Linear classifiers: Support vector machines and perceptrons
Support vector machines

- When the data is linearly separable, there may be more than one separator (hyperplane).

Which separator is best?
Support vector machines

- Find hyperplane that maximizes the *margin* between the positive and negative examples

\[
\begin{align*}
\text{x positive (} y = 1 \text{): } & \quad x \cdot w + b \geq 1 \\
\text{x negative (} y = -1 \text{): } & \quad x \cdot w + b \leq -1
\end{align*}
\]

For support vectors, \( x \cdot w + b = \pm 1 \)

Distance between point and hyperplane:

\[
\frac{|x \cdot w + b|}{\|w\|}
\]

Therefore, the margin is \( \frac{2}{\|w\|} \)

Finding the maximum margin hyperplane

1. Maximize margin $2 / \|w\|$

2. Correctly classify all training data:

   - $x_i$ positive ($y_i = 1$): $x_i \cdot w + b \geq 1$
   - $x_i$ negative ($y_i = -1$): $x_i \cdot w + b \leq -1$

Quadratic optimization problem:

$$\min_{w,b} \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i (w \cdot x_i + b) \geq 1$$

SVM training in general

• Separable data:

\[
\min_{w, b} \frac{1}{2} \|w\|^2 \quad \text{subject to} \quad y_i(w \cdot x_i + b) \geq 1
\]

Maximize margin
Classify training data correctly

• Non-separable data:

\[
\min_{w, b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i(w \cdot x_i + b))
\]

Maximize margin
Minimize classification mistakes
SVM training in general

\[
\min_{w, b} \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \max(0, 1 - y_i (w \cdot x_i + b))
\]
Nonlinear SVMs

- General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable.

\[ \Phi: x \rightarrow \phi(x) \]

Image source
Nonlinear SVMs

- Linearly separable dataset in 1D:

  ![Linearly separable dataset in 1D](image)

- Non-separable dataset in 1D:

  ![Non-separable dataset in 1D](image)

- We can map the data to a *higher-dimensional space*:

  ![Higher-dimensional space](image)

Slide credit: Andrew Moore
The kernel trick

• General idea: the original input space can always be mapped to some higher-dimensional feature space where the training set is separable

• **The kernel trick:** instead of explicitly computing the lifting transformation \( \varphi(x) \), define a kernel function \( K \) such that

\[
K(x, y) = \varphi(x) \cdot \varphi(y)
\]

(to be valid, the kernel function must satisfy *Mercer’s condition*)
The kernel trick

- Linear SVM decision function:

\[ w \cdot x + b = \sum_{i} \alpha_i y_i x_i \cdot x + b \]

The kernel trick

• Linear SVM decision function:

\[ w \cdot x + b = \sum_i \alpha_i y_i x_i \cdot x + b \]

• Kernel SVM decision function:

\[ \sum_i \alpha_i y_i \varphi(x_i) \cdot \varphi(x) + b = \sum_i \alpha_i y_i K(x_i, x) + b \]

• This gives a nonlinear decision boundary in the original feature space

Polynomial kernel: $K(x, y) = (c + x \cdot y)^d$
Gaussian kernel

- Also known as the radial basis function (RBF) kernel:

\[
K(x, y) = \exp\left(- \frac{1}{\sigma^2} \|x - y\|^2\right)
\]
Gaussian kernel

Demo: http://cs.stanford.edu/people/karpathy/svmjs/demo
SVMs: Pros and cons

**Pros**

- Kernel-based framework is very powerful, flexible
- Training is convex optimization, globally optimal solution can be found
- Amenable to theoretical analysis
- SVMs work very well in practice, even with very small training sample sizes

**Cons**

- No “direct” multi-class SVM, must combine two-class SVMs (e.g., with one-vs-others)
- Computation, memory (esp. for nonlinear SVMs)
Perceptron

Input

Weights

\( x_1 \)

\( w_1 \)

\( x_2 \)

\( w_2 \)

\( x_3 \)

\( w_3 \)

\( \cdot \)

\( \cdot \)

\( \cdot \)

\( x_D \)

\( w_D \)

Output: \( \text{sgn}(\mathbf{w} \cdot \mathbf{x} + b) \)

Can incorporate bias as component of the weight vector by always including a feature with value set to 1.
Loose inspiration: Human neurons
NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) — The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for news media.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learn by Doing

In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.
Linear separability

$x_1 \text{ and } x_2$

$x_1 \text{ or } x_2$

$x_1 \text{ xor } x_2$
Perceptron training algorithm

Initialize weights

Cycle through training examples in multiple passes (epochs)

For each training example:
  • If classified correctly, do nothing
  • If classified incorrectly, update weights
Perceptron update rule

For each training instance $\mathbf{x}$ with label $y$:

- Classify with current weights: $y' = \text{sgn}(\mathbf{w} \cdot \mathbf{x})$
- Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha (y - y') \mathbf{x}$
- $\alpha$ is a learning rate that should decay as a function of epoch $t$, e.g., $1000/(1000+t)$
- What happens if $y'$ is correct?
- Otherwise, consider what happens to individual weights $w_i \leftarrow w_i + \alpha (y - y') x_i$
  - If $y = 1$ and $y' = -1$, $w_i$ will be increased if $x_i$ is positive or decreased if $x_i$ is negative $\Rightarrow \mathbf{w} \cdot \mathbf{x}$ will get bigger
  - If $y = -1$ and $y' = 1$, $w_i$ will be decreased if $x_i$ is positive or increased if $x_i$ is negative $\Rightarrow \mathbf{w} \cdot \mathbf{x}$ will get smaller
Convergence of perceptron update rule

**Linearly separable data:** converges to a perfect solution

**Non-separable data:** converges to a minimum-error solution assuming learning rate decays as $O(1/t)$ and examples are presented in random sequence
Implementation details

Bias (add feature dimension with value fixed to 1) vs. no bias
Initialization of weights: all zeros vs. random
Learning rate decay function
Number of epochs (passes through the training data)
Order of cycling through training examples (random)
Multi-class perceptrons

One-vs-others framework: Need to keep a weight vector $\mathbf{w}_c$ for each class $c$

Decision rule: $c = \operatorname{argmax}_c \mathbf{w}_c \cdot \mathbf{x}$
Multi-class perceptrons

One-vs-others framework: Need to keep a weight vector $w_c$ for each class $c$

Decision rule: $c = \text{argmax}_c \ w_c \cdot x$

Update rule: suppose example from class $c$ gets misclassified as $c'$

- Update for $c$: $w_c \leftarrow w_c + \alpha x$
- Update for $c'$: $w_{c'} \leftarrow w_{c'} - \alpha x$
Visualizing the weights

How can we make a nonlinear classifier out of a perceptron?

Multi-layer neural networks (in the next lecture…)

input layer

hidden layer 1  hidden layer 2

output layer
Neural networks vs. SVMs
(a.k.a. “deep” vs. “shallow” learning)