Supervised Learning

- Given training data \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \)
- \( N \) input/output pairs; \( x_i \) - input, \( y_i \) - output/label
Supervised Learning

- Given training data \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \)
- \( N \) input/output pairs; \( x_i \) - input, \( y_i \) - output/label
- \( x_i \) is a vector consisting of \( D \) features
  - Also called attributes or dimensions
  - Features can be discrete or continuous
  - \( x_{im} \) denotes the \( m \)-th feature of \( x_i \)
Supervised Learning

- Given training data \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \)
- \( N \) input/output pairs; \( x_i \) - input, \( y_i \) - output/label
- \( x_i \) is a vector consisting of \( D \) features
  - Also called attributes or dimensions
  - Features can be discrete or continuous
  - \( x_{im} \) denotes the \( m \)-th feature of \( x_i \)

- Forms of the output:
  - \( y_i \in \{1, \ldots, C\} \) for classification; a discrete variable
  - \( y_i \in \mathbb{R} \) for regression; a continuous (real-valued) variable
Supervised Learning

- Given training data \( \{(x_1, y_1), \ldots, (x_N, y_N)\} \)

- \( N \) input/output pairs; \( x_i \) - input, \( y_i \) - output/label

- \( x_i \) is a vector consisting of \( D \) features
  
  - Also called attributes or dimensions
  
  - Features can be discrete or continuous
  
  - \( x_{im} \) denotes the \( m \)-th feature of \( x_i \)

- Forms of the output:
  
  - \( y_i \in \{1, \ldots, C\} \) for classification; a discrete variable
  
  - \( y_i \in \mathbb{R} \) for regression; a continuous (real-valued) variable

- **Goal:** predict the output \( y \) for an unseen test example \( x \)

- **This lecture:** Two intuitive methods
  
  - \( K \)-Nearest-Neighbors
  
  - Decision Trees
**K-Nearest Neighbor (K-NN)**

- Given training data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and a test point
- Prediction Rule: Look at the $K$ most similar training examples
Given training data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and a test point $x_i$,

Prediction Rule: Look at the $K$ most similar training examples

For classification: assign the majority class label (majority voting)
For regression: assign the average response
**$K$-Nearest Neighbor ($K$-NN)**

- Given training data $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ and a test point
- Prediction Rule: Look at the $K$ most similar training examples

- For classification: assign the majority class label (**majority voting**)
- For regression: assign the average response

**The algorithm requires:**
- Parameter $K$: number of nearest neighbors to look for
- **Distance function**: To compute the similarities between examples
**K-Nearest Neighbor (K-NN)**

- Given training data \( \mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\} \) and a test point
- Prediction Rule: Look at the \( K \) most similar training examples

![Diagram of K-NN with points and neighborhoods]

- For classification: assign the majority class label (majority voting)
- For regression: assign the average response

The algorithm requires:
- Parameter \( K \): number of nearest neighbors to look for
- **Distance function**: To compute the similarities between examples

**Special Case**: 1-Nearest Neighbor
K-Nearest Neighbors Algorithm

- Compute the test point’s distance from each training point
K-Nearest Neighbors Algorithm

- Compute the test point’s distance from each training point
- Sort the distances in ascending (or descending) order
K-Nearest Neighbors Algorithm

- Compute the test point’s distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the $K$ nearest neighbors
**K-Nearest Neighbors Algorithm**

- Compute the test point's distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the $K$ nearest neighbors
- Use **majority rule** (for classification) or **averaging** (for regression)
**K-Nearest Neighbors Algorithm**

- Compute the test point’s distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the $K$ nearest neighbors
- Use majority rule (for classification) or averaging (for regression)

**Note:** $K$-Nearest Neighbors is called a *non-parametric* method

- Unlike other supervised learning algorithms, $K$-Nearest Neighbors doesn’t learn an explicit mapping $f$ from the training data
**K-Nearest Neighbors Algorithm**

- Compute the test point’s distance from each training point
- Sort the distances in ascending (or descending) order
- Use the sorted distances to select the $K$ nearest neighbors
- Use majority rule (for classification) or averaging (for regression)

**Note:** $K$-Nearest Neighbors is called a *non-parametric* method

- Unlike other supervised learning algorithms, $K$-Nearest Neighbors doesn’t learn an explicit mapping $f$ from the training data
- It simply uses the training data at the test time to make predictions
The $K$-NN algorithm requires computing distances of the test example from each of the training examples.
The $K$-NN algorithm requires computing distances of the test example from each of the training examples.

Several ways to compute distances.

The choice depends on the type of the features in the data.
The $K$-NN algorithm requires computing distances of the test example from each of the training examples.

Several ways to compute distances.

The choice depends on the type of the features in the data.

Real-valued features ($\mathbf{x}_i \in \mathbb{R}^D$): Euclidean distance is commonly used.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{||\mathbf{x}_i||^2 + ||\mathbf{x}_j||^2 - 2\mathbf{x}_i^T \mathbf{x}_j}$$
**K-NN: Computing the distances**

- The *K*-NN algorithm requires computing distances of the test example from each of the training examples.
- Several ways to compute distances.
- The choice depends on the type of the features in the data.
- **Real-valued features** \((\mathbf{x}_i \in \mathbb{R}^D)\): Euclidean distance is commonly used

\[
d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{||\mathbf{x}_i||^2 + ||\mathbf{x}_j||^2 - 2\mathbf{x}_i^T \mathbf{x}_j}
\]

- Generalization of the distance between points in 2 dimensions.
The $K$-NN algorithm requires computing distances of the test example from each of the training examples.

Several ways to compute distances.

The choice depends on the type of the features in the data.

Real-valued features ($x_i \in \mathbb{R}^D$): Euclidean distance is commonly used.

$$d(x_i, x_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{||x_i||^2 + ||x_j||^2 - 2x_i^T x_j}$$

Generalization of the distance between points in 2 dimensions:

$$||x_i|| = \sqrt{\sum_{m=1}^{D} x_{im}^2}$$ is called the norm of $x_i$.

Norm of a vector $x$ is also its length.
The $K$-NN algorithm requires computing distances of the test example from each of the training examples.

Several ways to compute distances.

The choice depends on the type of the features in the data.

Real-valued features ($x_i \in \mathbb{R}^D$): Euclidean distance is commonly used.

\[
d(x_i, x_j) = \sqrt{\sum_{m=1}^{D} (x_{im} - x_{jm})^2} = \sqrt{\|x_i\|^2 + \|x_j\|^2 - 2x_i^T x_j}
\]

Generalization of the distance between points in 2 dimensions.

$\|x_i\| = \sqrt{\sum_{m=1}^{D} x_{im}^2}$ is called the norm of $x_i$.

Norm of a vector $x$ is also its length.

$x_i^T x_j = \sum_{m=1}^{D} x_{im}x_{jm}$ is called the dot (or inner) product of $x_i$ and $x_j$.

Dot product measures the similarity between two vectors (orthogonal vectors have dot product=0, parallel vectors have high dot product).
K-NN: Feature Normalization

- Note: Features should be on the same scale

- Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize
Note: **Features should be on the same scale**

Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize.

One way is:
- Replace $x_{im}$ by $z_{im} = \frac{(x_{im} - \bar{x}_m)}{\sigma_m}$ (make them **zero mean, unit variance**).
K-NN: Feature Normalization

- Note: Features should be on the same scale

- Example: if one feature has its values in millimeters and another has in centimeters, we would need to normalize

- One way is:
  - Replace \( x_{im} \) by \( z_{im} = \frac{(x_{im} - \bar{x}_m)}{\sigma_m} \) (make them zero mean, unit variance)
  - \( \bar{x}_m = \frac{1}{N} \sum_{i=1}^{N} x_{im} \): empirical mean of \( m^{th} \) feature
  - \( \sigma_m^2 = \frac{1}{N} \sum_{i=1}^{N} (x_{im} - \bar{x}_m)^2 \): empirical variance of \( m^{th} \) feature
**K-NN: Some other distance measures**

- **Binary-valued features**
  - Use Hamming distance: \( d(x_i, x_j) = \sum_{m=1}^{D} \mathbb{I}(x_{im} \neq x_{jm}) \)
  - Hamming distance counts the number of features where the two examples disagree

- **Mixed feature types** (some real-valued and some binary-valued)?
  - Can use mixed distance measures
  - E.g., Euclidean for the real part, Hamming for the binary part

- Can also assign **weights** to features: \( d(x_i, x_j) = \sum_{m=1}^{D} w_m d(x_{im}, x_{jm}) \)
Choice of $K$ - Neighborhood Size

- Small $K$
  - Creates many small regions for each class
  - May lead to non-smooth decision boundaries and overfit
Choice of $K$ - Neighborhood Size

- **Small $K$**
  - Creates many small regions for each class
  - May lead to non-smooth decision boundaries and overfit

- **Large $K$**
  - Creates fewer larger regions
  - Usually leads to smoother decision boundaries (caution: too smooth decision boundary can underfit)
Choice of $K$ - Neighborhood Size

- **Small $K$**
  - Creates many small regions for each class
  - May lead to non-smooth decision boundaries and overfit

- **Large $K$**
  - Creates fewer larger regions
  - Usually leads to smoother decision boundaries (caution: too smooth decision boundary can underfit)

- **Choosing $K$**
  - Often data dependent and heuristic based
  - Or using cross-validation (using some held-out data)
  - In general, a $K$ too small or too big is bad!
K-Nearest Neighbor: Properties

- **What's nice**
  - Simple and intuitive; easily implementable
$K$-Nearest Neighbor: Properties

- **What's nice**
  - Simple and intuitive; easily implementable
  - Asymptotically **consistent** (a theoretical property)
    - With infinite training data and large enough $K$, $K$-NN approaches the best possible classifier (**Bayes optimal**)
**K-Nearest Neighbor: Properties**

- **What’s nice**
  - Simple and intuitive; easily implementable
  - Asymptotically **consistent** (a theoretical property)
    - With infinite training data and large enough $K$, $K$-NN approaches the best possible classifier (**Bayes optimal**)

- **What’s not so nice..**
  - Store all the training data **in memory** even at test time
    - Can be memory intensive for large training datasets
    - An example of **non-parametric**, or **memory/instance-based** methods
    - Different from **parametric**, **model-based** learning models
$K$-Nearest Neighbor: Properties

- **What's nice**
  - Simple and intuitive; easily implementable
  - Asymptotically **consistent** (a theoretical property)
    - With infinite training data and large enough $K$, $K$-NN approaches the best possible classifier (**Bayes optimal**)
- **What’s not so nice.**
  - Store all the training data in memory even at test time
    - Can be memory intensive for large training datasets
    - An example of **non-parametric**, or **memory/instance-based** methods
    - Different from **parametric**, **model-based** learning models
  - Expensive at test time: $O(ND)$ computations for each test point
    - Have to **search through all training data** to find nearest neighbors
    - Distance computations with $N$ training points ($D$ features each)
K-Nearest Neighbor: Properties

What's nice

- Simple and intuitive; easily implementable
- Asymptotically consistent (a theoretical property)
  - With infinite training data and large enough $K$, $K$-NN approaches the best possible classifier (Bayes optimal)

What's not so nice..

- Store all the training data in memory even at test time
  - Can be memory intensive for large training datasets
  - An example of non-parametric, or memory/instance-based methods
  - Different from parametric, model-based learning models

- Expensive at test time: $O(ND)$ computations for each test point
  - Have to search through all training data to find nearest neighbors
  - Distance computations with $N$ training points ($D$ features each)

- Sensitive to noisy features
What’s nice

- Simple and intuitive; easily implementable
- Asymptotically consistent (a theoretical property)
  - With infinite training data and large enough $K$, $K$-NN approaches the best possible classifier (Bayes optimal)

What’s not so nice...

- Store all the training data in memory even at test time
  - Can be memory intensive for large training datasets
  - An example of non-parametric, or memory-instance-based methods
  - Different from parametric, model-based learning models

- Expensive at test time: $O(ND)$ computations for each test point
  - Have to search through all training data to find nearest neighbors
  - Distance computations with $N$ training points ($D$ features each)

- Sensitive to noisy features
- May perform badly in high dimensions (curse of dimensionality)
  - In high dimensions, distance notions can be counter-intuitive!
Computational speed-ups (don’t want to spend $O(ND)$ time)
  - Improved data structures for fast nearest neighbor search
  - Even if *approximately* nearest neighbors, yet may be good enough

Efficient Storage (don’t want to store all the training data)
  - E.g., subsampling the training data to retain “prototypes”
  - Leads to computational speed-ups too!

Metric Learning: Learning the “right” distance metric for a given dataset