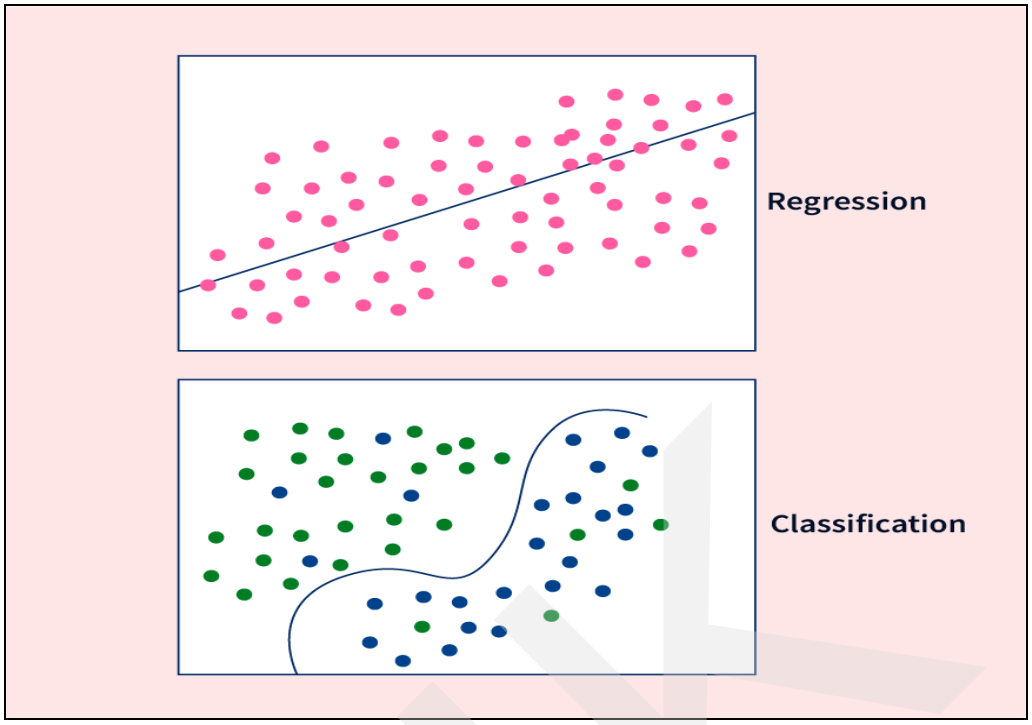
KNN (K-Nearest Neighbors) is a **supervised** learning algorithm. It uses labeled data to classify or predict new data points based on the proximity of similar instances in the training dataset. In essence, it "memorizes" the training data and then makes predictions based on the nearest neighbors to a new data point.

Labeled Data:
KNN relies on a training dataset where each data point is already labeled with a known class or value.

Learning a Function:
The algorithm "memorizes" the labeled training data and uses it to learn a function that maps input data points to corresponding output labels.

Classification and Regression:
KNN can be used for both classification (assigning a class label) and regression (predicting a continuous value) tasks, both of which fall under supervised learning.





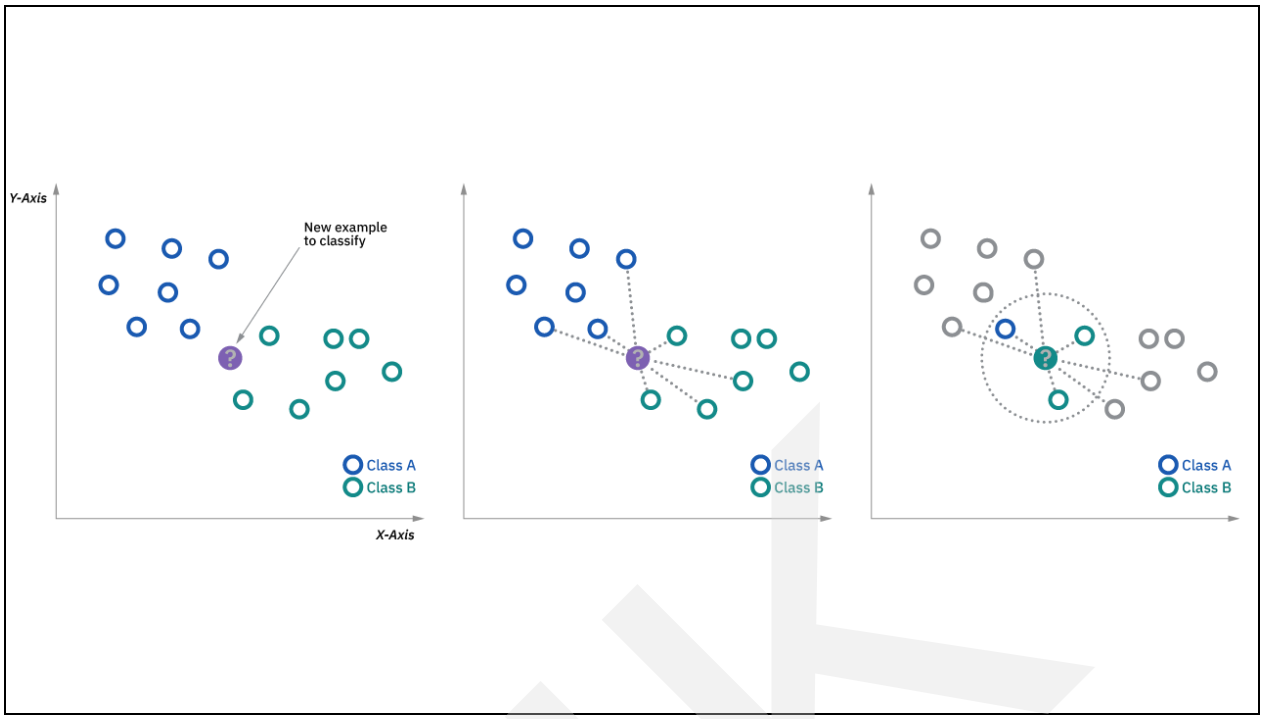
Lazy Learning:

KNN is also known as a **"lazy learner"** because it doesn't build a specific model during training. It simply stores the training data and makes predictions when a new instance is encountered.

K-Means Clustering (Unsupervised):

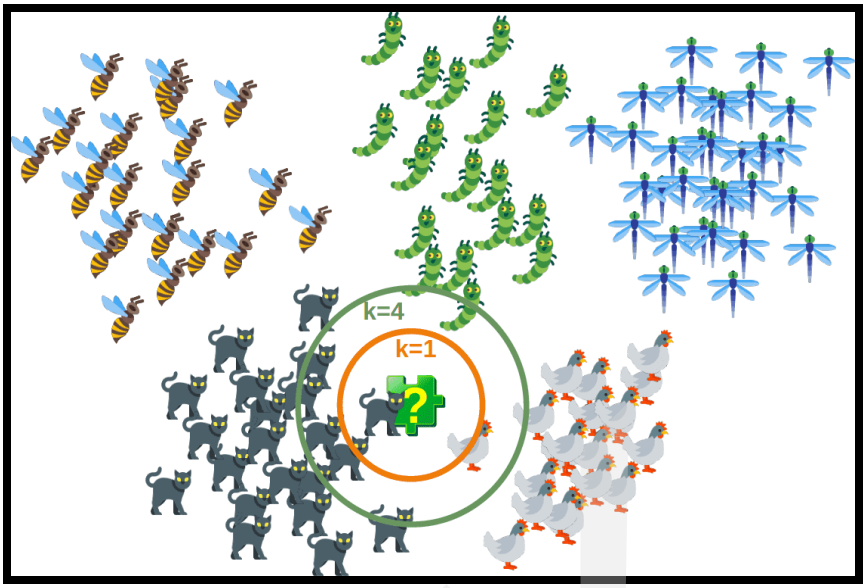
While KNN is often confused with K-means clustering, ***K-means is an unsupervised learning*** algorithm used for grouping **unlabeled data** points into clusters. KNN, on the other hand, uses labeled data to classify or predict.

Here's a to-the-point and organized summary of the lecture notes on **k-Nearest Neighbors (kNN)** from STAT 479 by Sebastian Raschka. I've explained complex terms in simple language and added creative, relatable examples. Key points are at the end.



📌 Introduction to kNN

- **kNN** is one of the simplest supervised machine learning algorithms.
- It works by storing training data and using it at prediction time—this is called **lazy learning** (unlike eager algorithms that learn a model ahead of time).
- **Nearest Neighbor (NN)** is just kNN with k=1.
- It predicts the output of a new point by looking at the "k" closest points and using their labels to decide.



This image illustrates how the **k-Nearest Neighbors (kNN)** algorithm classifies a new data point based on its closest neighbors. The unknown point (green puzzle) is surrounded by labeled data (animals). With **k = 1**, it is classified as a **cat** 🐈 because the nearest neighbor is a cat. With **k = 4**, it still becomes a cat since 3 out of the 4 closest neighbors are cats. This shows how kNN uses the majority label among the nearest k neighbors to make predictions, and how the choice of k affects the result.

🔍 Core Concepts and Categories

kNN is:

- **Lazy learning**: Doesn't learn until prediction.
- **Instance-based**: Compares the new data to stored examples.
- **Non-parametric**: Doesn't assume a model structure.
- **Discriminative**: Focuses on the boundary between classes, not on how the data is generated.

🎯 Common Use Cases

- **Computer vision & biometrics**: Face/ear recognition using geometric features.
- **Recommender systems**: Suggesting items using collaborative filtering.

- **Outlier detection**: Spotting data points that don't fit.

⚙️ How kNN Works (Algorithm Overview)

Training:

- Just store the data.

Prediction:

- Calculate the distance between the query point and all stored points.
- Select the k nearest ones.
- For classification: use majority/plurality vote.
- For regression: take the average (or median) of the neighbors.

🧱 Decision Boundary and Voronoi Diagrams

- The space is divided so that each region is "owned" by the closest training point.
- This creates **Voronoi cells**—regions that resemble cells in a honeycomb.
- For k>1, boundaries become smoother.

🧮 Classification vs. Regression

Classification:

- Use the most frequent class among the k neighbors.

Regression:

- Take the average of the values of the k neighbors.

📉 Curse of Dimensionality

- As the number of features (dimensions) increases, data becomes sparse.

- Neighbors become less meaningful because everything is far apart.

The Curse of Dimensionality refers to the challenges that arise when working with high-dimensional data:

1. **Sparsity** – Data becomes increasingly sparse as dimensions grow, making it harder to find meaningful patterns.

2. **Distance Metrics Breakdown** – Distances between points become less meaningful, as most pairs appear equally distant.

3. **Computational Complexity** – Processing and storage requirements grow exponentially with dimensions.

4. **Overfitting Risk** – Models become more prone to overfitting due to excessive features relative to samples.

5. **Visualization Difficulty** – Human interpretation becomes nearly impossible beyond 3D.

Solutions include dimensionality reduction (PCA, t-SNE), feature selection, and regularization techniques.

⏱️ **Time Complexity (Big-O Notation)**

- Brute-force kNN has **O(n × m)** complexity (n = number of examples, m = features).
- Can be slow with large datasets.

🚀 **Speeding Up kNN**

1. **Priority Queues (Heaps)**: Smart data structures that help find nearest neighbors faster.
2. **Data Structures**:
    - **KD-Trees**: Efficient in low dimensions.

    - **Ball-Trees**: Better for higher dimensions.

3. **Dimensionality Reduction**:

    - **Feature Selection**: Choose fewer relevant features.
      Feature Extraction: Use techniques like PCA.

4. **Simpler Distance Metrics**: Use Euclidean over complex ones like Mahalanobis.

5. **Pruning**:

    - Remove points that don't affect results.
    - Use prototypes to summarize dense regions.

6. **Parallelization**: Run distance calculations on multiple cores/machines.

📏 Distance Measures

- **Euclidean**: Straight-line distance (most common).
- **Manhattan**: Only moves in vertical and horizontal lines.
- **Minkowski**: General form of both above.
- **Hamming**: For binary features (count differences).
- **Cosine similarity**: For text-based vectors or word frequencies.

⚖️ Feature Scaling & Weighting

- Scaling ensures no feature dominates (e.g., age vs. income).
- Weighting gives more importance to certain features in distance calculations.

🎯 Distance-Weighted kNN

- Closer neighbors have more influence on the decision.

- Example: In classification, closer neighbors get higher "votes."

🔧 Improving Accuracy

- **Tune hyperparameters**:
    - Choose the best k (3–15 is typical; odd numbers for binary classification).
    - Scale features and pick the best distance measure.
- Use **cross-validation** to evaluate different settings.

✅ Advantages

- Simple to understand and implement.
- Works well with sufficient data and smart tuning.

❌ Disadvantages

- Slow with large data.
- Struggles with high-dimensional data.
- Sensitive to irrelevant or scaled features.

🧠 Advanced Perspectives

Error Bounds:

- 1-NN can perform nearly as well as the optimal classifier (Bayes optimal).

Bayesian View:

- kNN can be interpreted in probabilistic terms using Bayes' theorem.

🧩 Related Techniques

- **Locally Weighted Regression**: Fit regression locally around a query point.
- **Kernel Methods**: Use similarity functions to transform data into a higher dimension.

🐍 kNN in Python

- Scikit-learn provides built-in support.
- Choose method='auto' to let the library pick the best data structure (KD-Tree or Ball-Tree).

📌 Main Key Points (TL;DR)

- kNN is a **lazy**, **instance-based**, **non-parametric** algorithm.
- Uses **distance metrics** to find **nearest neighbors** for prediction.
- Great for **small datasets** or as a **baseline**.
- Struggles in **high dimensions**—use feature selection/reduction.
-
-
- Performance can be improved using **efficient data structures**, **scaling**, **parallelization**, and **distance weighting**.
- Works for both **classification and regression** tasks.
- Must tune **k**, **distance measure**, and **scaling** for best results.