Recall: Supervised Learning

Formal definition:

**Given** training data:

\[ X = \{x_1, \ldots, x_n\} \text{ inputs} \]
\[ Y = \{y_1, \ldots, y_n\} \text{ labels - if discrete: classification} \]

**Produce:**

Decision function \( f : X \rightarrow Y \)

That minimizes error:

\[ \sum_i err(f(x_i), y_i) \]
Decision Trees

The Perceptron

If your input \( x \) is real-valued ... explicit decision boundary?

The Perceptron

Which side of a line are you on?

\[ w \cdot x = ||w|| ||x|| \cos(\theta) \]
The Perceptron

How do you reduce error?

\[ e = (y_i - (w \cdot x_i + c))^2 \]

\[ \frac{\partial e}{\partial w_j} = -2(y_i - (w \cdot x_i + c))x_i(j) \]

descend this gradient to reduce error

The Perceptron Algorithm

Assume you have a batch of data:
\[ X = \{x_1, \ldots, x_n\} \]
\[ Y = \{y_1, \ldots, y_n\} \]

set w, c to 0.
for each \( x_i \):
\[ \text{predict } z_i = \text{sign}(w \cdot x_i + c) \]
if \( z_i \neq y_i \):
\[ w = w + a(y_i - z_i)x_i \]
converges if data is linearly separate

Probabilities

What if you want a probabilistic classifier?

Instead of \( \text{sign} \), squash output of linear sum down to \([0, 1]\):
\[ \sigma(w \cdot x + c) \]

Resulting algorithm: logistic regression.

Perceptrons

What can’t you do?
Frank Rosenblatt

Built the *Mark I* in 1960.

**Neural Networks**

\[ \sigma(w \cdot x + c) \]

logistic regression

**Nonparametric Methods**

Most ML methods can be characterized by setting a few parameters.

Alternative approach:
- Let the data speak for itself.
- No finite-sized parameter vector.
- Usually more interesting decision boundaries.

**K-Nearest Neighbors**

*Given* training data:

\[ X = \{x_1, \ldots, x_n\} \]

\[ Y = \{y_1, \ldots, y_n\} \]

Distance metric \( D(x_i, x_j) \)

For a new data point \( x_{\text{new}} \):

find \( k \) nearest points in \( X \) (measured via \( D \))

set \( y_{\text{new}} \) is the majority label
K-Nearest Neighbors

Properties:
• No learning phase.
• Must store all the data.
• $\log(n)$ computation per sample - grows with data.

Decision boundary: any function, given enough data.

Classic trade-off: memory and compute time for flexibility.