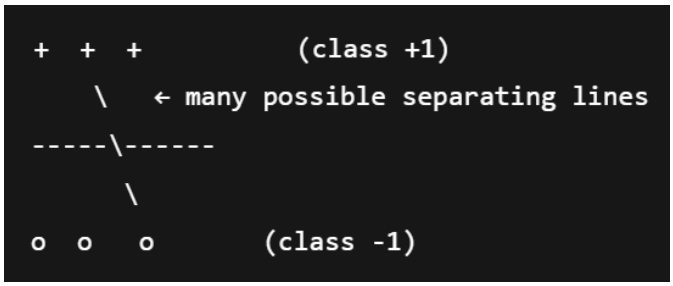
Big-picture

We saw a few decision boundaries:

- KNN → very wiggly boundary that follows the data closely
- Decision Trees → boundaries are axis-aligned splits (vertical/horizontal lines)
- Naive Bayes → probabilistic boundary based on feature distributions
- SVM → “find the widest street that separates the classes”

Imagine two classes of points in 2D that are linearly separable. There are many possible separating lines, but SVM picks the one with the largest margin — i.e., the biggest gap between the line and the closest points from each class.

Those closest points are called support vectors.



SVM says: “Among all valid separating lines, choose the one with the biggest buffer zone between + and -.”

The Decision Boundary: What equation does a separating line have?

SVM uses a linear boundary.

In 2-D, any straight line can be written as: $w1x1 + w2x2 + b = 0$ (or $w^Tx + b = 0$)

The “Street” (Margins)

Once you pick a separating line, SVM constructs two parallel lines:

- $w^Tx + b = +1$
- $w^Tx + b = -1$

Those two lines are the edges of the street.

The middle line $w^Tx + b = 0$ is the decision boundary.

Points must lie on the correct side:

- Positive class points must satisfy $w^Tx + b >= +1$
- Negative class points must satisfy $w^Tx + b <= -1$

The distance between the two margin lines turns out to be: $width = (2 / ||w||)$

To make the street as wide as possible, we want $||w||$ as small as possible

Why only the closest points matter

The closest points (touching the margin lines) determine the width.

- These are the support vectors.

- Everything else doesn't affect the boundary.
- KNN uses all points equally → sensitive to noise
- SVM uses only the hardest points to separate → more robust

Perceptron Algorithm

Goal: want to find a line $w^T x + b = 0$ that correctly separates +1 and -1 points. This algorithm does that by iteratively nudging the line whenever it makes a mistake.

Key intuition: if a point (xi, yi) is misclassified ……

- Its label says it should be on one side…
- But the line says it's on the wrong side.
- So we *move the line toward that point* so it becomes correctly classified.

The Update:

- $W_{new} = w_{old} + yi \cdot a \cdot xi$
- $New = b_{old} + yi \cdot a$

Where yi = +1 or -1…… xi = the points feature vector……. a = small learning rate

Why does this work?

- If the point is from the + class, you nudge the boundary toward + side
- If it's from the – class, you nudge it the other way
- This gradually rotates/shifts the line until it correctly separates everything (if possible).

The Full Perceptron Algorithm:

1. Start with a random line
   - Pick random:
     - weight vector w
     - bias b

2. Set hyperparameters
   - Number of passes/iterations (e.g., 100)
   - Learning rate a
   - Expansion/retraction rate c (used in this SVM-style version)

3. Repeat for many iterations
   01. For each iteration:
       a. Pick a training point (xi, yi)
       b. Check if it's correctly classified
   02. If correct → **do nothing**
   03. If incorrect → **update**
   04. Multiply the line by c to "expand/retract" the street (this controls the margin width)
   05. After many iterations …… the line will have been shifted and rotated toward a boundary that separates the classes.

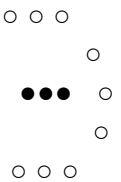
Width of a Street

- Width = 2 / ||w||
- Size of w is inversely proportional to the width of the street.
- Find widest street: max(2 / ||w||) = min(||w||) = min(½ $||w||^2$)

Kernel Function (intuition)

Sometimes no straight line can separate the data in the original feature space.

Ex:



- The inner product of a space describes how close / similar points are
- Kernel Functions allow for specifying the closeness / similarity of points in a hypothetical transformed space
- The hope is that with that new notion of closeness, points in the dataset are linearly separable.