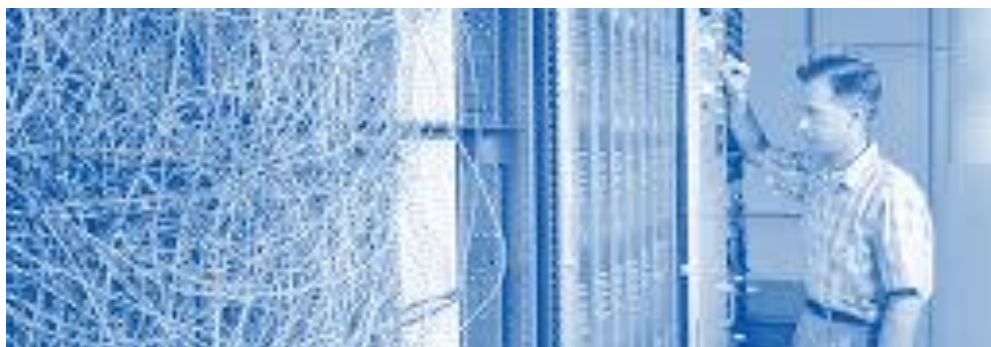
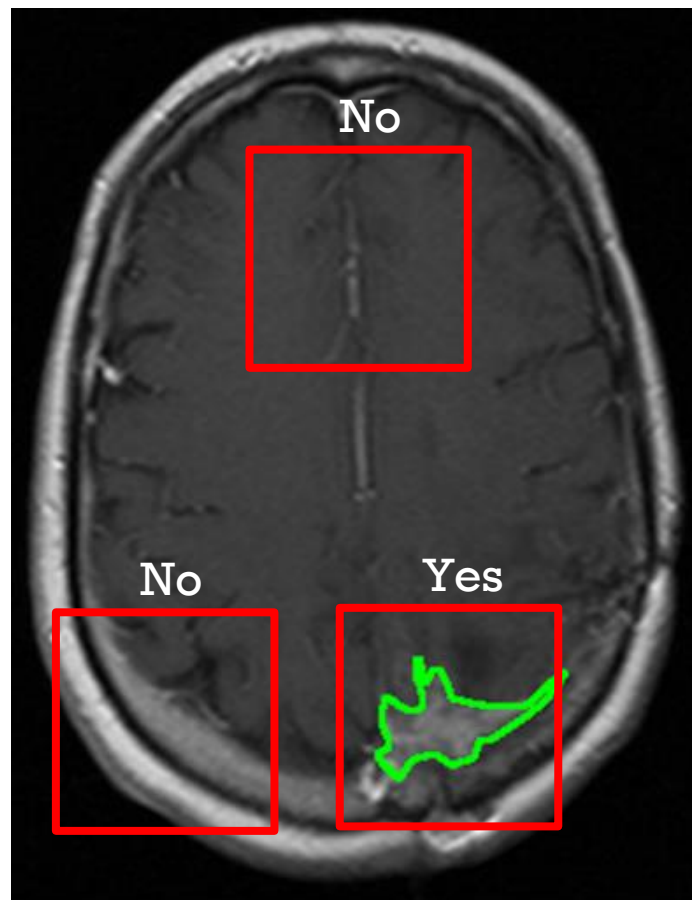


CLASSIFICATION

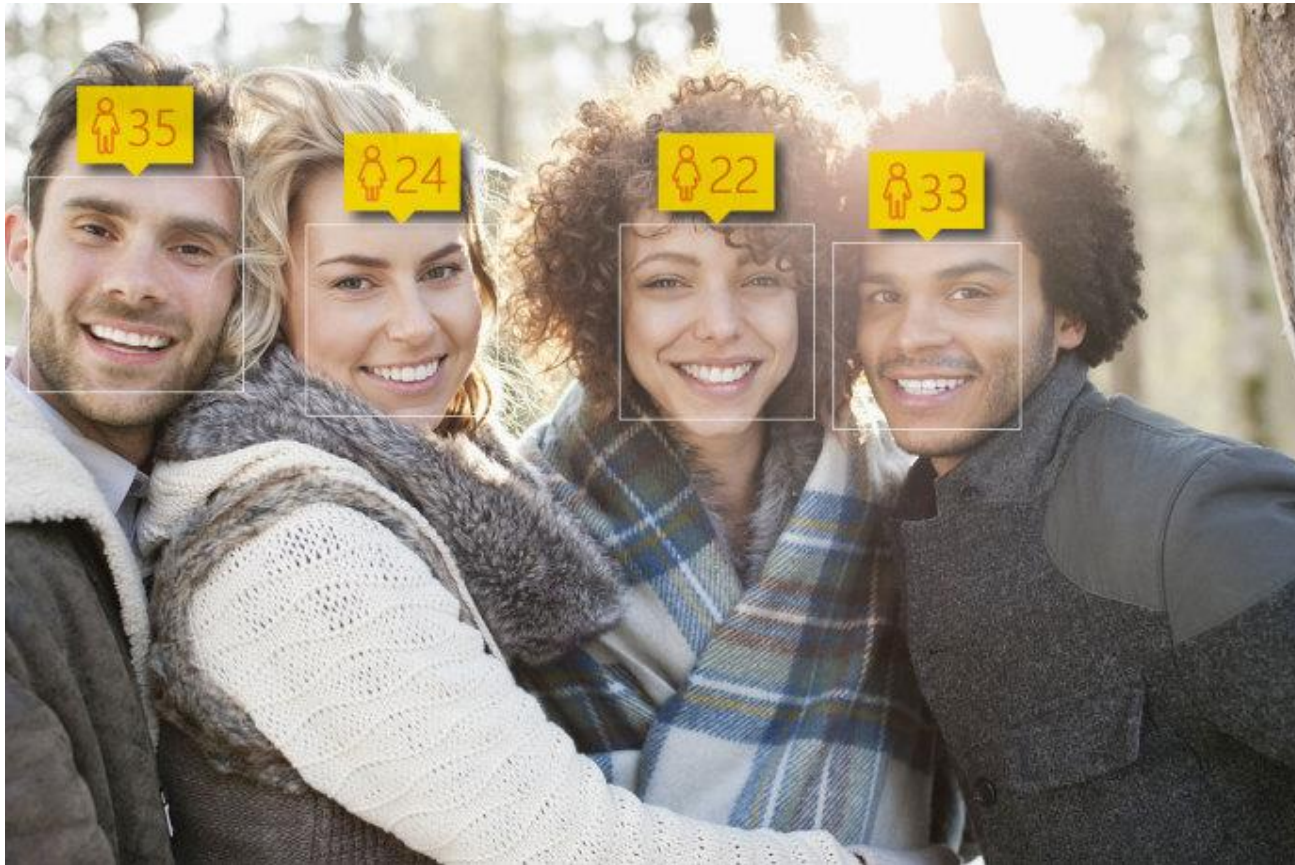


TUMOR CLASSIFICATION

Tumor?



GENDER FROM IMAGES



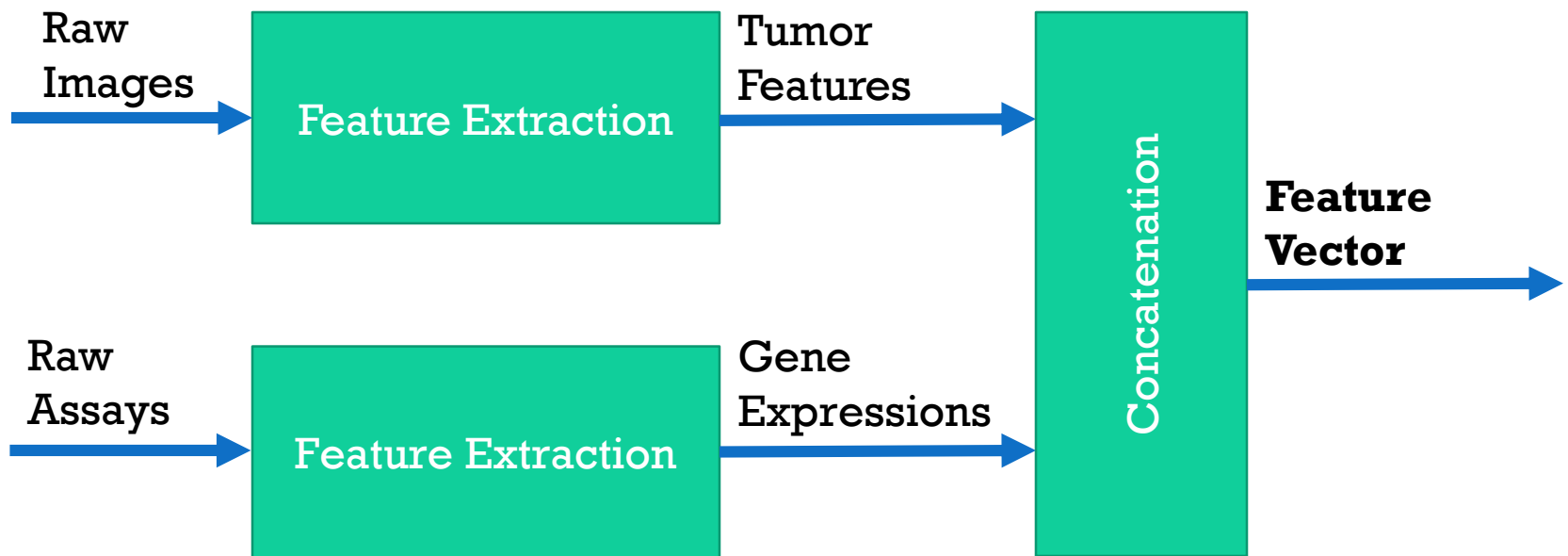
CLASSIFICATION

Machine Learning > Prediction > Classification

- **Task.** Classification
- **Experience.** Training Data
- **Performance.** Accuracy on New Data



FEATURE ENGINEERING



TRAINING DATA

- **Training data** $\mathcal{S}_n = \{ (x^{(i)}, y^{(i)}) \mid i = 1, \dots, n \}$
 - Feature vectors $x^{(i)} = (x_1^{(i)}, \dots, x_d^{(i)})^\top \in \mathbb{R}^d$
 - Labels $y^{(i)} \in \{-1, 1\}$



MODEL

- **Model** (or Hypothesis Class) \mathcal{H}
 - Classifier $h \in \mathcal{H}$, $h : \mathbb{R}^d \rightarrow \{-1, 1\}$



LEARNING ALGORITHM

- **Objective function** (or estimation criterion)
 $\mathcal{L}(h, \mathcal{S}_n)$
- Pick classifier $\hat{h} \in \mathcal{H}$ that minimizes $\mathcal{L}(h, \mathcal{S}_n)$.

This process is also called Training.



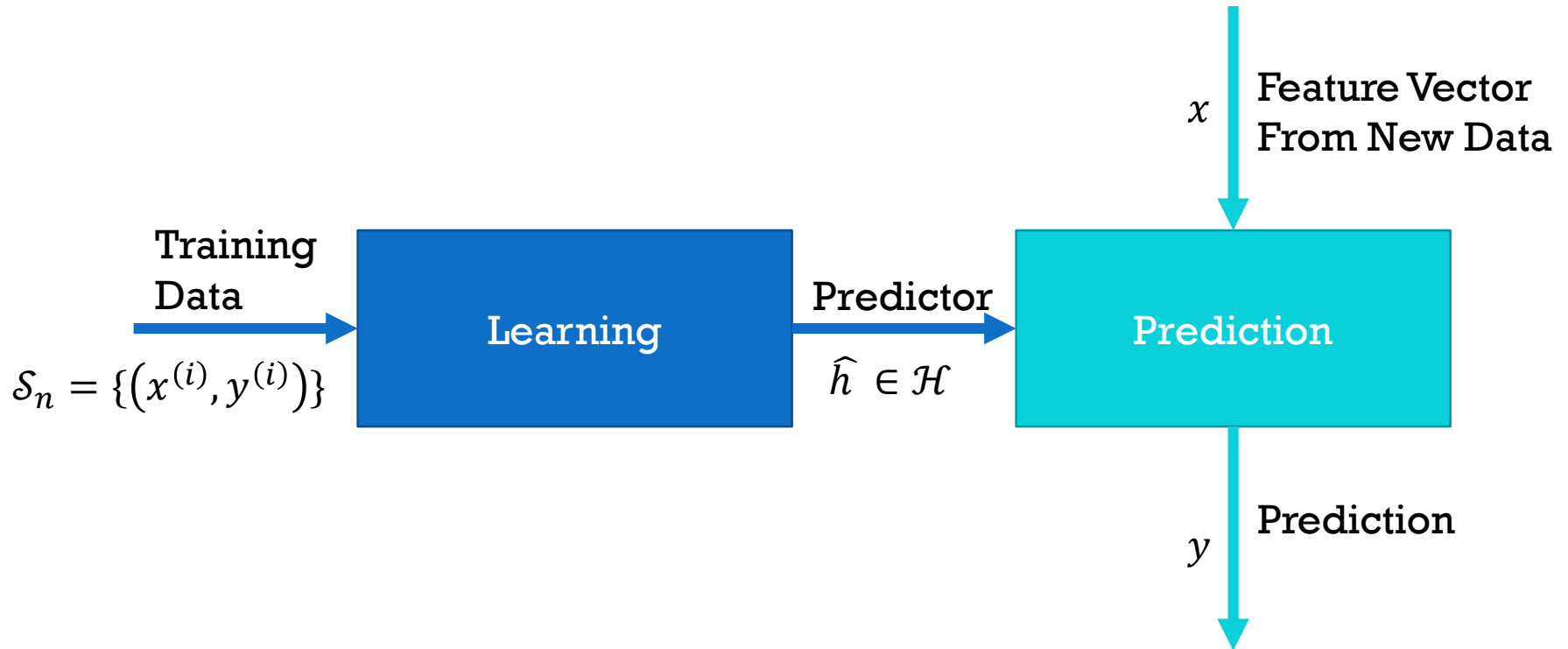
GENERALIZATION

The goal of machine learning is to find a classifier $\hat{h} \in \mathcal{H}$ that **generalizes** well, i.e. that predicts well on *unseen* data \mathcal{S}_* .

We often use some performance metric $\mathcal{R}(\hat{h}, \mathcal{S}_*)$ to measure how well a classifier \hat{h} generalizes. This metric can be different from $\mathcal{L}(h, \mathcal{S}_n)$.



LEARNING AND PREDICTION



Assumption. New data is **identically distributed** to training data.






WHAT NOT TO DO




GENDER FROM IMAGES

$$x = \begin{matrix} \text{128 pixels} \\ \text{128 pixels} \end{matrix} \begin{matrix} \text{128 pixels} \\ \text{128 pixels} \end{matrix} \in \mathbb{R}^{16384}$$


Training data $(x^{(1)}, y^{(1)}), \dots, (x^{(50)}, y^{(50)})$




GENDER FROM IMAGES

$$x = \begin{matrix} & 128 \text{ pixels} \\ & \text{128 pixels} \end{matrix} \begin{matrix} \text{128 pixels} \\ \text{128 pixels} \end{matrix} \in \mathbb{R}^{16384}$$


Model \mathcal{H} = set of all functions $h: \mathbb{R}^{16384} \rightarrow \{-1, 1\}$



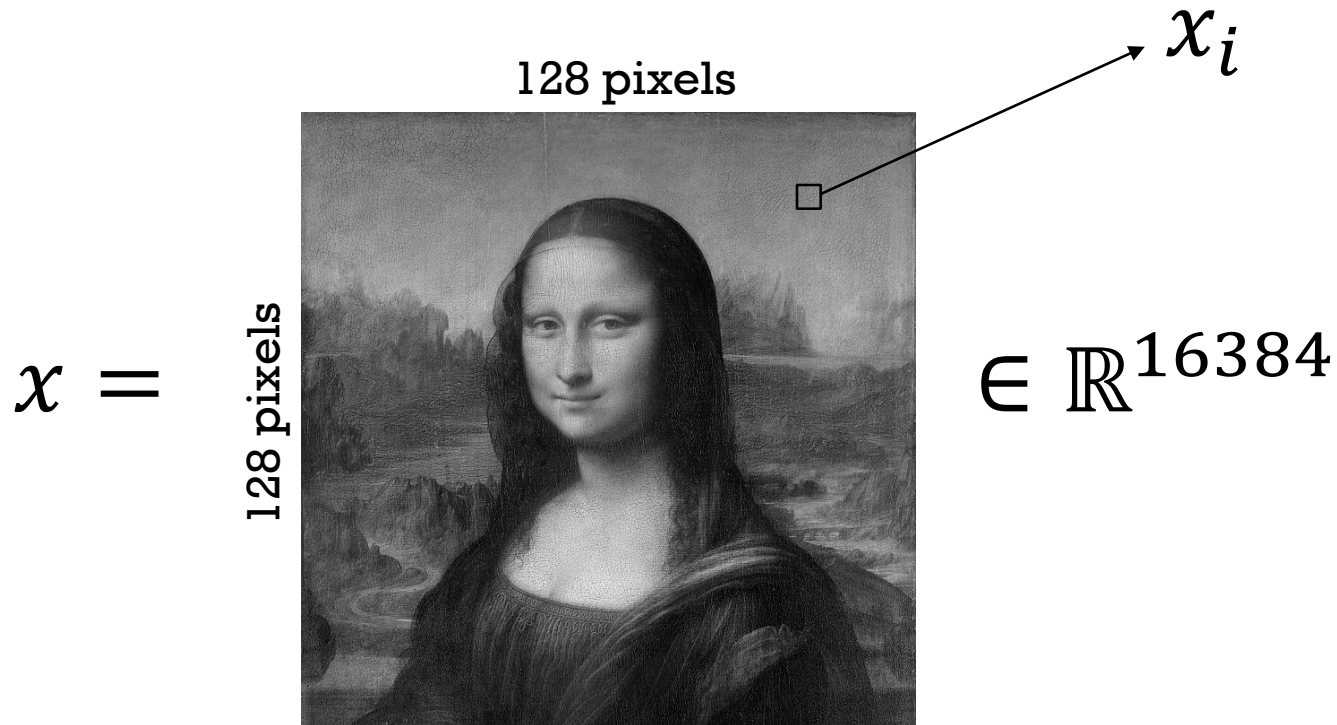
GENDER FROM IMAGES

$$x = \begin{matrix} & 128 \text{ pixels} \\ & \text{128 pixels} \end{matrix} \begin{matrix} \text{128 pixels} \\ \text{128 pixels} \end{matrix} \in \mathbb{R}^{16384}$$


Learning algorithm - minimize training error



SINGLE-PIXEL CLASSIFIER



$$\hat{h}(x) = \begin{cases} y^{(t)} & \text{if } x_i^{(t)} = x_i \text{ for some } t, \\ -1 & \text{otherwise.} \end{cases}$$



SINGLE-PIXEL CLASSIFIER

$$\hat{h}(x) = \begin{cases} y^{(t)} & \text{if } x_i^{(t)} = x_i \text{ for some } t, \\ -1 & \text{otherwise.} \end{cases}$$

- This classifier performs perfectly on training data.
- But it performs horribly (50% wrong) on new data.

What went wrong?

- Model \mathcal{H} was too big.
- Classifier ended up **overfitting** the training data.



MODEL SELECTION

If model \mathcal{H} is too big, then $\hat{h} \in \mathcal{H}$

- performs well on training data, but poorly on new data.

If model \mathcal{H} is too small (e.g. one element set), then $\hat{h} \in \mathcal{H}$

- performs poorly on training data, and poorly on new data.

Finding a model with the right size is called **model selection**.





LINEAR CLASSIFICATION



LINEAR CLASSIFIERS THROUGH ORIGIN

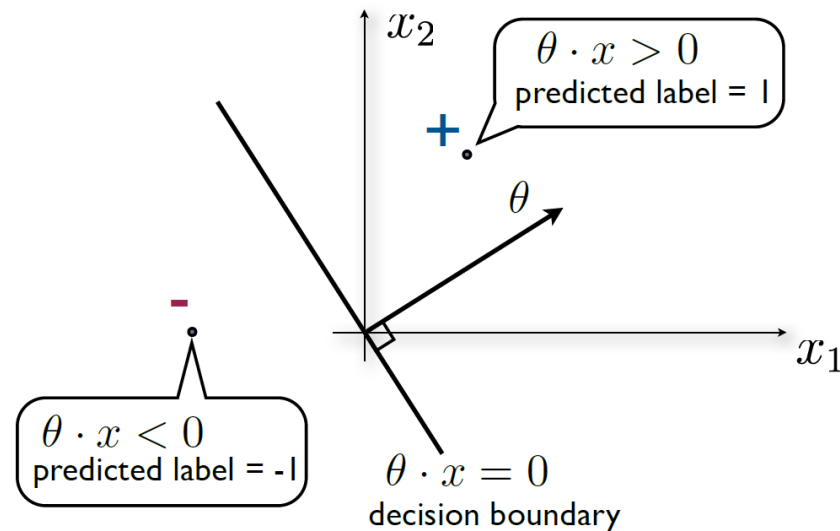
Let the model \mathcal{H} be the set of classifiers of the form

$$\begin{aligned} h(x; \theta) &= \text{sign}(\theta_1 x_1 + \cdots + \theta_d x_d) \\ &= \text{sign}(\theta \cdot x) \\ &= \text{sign}(\theta^\top x) \\ &= \begin{cases} +1 & \text{if } \theta^\top x \geq 0, \\ -1 & \text{if } \theta^\top x < 0. \end{cases} \end{aligned}$$

We say that \mathcal{H} is **parametrized** by the vector θ .



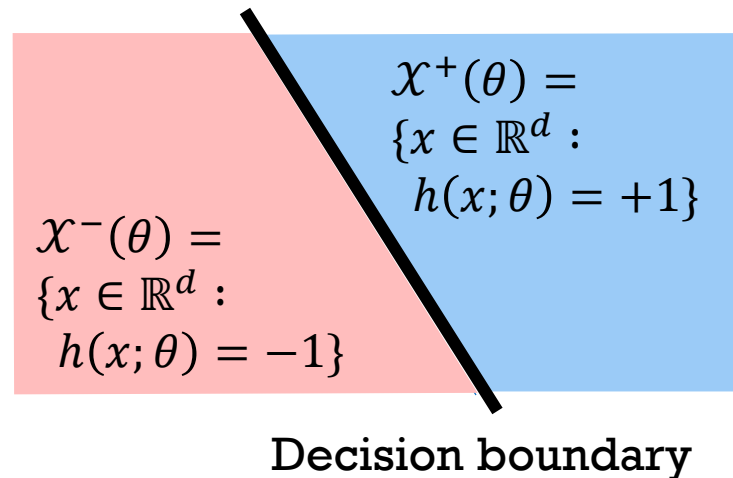
DECISION BOUNDARIES



- Decision boundary is a **hyperplane** of dimension $d - 1$ that passes through the origin.
- Vector θ is orthogonal to the decision boundary.
- Vector θ points in direction of greatest increase to $g(x) = \theta^\top x$.



HALF SPACES



A classifier h partitions the space into regions that are separated by decision boundaries. In each region, all the points map to the same label. Many regions could have the same label.

For linear classifiers, these regions are half spaces.



OBJECTIVE FUNCTION

Choose classifier that minimizes the *training error*

$$\begin{aligned}\mathcal{E}_n(\theta) &= \frac{1}{n} \sum_{t=1}^n \mathbb{I} \left[y^{(t)} (\theta^\top x^{(t)}) \leq 0 \right] \\ &\approx \frac{1}{n} \sum_{t=1}^n \mathbb{I} \left[y^{(t)} \neq h(x^{(t)}; \theta) \right]\end{aligned}$$

Here, $\mathbb{I}[\cdot]$ is the *indicator* function that returns a 1 if the logical statement in its argument is true, and 0 otherwise.

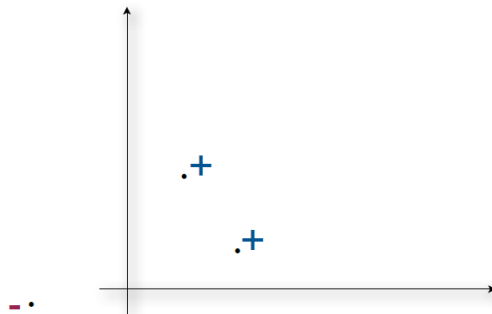
Note that $y^{(t)} (\theta^\top x^{(t)}) \leq 0$ if and only if

- $y^{(t)}$ and $\theta^\top x^{(t)}$ have different signs, or
- $\theta^\top x^{(t)}$ is exactly zero.

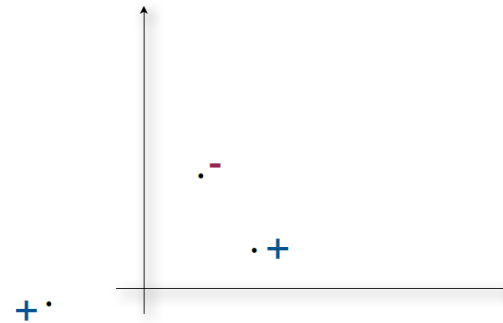


LINEARLY SEPARABLE THROUGH ORIGIN

The training data \mathcal{S}_n is **linearly separable through origin** if there exists a parameter vector $\hat{\theta}$ such that $y^{(t)}(\hat{\theta}^\top x^{(t)}) > 0$ for all t .



linearly separable through origin



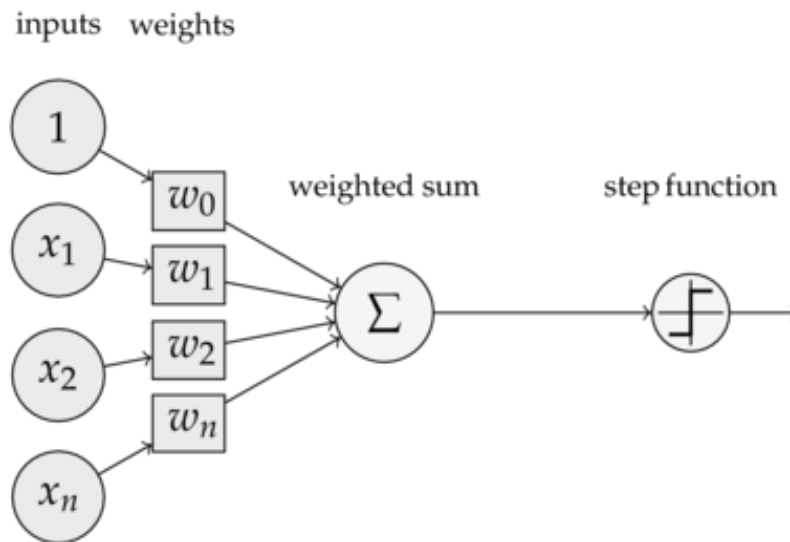
not linearly separable

Challenge. How do you prove the training data on the right is not linearly separable?



PERCEPTRON

Linear classifiers are often also called **perceptrons**.



For linear classifiers through origin, the weight w_0 is zero. In general, it could be non-zero. We will see this later.



PERCEPTRON ALGORITHM

Mistake driven

1. Initialize $\theta = 0$
2. For each t from 1 to n ,
 - a. Check if $h(x^{(t)}; \theta) = y^{(t)}$.
 - b. If not, update θ to correct the mistake.
3. Stop if no mistakes were found in Step (2). Otherwise, go back to Step (2).



PERCEPTRON ALGORITHM

Mistake driven

1. Initialize $\theta^{(0)} = 0$ and $k = 0$.
2. For each t from 1 to n ,
 - a. If $y^{(t)}(\theta^{(k)\top} x^{(t)}) \leq 0$,
 - i. Update $\theta^{(k+1)} = \theta^{(k)} + y^{(t)} x^{(t)}$.
 - ii. Increase k by 1.
3. If no mistakes were found in Step (2), stop. Otherwise, go back to Step (2).

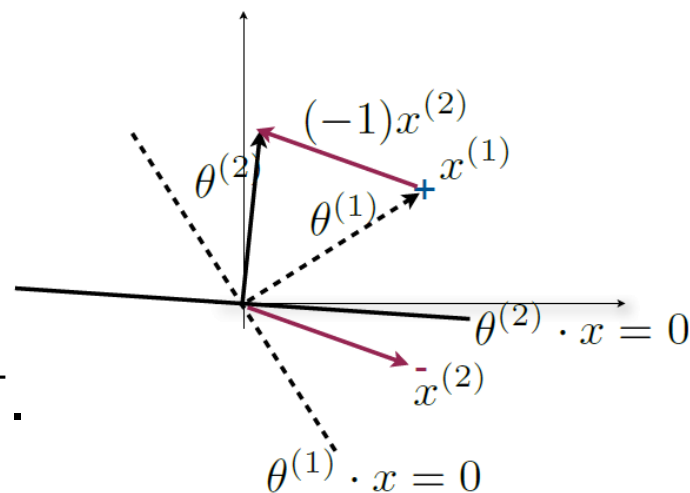
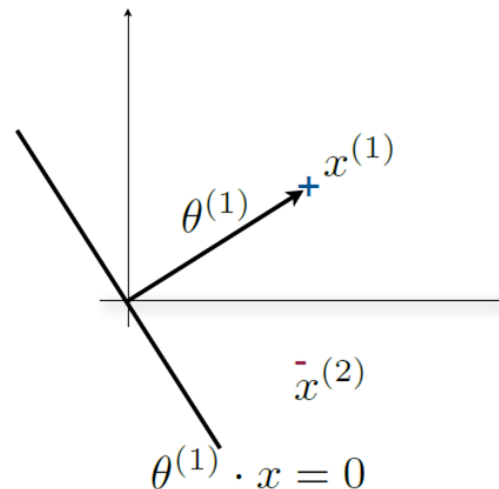


EXAMPLE

Training data

- $(x^{(1)}, y^{(1)}) = ((2, 2)^T, +1)$
- $(x^{(2)}, y^{(2)}) = ((2, -1)^T, -1)$

- Initialize $\theta^{(0)} = 0$.
- Since $y^{(1)}\theta^{(0)\top}x^{(1)} = 0$,
update $\theta^{(1)} = 0 + (2, 2)^T = (2, 2)^T$.
- Since $y^{(2)}\theta^{(1)\top}x^{(2)} = -2$,
update $\theta^{(2)} = (2, 2)^T - (2, -1)^T = (0, 3)^T$.
- $y^{(1)}\theta^{(2)\top}x^{(1)} = 6 > 0$.
- $y^{(2)}\theta^{(2)\top}x^{(2)} = 3 > 0$.
- No more mistakes, so we are done.



WHY DOES IT WORK?

Recall that the goal is to find θ such that $y^{(t)}(\theta^\top x^{(t)}) > 0$ for all the training samples.

$$\begin{aligned} y^{(t)} \theta^{(k+1)\top} x^{(t)} &= y^{(t)} (\theta^{(k)} + y^{(t)} x^{(t)})^\top x^{(t)} \\ &= y^{(t)} \theta^{(k)\top} x^{(t)} + (y^{(t)})^2 x^{(t)\top} x^{(t)} \\ &= y^{(t)} \theta^{(k)\top} x^{(t)} + \|x^{(t)}\|^2 \end{aligned}$$

So the value of $y^{(t)}(\theta^\top x^{(t)})$ increased. However, this does not guarantee that its value becomes positive, or that its value for other data points does not decrease.



WHY DOES IT WORK?

Theorem. If the training data is linearly separable, then the perceptron algorithm terminates after a finite number of steps.

Proof. Not in the scope of this class, but it is not difficult. Basic idea is compare certain lower and upper bounds for $\|\theta^{(k)}\|$.

Non linearly-separable. In this case, the perceptron algorithm will never terminate, because there will always be a mistake for all values of $\theta^{(k)}$. Other learning algorithms are needed.



PERCEPTRON WITHOUT OFFSET

Summary

1. Training Set (**Linearly Separable through Origin**)

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})$$

2. Model (**Linear Classifiers through Origin**)

$$h(x; \theta) = \text{sign}(\theta_1 x_1 + \dots + \theta_d x_d)$$

3. Learning

- a. Objective Function (**Fraction of Misclassified Points**)

$$\mathcal{E}_n(\theta) = \frac{1}{n} \sum_{t=1}^n \mathbb{I} [y^{(t)} (\theta^\top x^{(t)}) \leq 0]$$

- a. Algorithm (**Perceptron Algorithm without Offset**)



LINEAR CLASSIFIERS WITH OFFSET

Let the model \mathcal{H} be the set of classifiers of the form

$$h(x; \theta) = \text{sign}(\theta_1 x_1 + \cdots + \theta_d x_d + \theta_0) = \text{sign}(\theta^\top x + \theta_0)$$

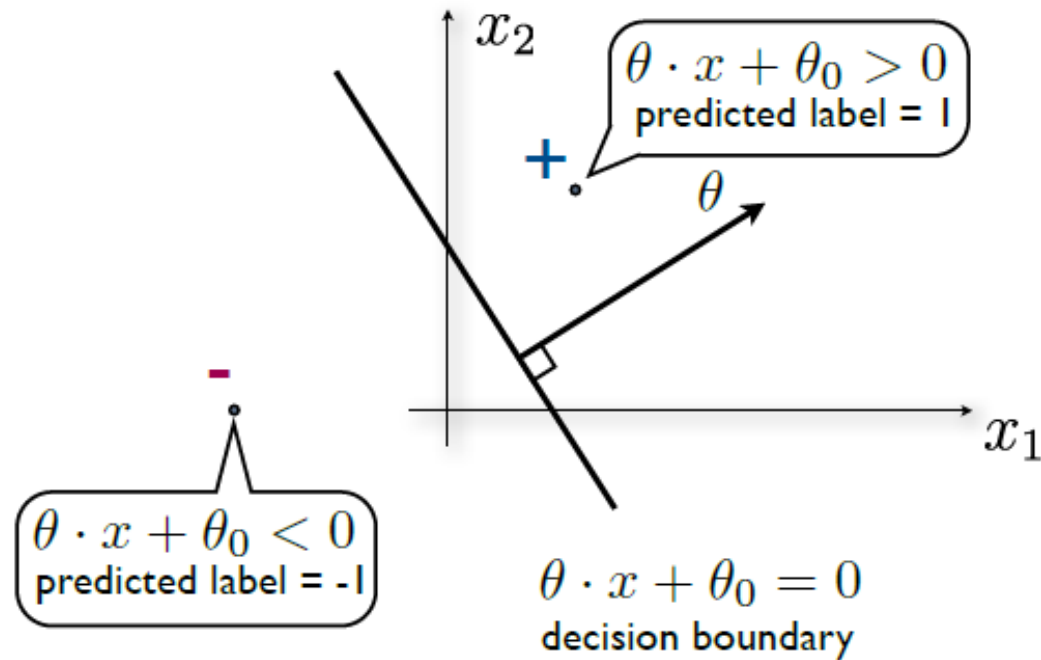
Scalar parameter θ_0 is called the *offset*.

Same as introducing extra '1' to feature vector x

$$h(x; \theta) = \text{sign}(\theta_1 x_1 + \cdots + \theta_d x_d + \theta_0(1))$$
$$(x_1, \quad \dots, \quad x_d, \quad 1)$$



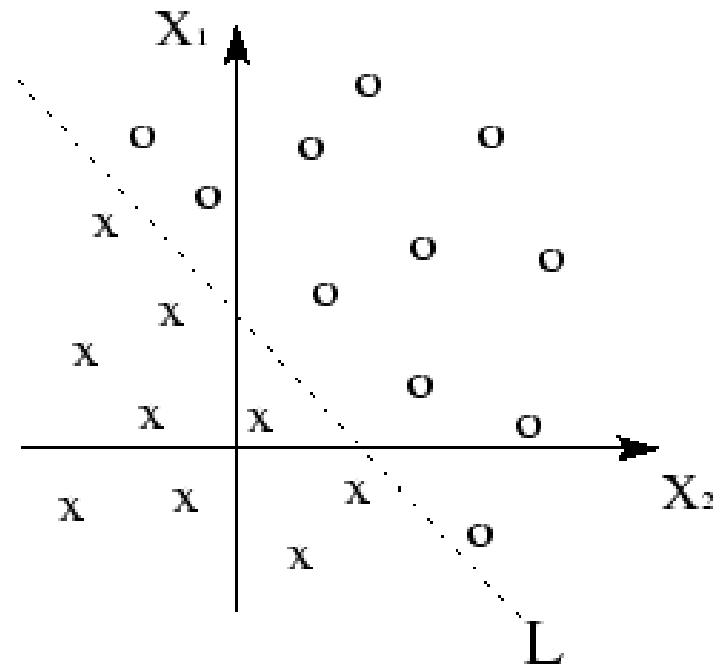
LINEAR CLASSIFIERS WITH OFFSET



LINEARLY SEPARABLE

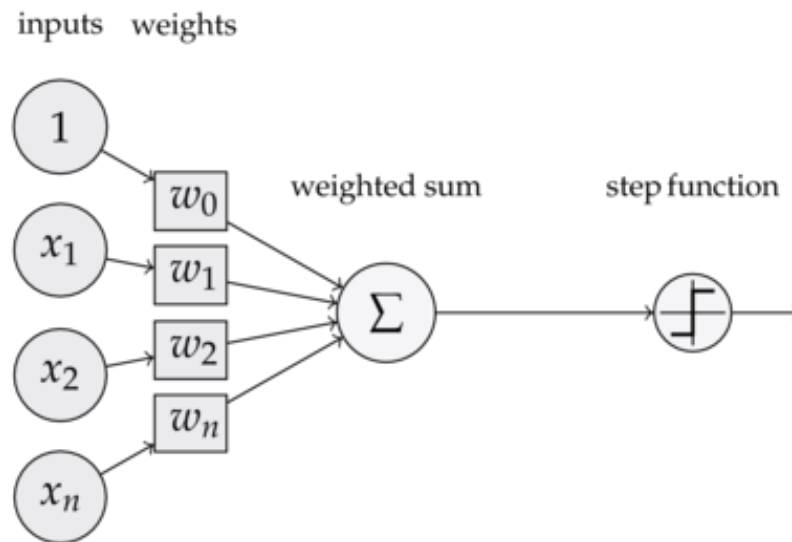
The training data \mathcal{S}_n is **linearly separable** if there exists a parameter vector $\hat{\theta}$ and offset $\hat{\theta}_0$ such that for all t ,

$$y^{(t)}(\hat{\theta}^\top x^{(t)} + \hat{\theta}_0) > 0.$$



PERCEPTRON

Historically, perceptrons were developed with offsets.



PERCEPTRON ALGORITHM WITH OFFSET

Mistake driven

1. Initialize $\theta^{(0)} = 0$, $\theta_0^{(0)} = 0$ and $k = 0$.
2. For each t from 1 to n ,
 - a. If $y^{(t)} \left(\theta^{(k)\top} x^{(t)} + \theta_0^{(k)} \right) \leq 0$,
 - i. Set $\theta^{(k+1)} = \theta^{(k)} + y^{(t)} x^{(t)}$.
 - ii. Set $\theta_0^{(k+1)} = \theta_0^{(k)} + y^{(t)}$.
 - iii. Increase k by 1.
3. If no mistakes were found in Step (2), stop. Otherwise, go back to Step (2).



PERCEPTRON WITH OFFSET

Summary

1. Training Set (**Linearly Separable**)

$$(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})$$

2. Model (**Linear Classifiers**)

$$h(x; \theta) = \text{sign}(\theta_1 x_1 + \dots + \theta_d x_d + \theta_0)$$

3. Learning

- a. Objective Function (**Fraction of Misclassified Points**)

$$\mathcal{E}_n(\theta) = \frac{1}{n} \sum_{t=1}^n \mathbb{I} [y^{(t)} (\theta^\top x^{(t)} + \theta_0) \leq 0]$$

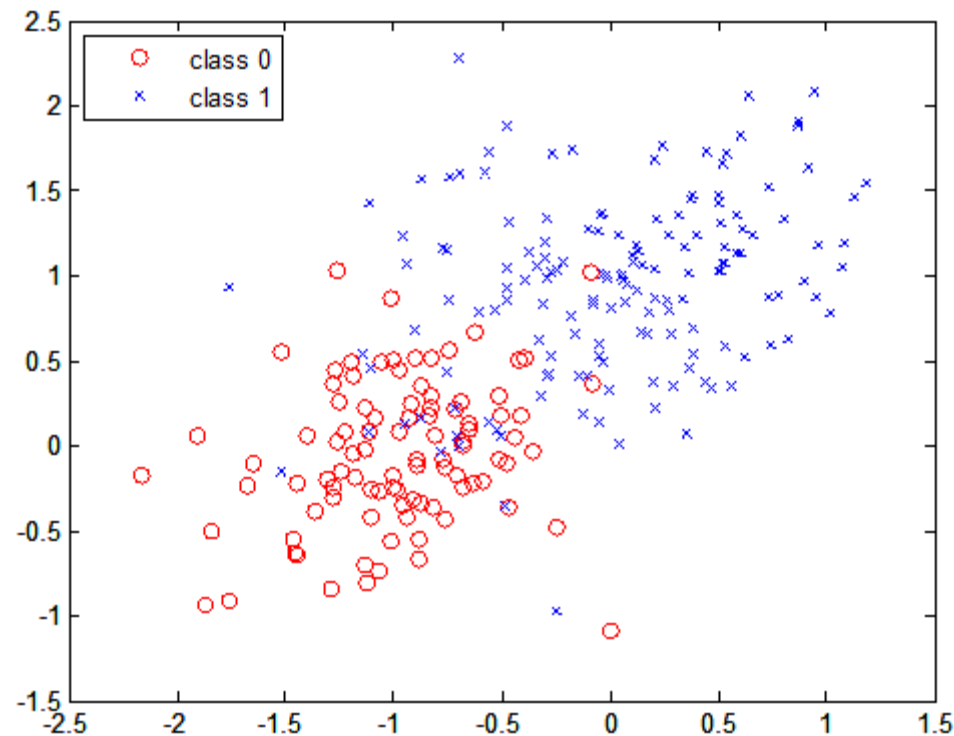
- a. Algorithm (**Perceptron Algorithm**)





LINEAR CLASSIFICATION (NON-SEPARABLE)





Perceptron algorithm does not converge for training sets that are not linearly separable.



LOSS FUNCTIONS

Training Error for Perceptron Algorithm

$$\mathcal{E}_n(\theta) = \frac{1}{n} \sum_{t=1}^n \mathbb{I}[z \leq 0],$$

$$z = y^{(t)}(\theta^\top x^{(t)} + \theta_0)$$

General Form

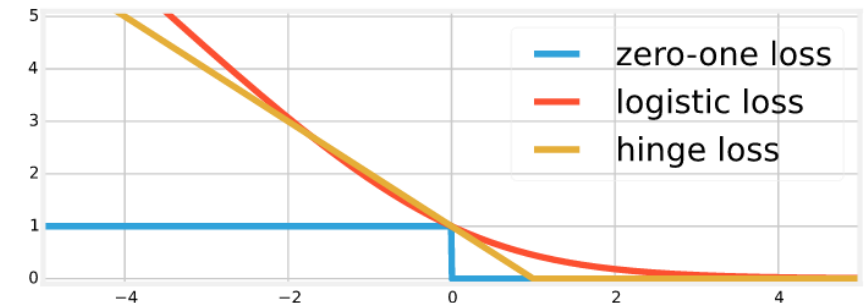
$$\mathcal{E}_n(\theta) = \frac{1}{n} \sum_{t=1}^n \text{Loss}(z)$$

Zero-One Loss

$$\text{Loss}_{01}(z) = \mathbb{I}[z \leq 0]$$

Hinge Loss

$$\text{Loss}_H(z) = \max\{1 - z, 0\}$$

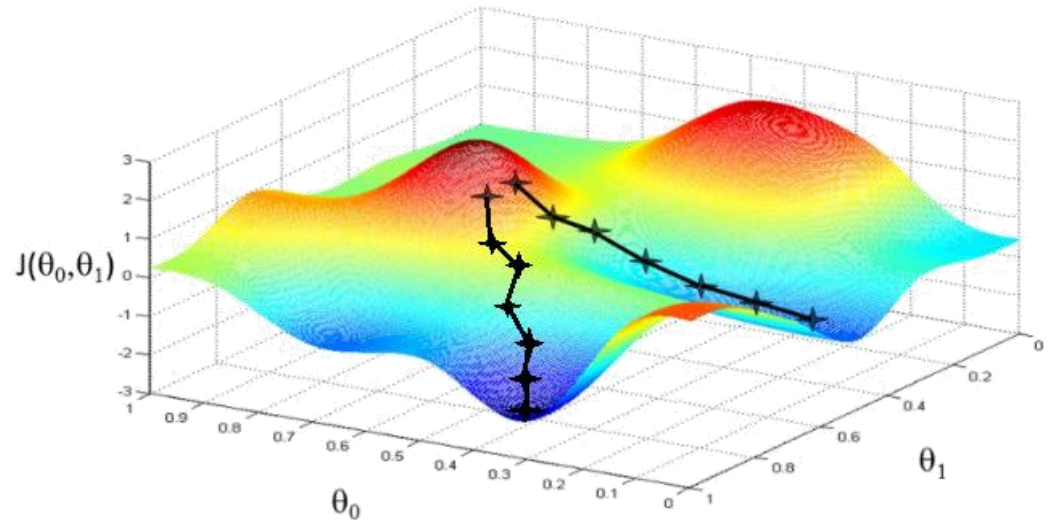


CONVEX!



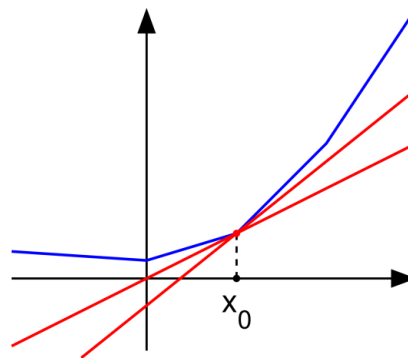
OPTIMIZATION

Gradient Descent



$$\theta^{(k+1)} = \theta^{(k)} - \eta_k \nabla_{\theta} \mathcal{E}_n(\theta) |_{\theta=\theta^{(k)}} , \quad \eta_k \text{ learning rate}$$

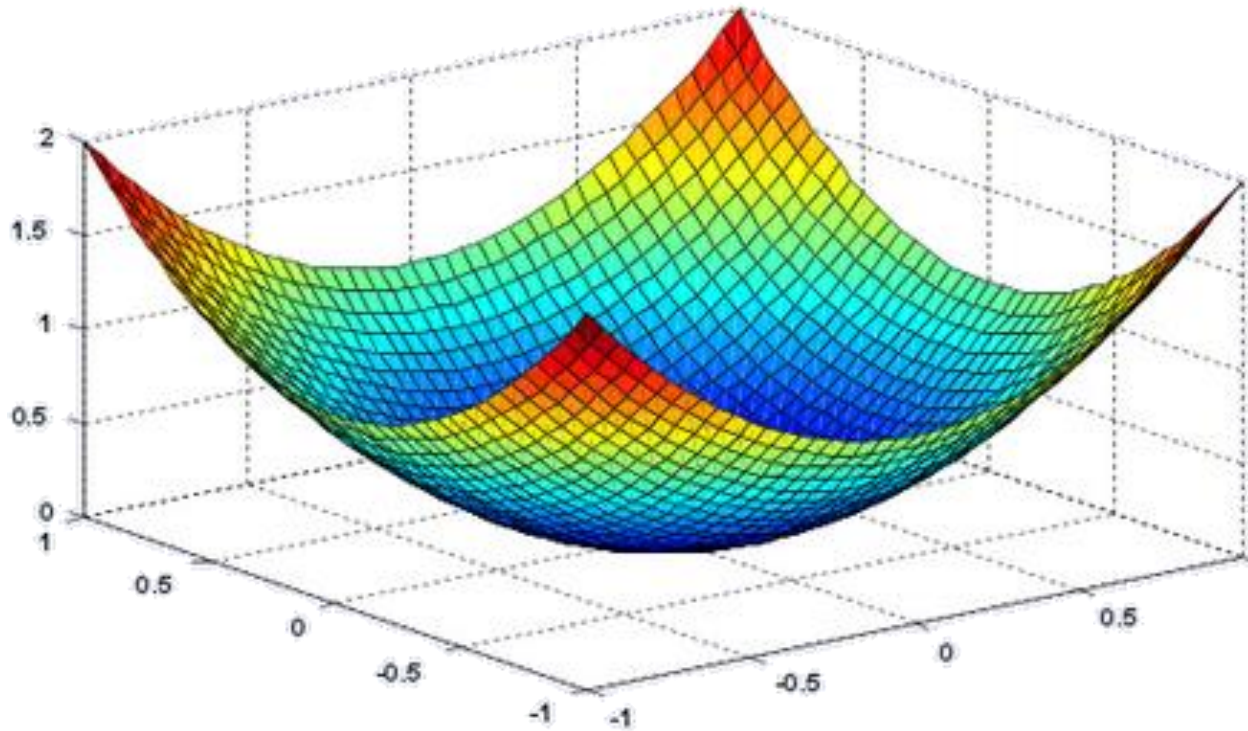
Sub-Gradients $\partial f(x)$ and Descent



For those interested, $\partial f(x)$ is the set of all vectors v such that for all y ,
 $f(y) - f(x) \geq v^T (y - x)$.



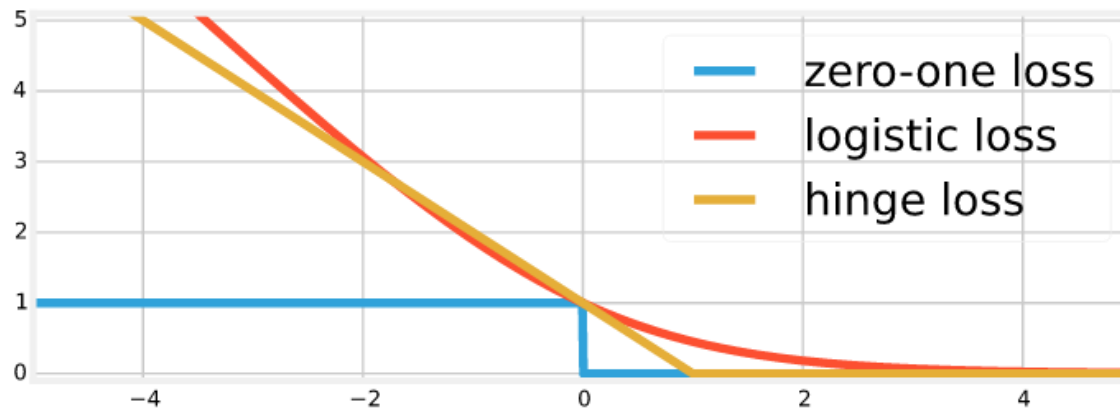
CONVEX OPTIMIZATION



Local Minimum = Global Minimum.
Fast Algorithms.



HINGE LOSS GRADIENT



$$\nabla_z \text{Loss}_H(z) = \begin{cases} 0 & \text{if } z > 1, \\ -1 & \text{otherwise.} \end{cases}$$

$$\nabla_{\theta} \text{Loss}_H \left(y^{(t)} (\theta^{\top} x^{(t)} + \theta_0) \right) = \begin{cases} 0 & \text{if } y^{(t)} (\theta^{\top} x^{(t)} + \theta_0) > 1, \\ -y^{(t)} x^{(t)} & \text{otherwise.} \end{cases}$$



STOCHASTIC GRADIENT DESCENT

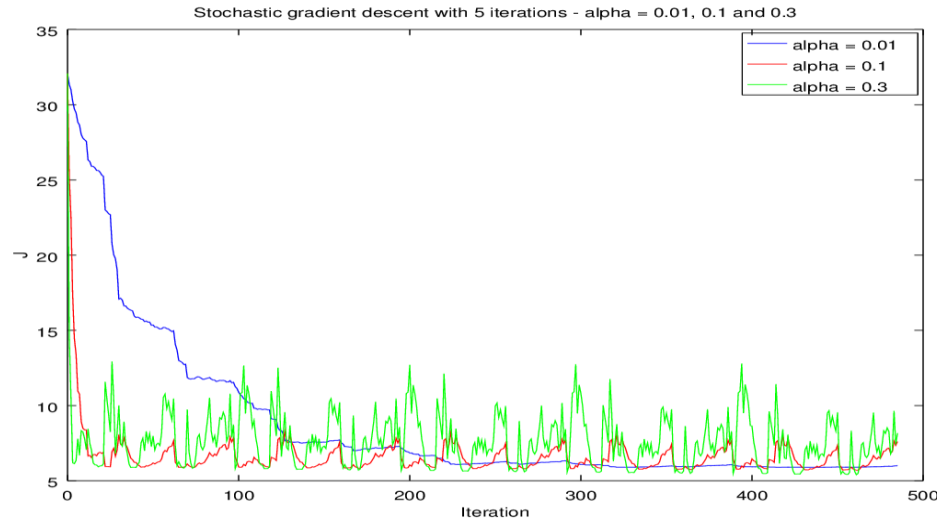
1. Initialize $\theta^{(0)} = 0$ and $k = 0$.
2. Select $t \in \{1, 2, \dots, n\}$ at random.
 - a. If $y^{(t)}(\theta^{(k)\top} x^{(t)}) \leq 1$,
 - i. Set $\theta^{(k+1)} = \theta^{(k)} + \eta_k y^{(t)} x^{(t)}$.
 - ii. Increase k by 1.
3. Repeat (2) until convergence.

Differences from Perceptron Algorithm

- Check $z \leq 1$ rather than $z \leq 0$
- Decreasing η_k rather than $\eta = 1$
- Randomly choosing samples rather than cycling through them



STOCHASTIC GRADIENT DESCENT



1. Choosing the learning rate so that $\theta^{(k)}$ converges.

$$\eta_k = \frac{1}{k+1}$$

2. Keeping track of the best result $\hat{\theta}$ (minimizes hinge loss) and outputting $\hat{\theta}$ when the algorithms stops, rather than current parameter vector $\theta^{(k)}$.

