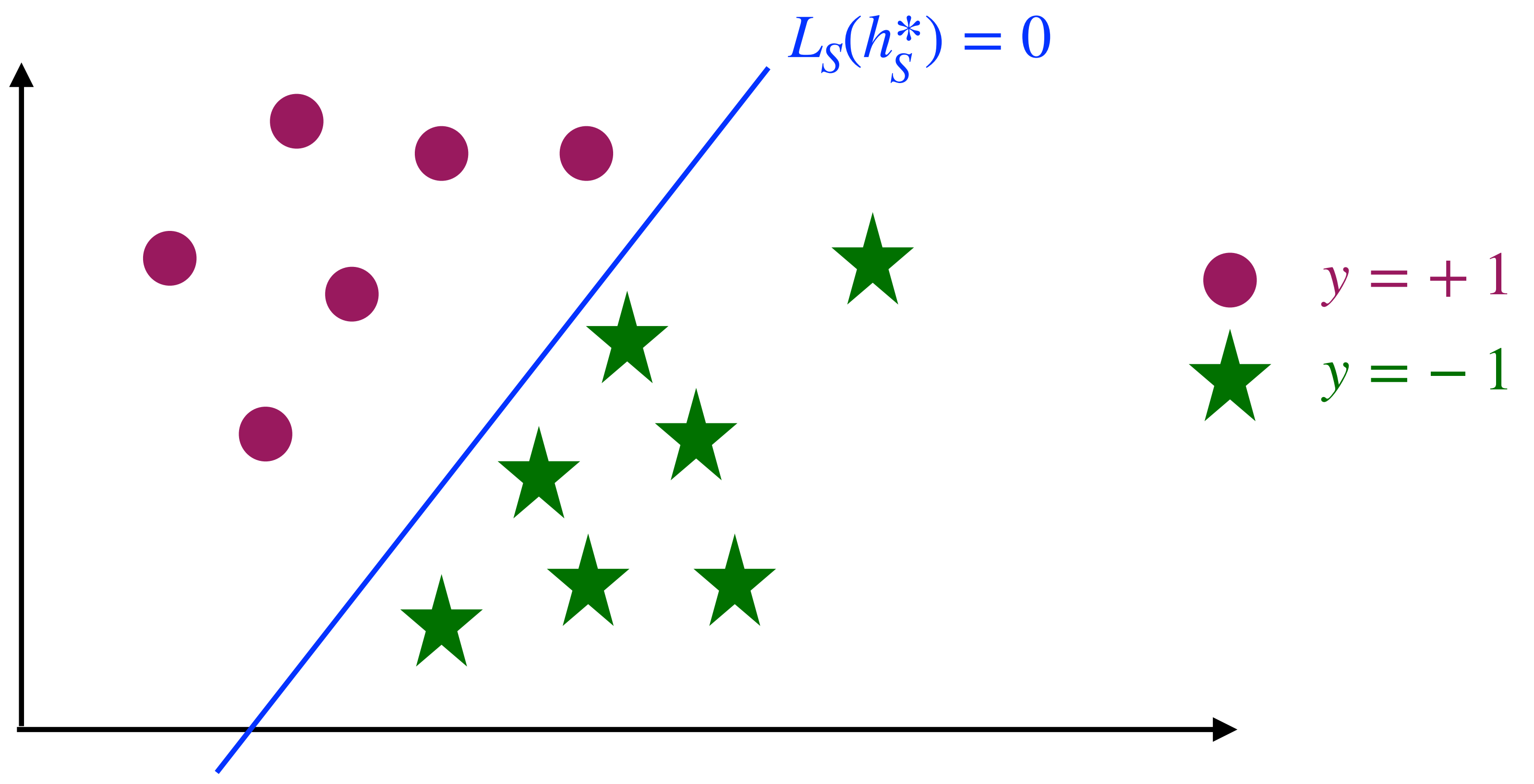
Machine Learning for Physical Scientists

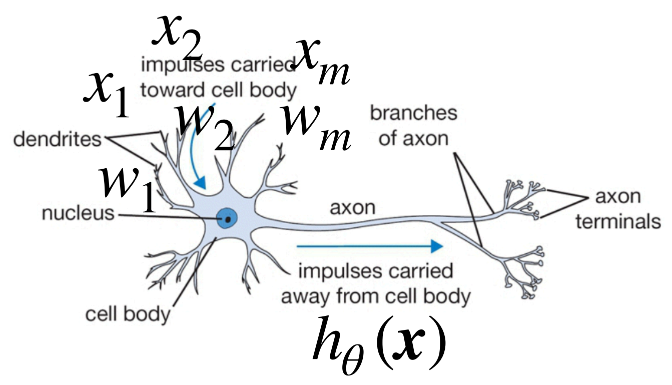
Lecture 6
Logistic Regression

(Combining Bayesian Inference with ERM for Classification)

Simplest Binary Classification: the Perceptron Algorithm

The first model of “neural computation” (Rosenblatt 1958)





x_1
x_2
x_m
impulses carried toward cell body
w_1
w_2
w_m
dendrites
nucleus
cell body
axon
branches of axon
axon terminals
impulses carried away from cell body
$h_\theta(\boldsymbol{x})$

$$h_\theta(\boldsymbol{x}) = \operatorname{sign}\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$$

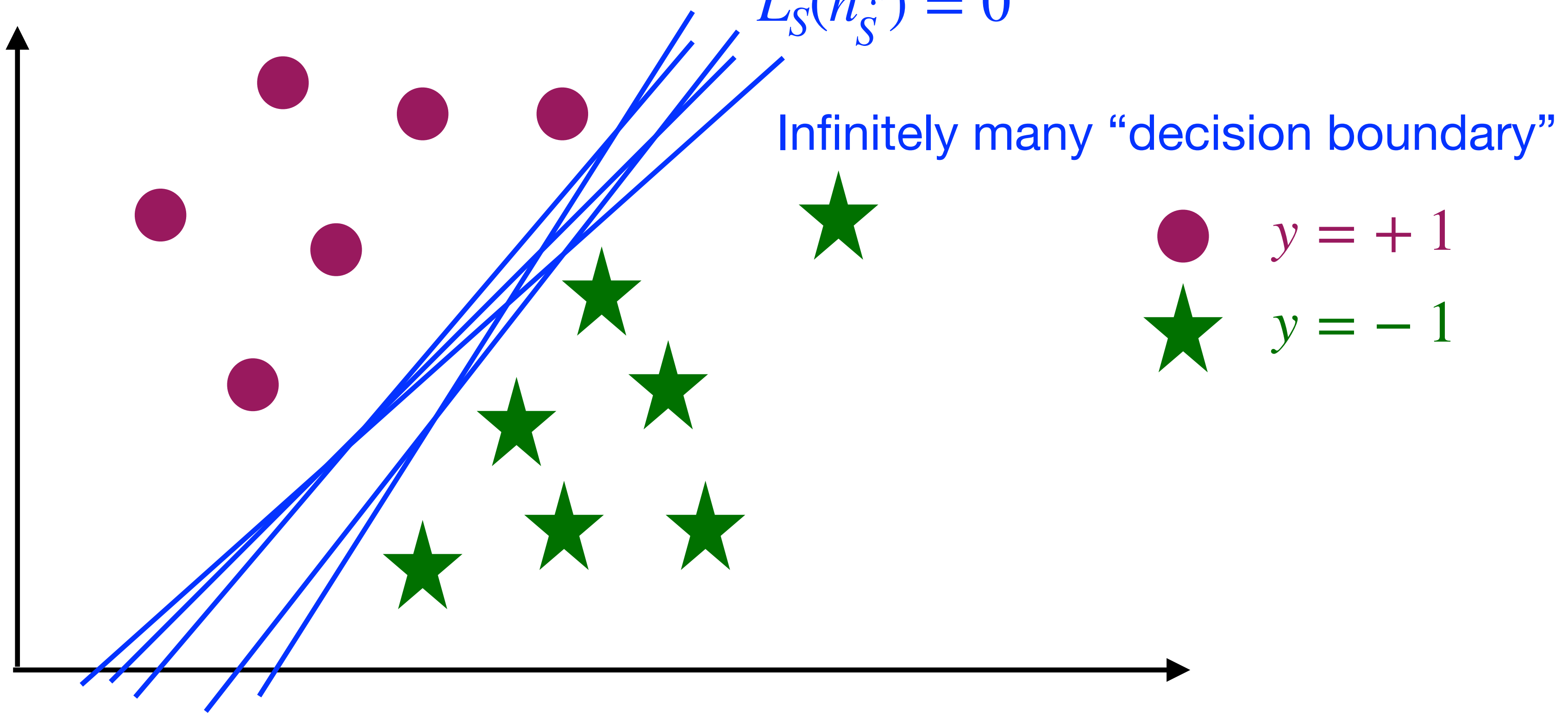
$$\ell(y_i, h_\theta(\boldsymbol{x}_i)) = \mathbf{1}_{h_\theta(\boldsymbol{x}_i) \neq y_i}$$ (misclassification costs 1 point deduction)

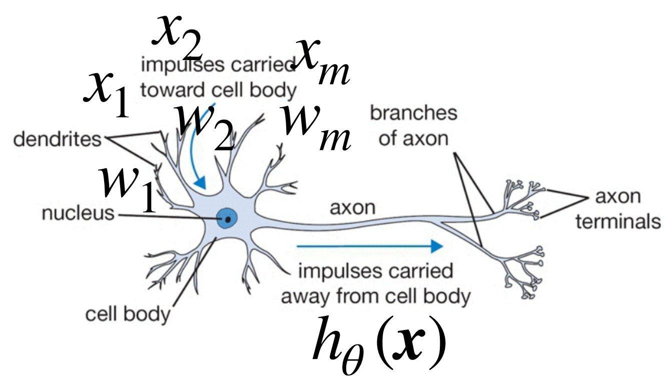
$$L_S(h) = \frac{1}{m}\sum_{i=1}^{m} \ell(y_i, h_\theta(\boldsymbol{x}_i))$$

A perceptron learning algorithm guarantees convergence to an empirically optimal hypothesis, *if the training set is linearly separable.*

Simplest Binary Classification: the Perceptron Algorithm

The first model of “neural computation” (Rosenblatt 1958)





x_2
x_1
x_m
impulses carried toward cell body
w_2
w_m
w_1
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body
$h_\theta(\boldsymbol{x})$

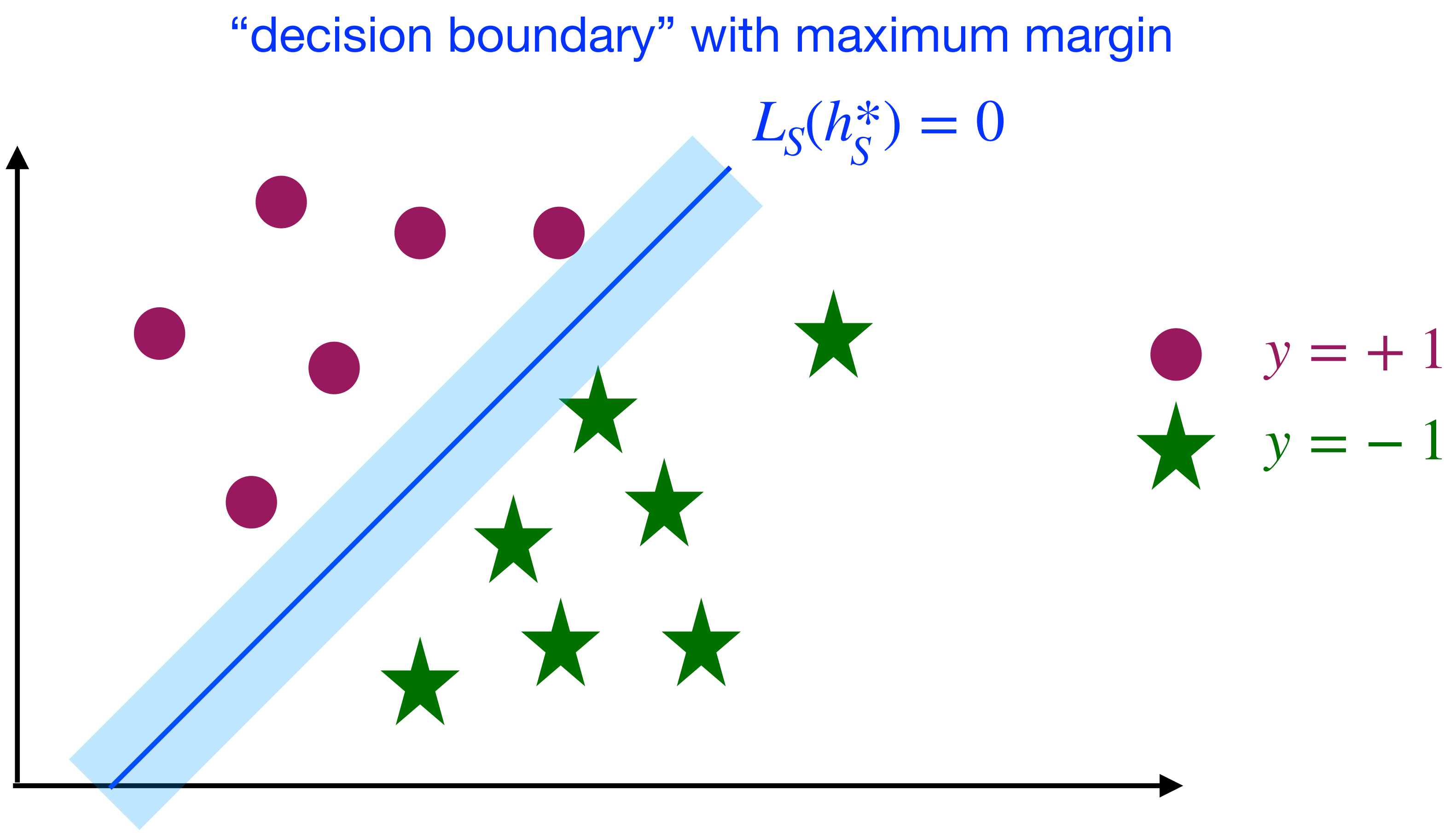
$$h_\theta(\boldsymbol{x}) = \operatorname{sign}\left(\boldsymbol{w}^T\boldsymbol{x} + b\right)$$

$$\ell(y_i, h_\theta(\boldsymbol{x}_i)) = \mathbf{1}_{h_\theta(\boldsymbol{x}_i)\neq y_i}$$ (misclassification costs 1 point deduction)

$$L_S(h) = \frac{1}{m}\sum_{i=1}^{m}\ell(y_i, h_\theta(\boldsymbol{x}_i))$$

A perceptron learning algorithm guarantees convergence to an empirically optimal hypothesis, *if the training set is linearly separable. But there’s an infinitely many solution.*

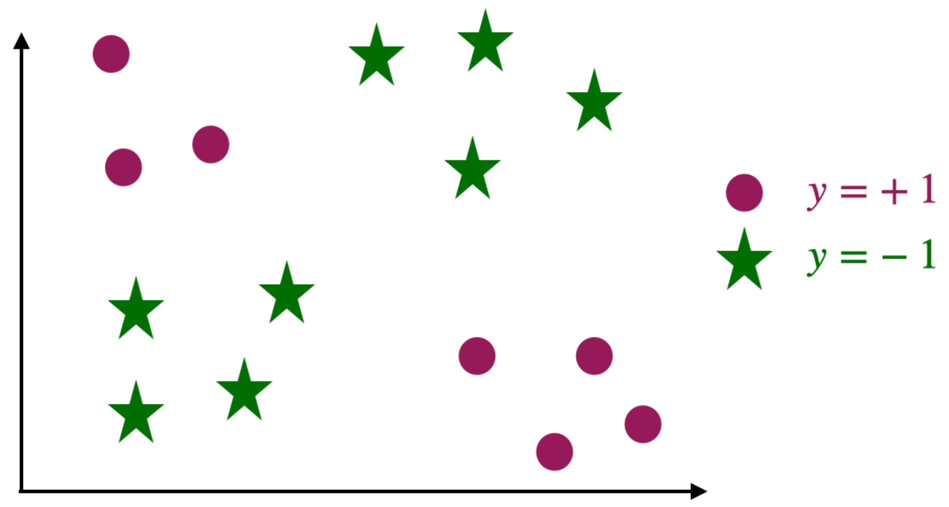
Linear Support Vector Machine (LSVM)



LSVM searches for the empirically optimal hypothesis with *maximum margin*.

However, linear classification model can not classify non-linearly-separable dataset

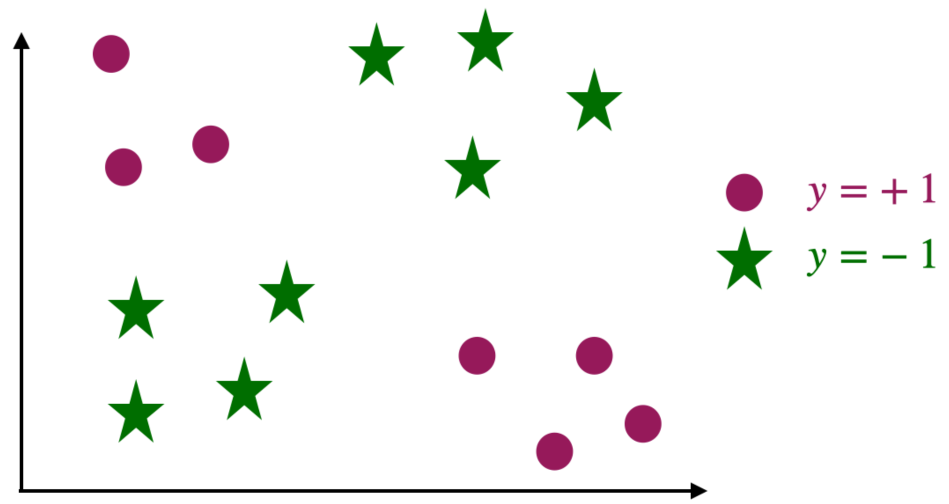
XOR problem



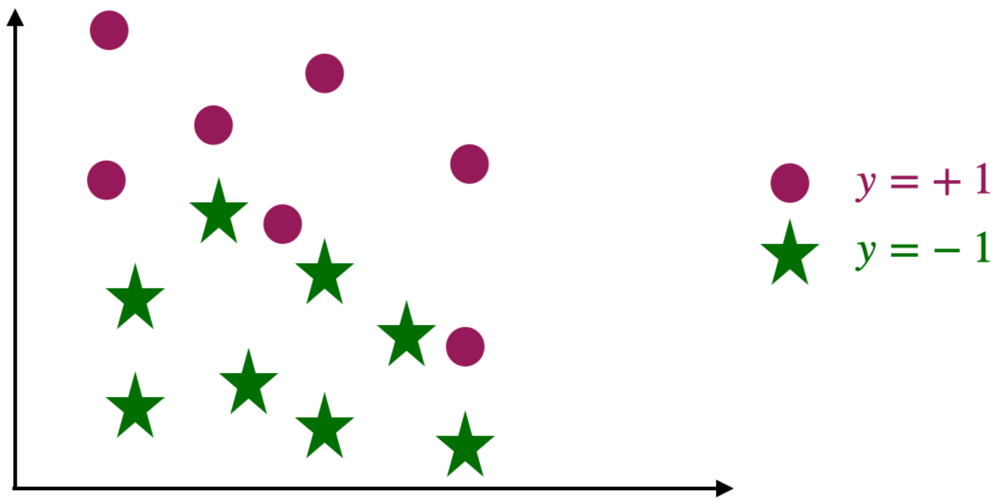
Kernel method (later)

However, linear classification model can not classify non-linearly-separable dataset

XOR problem



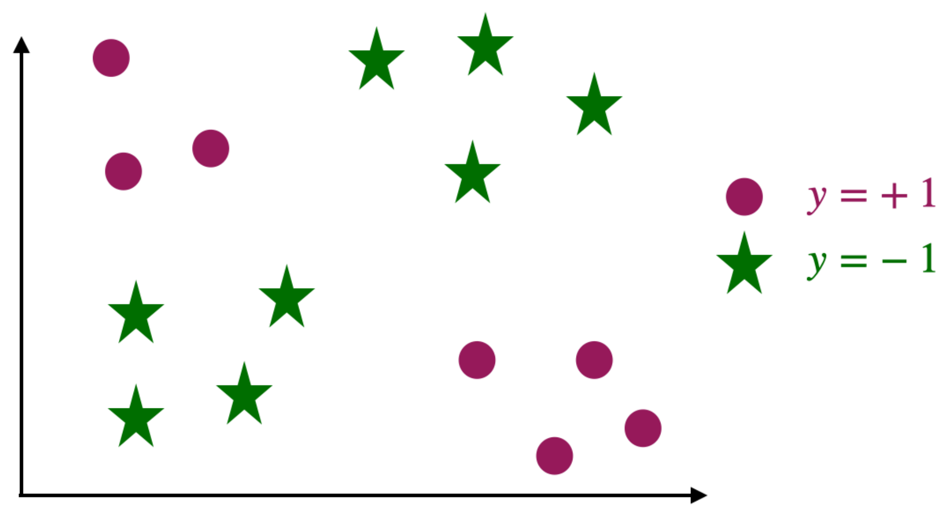
almost linearly separable



Kernel method (later)

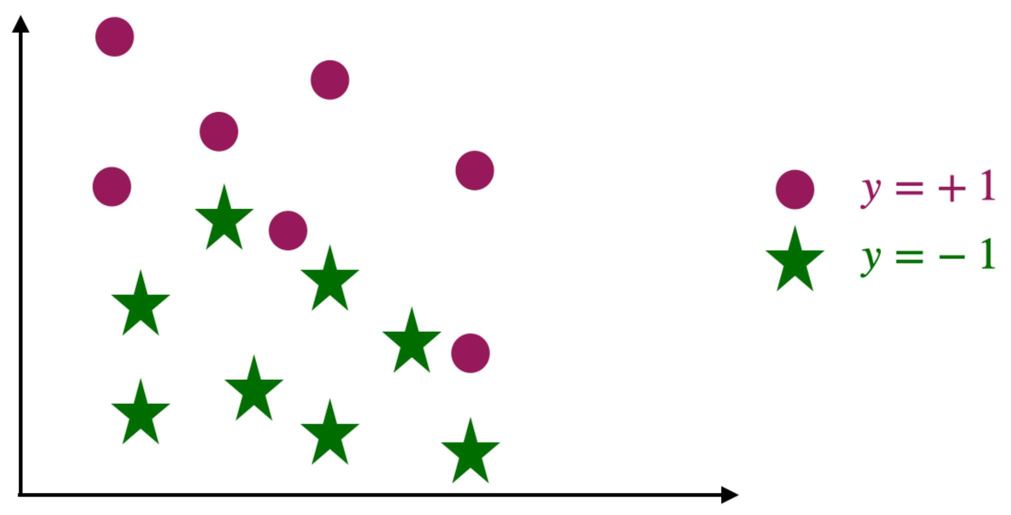
However, linear classification model can not classify non-linearly-separable dataset

XOR problem



Kernel method (later)

almost linearly separable



Perceptron Learning Algorithm and LSVM do not converge!

Probabilistic modeling to the rescue!

One correction to previous lecture on Probabilistic Modeling

Generative model learns $P(\boldsymbol{x}, y)$ from data,

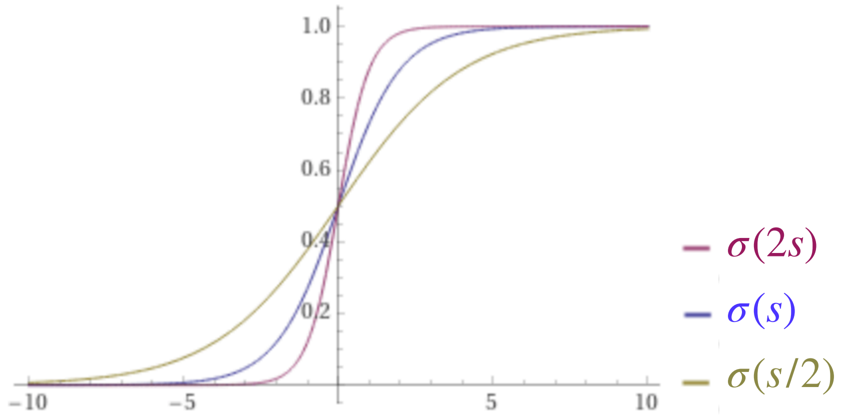
typically from $P(\boldsymbol{x} \mid y)$ and $P(y)$ separately.

Discriminative model learns $P(y \mid \boldsymbol{x})$ from data

Logistic Regression:
a useful ansatz for binary classification with “soft” linear decision boundary

$$P\left(y \mid \mathbf{x}, \mathbf{w}\right) = \frac{1}{1 + e^{-y\left(\mathbf{w}^T\mathbf{x}\right)}} \quad , \; y \in \{-1, +1\}$$

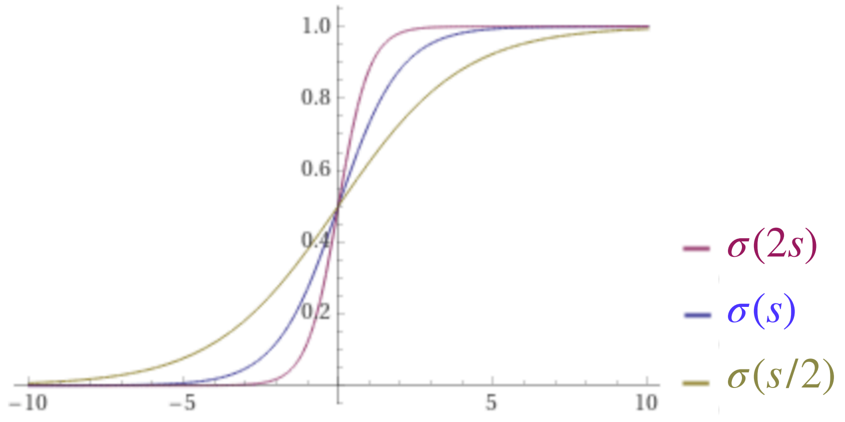
$$= \sigma[y(\mathbf{w}^T\mathbf{x})]$$ (sigmoid function)



Logistic Regression:
a useful ansatz for binary classification with “soft” linear decision boundary

$$P\left(y \mid \mathbf{x}, \mathbf{w}\right) = \frac{1}{1 + e^{-y\left(\mathbf{w}^T\mathbf{x}\right)}} \quad , \; y \in \{-1, +1\}$$

$$= \sigma[y(\mathbf{w}^T\mathbf{x})] \qquad \text{(sigmoid function)}$$



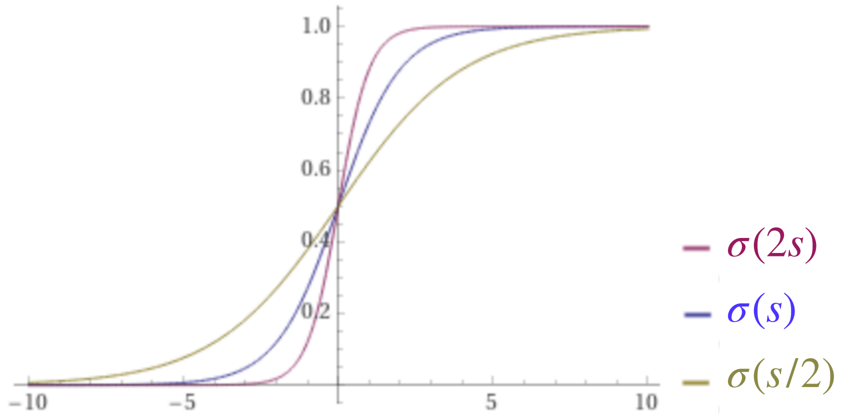
When $\mathbf{w}^T\mathbf{x} = 0$, the classification is maximally uncertain: $P(y = +1 \,|\, \mathbf{x}, \mathbf{w}) = P(y = -1 \,|\, \mathbf{x}, \mathbf{w}) = 1/2$.

This traces out the hyperplane that is the decision boundary, because above or below the hyperplane it’s more likely to be in one class than the other.

Logistic Regression:
a useful ansatz for binary classification with “soft” linear decision boundary

$$P\left(y \mid \mathbf{x}, \mathbf{w}\right) = \frac{1}{1 + e^{-y\left(\mathbf{w}^T\mathbf{x}\right)}} \quad , \; y \in \{-1, +1\}$$

$$= \sigma[y(\mathbf{w}^T\mathbf{x})] \qquad \text{(sigmoid function)}$$



When $\mathbf{w}^T\mathbf{x} = 0$, the classification is maximally uncertain: $P(y = +1 \,|\, \mathbf{x}, \mathbf{w}) = P(y = -1 \,|\, \mathbf{x}, \mathbf{w}) = 1/2$.

This traces out the hyperplane that is the decision boundary, because above or below the hyperplane it’s more likely to be in one class than the other.

To learn this discriminative model, given a training set $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, the most straightforward way is to perform MLE.

Namely, we want to optimise for $\mathbf{w}_{MLE}$ that maximises

$$\log\left(\prod_{i=1}^{n} P\left(y_i \mid \mathbf{x}_i, \mathbf{w}\right)\right) = -\sum_{i=1}^{n} \log\left(1 + e^{-y_i(\mathbf{w}^T\mathbf{x}_i)}\right)$$

If we define the negative log-likelihood as an empirical risk, then

$$\boxed{\ell(\mathbf{w}) \equiv \sum_{i=1}^{n} \log\left(1 + e^{-y_i(\mathbf{w}^T\mathbf{x}_i)}\right)}$$

And the ERM problem reduces to

$$\hat{\mathbf{w}}_{MLE} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \log\left(1 + e^{-y_i(\mathbf{w}^T\mathbf{x}_i)}\right)$$

If we define the negative log-likelihood as an empirical risk, then

$$\boxed{\ell(\mathbf{w}) \equiv \sum_{i=1}^{n} \log\left(1 + e^{-y_i(\mathbf{w}^T\mathbf{x}_i)}\right)}$$

And the ERM problem reduces to

$$\hat{\mathbf{w}}_{MLE} = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \log\left(1 + e^{-y_i(\mathbf{w}^T\mathbf{x}_i)}\right)$$

However, unlike in regularized linear model, taking the gradient of the loss and set to 0 to find critical points does not lead to an analytically tractable form of the optimal parameters.

In fact, the gradient is $\nabla_{\mathbf{w}}\ell(\mathbf{w}) = \sum_{i} y_i\mathbf{x}_i\left(1 - P\left(y_i \mid \mathbf{x}_i, \mathbf{w}\right)\right)$

As you'll show in homework 2, the above empirical risk $\ell(\mathbf{w})$ is the "cross-entropy error" and is a convex function of $\mathbf{w}$. As a result, hill-climbing or gradient-based optimization guarantees to find the global minima.

Iterative Method for Optimization (Learning): Gradient Descent

Simplest method for non-linear optimization based on gradient of the loss (cost) function:

Start at $\mathbf{w}_t$, then take a step along the direction of the steepest slope $\hat{\mathbf{v}}$

Iterative Method for Optimization (Learning): Gradient Descent

Simplest method for non-linear optimization based on gradient of the loss (cost) function:

Start at $\mathbf{w}_t$, then take a step along the direction of the steepest slope $\hat{\mathbf{v}}$

Choose the step size η (learning rate), then step: $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \hat{\mathbf{v}}$, where $\hat{\mathbf{v}}$ is a unit vector.

What is the direction of the steepest slope $\hat{\mathbf{v}}$?

Iterative Method for Optimization (Learning): Gradient Descent

Simplest method for non-linear optimization based on gradient of the loss (cost) function:

Start at $\mathbf{w}_t$, then take a step along the direction of the steepest slope $\hat{\mathbf{v}}$

Choose the step size η (learning rate), then step: $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta \hat{\mathbf{v}}$, where $\hat{\mathbf{v}}$ is a unit vector.

What is the direction of the steepest slope $\hat{\mathbf{v}}$?

$$\Delta \ell = \ell(\mathbf{w}_t + \eta \hat{\mathbf{v}}) - \ell(\mathbf{w}_t)$$

$$\approx \eta \nabla \ell(\mathbf{w}_t)^T \hat{\mathbf{v}} + O\left(\eta^2\right)$$

Iterative Method for Optimization (Learning): Gradient Descent

Simplest method for non-linear optimization based on gradient of the loss (cost) function:

Start at $\mathbf{w}_t$, then take a step along the direction of the steepest slope $\hat{\mathbf{v}}$

Choose the step size η (learning rate), then step: $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta\hat{\mathbf{v}}$, where $\hat{\mathbf{v}}$ is a unit vector.

What is the direction of the steepest slope $\hat{\mathbf{v}}$?

$$\Delta\ell = \ell(\mathbf{w}_t + \eta\hat{\mathbf{v}}) - \ell(\mathbf{w}_t)$$

$$\approx \eta\,\nabla\ell(\mathbf{w}_t)^T\hat{\mathbf{v}} + O\left(\eta^2\right)$$

Thus the steepest downhill direction is

$$\boxed{\hat{\mathbf{v}} = -\frac{\nabla\ell(\mathbf{w}_t)}{\|\nabla\ell(\mathbf{w}_t)\|}}$$

Iterative Method for Optimization (Learning): Gradient Descent

Simplest method for non-linear optimization based on gradient of the loss (cost) function:

Start at $\mathbf{w}_t$, then take a step along the direction of the steepest slope $\hat{\mathbf{v}}$

Choose the step size η (learning rate), then step: $\mathbf{w}_{t+1} = \mathbf{w}_t + \eta\hat{\mathbf{v}}$, where $\hat{\mathbf{v}}$ is a unit vector.

What is the direction of the steepest slope $\hat{\mathbf{v}}$?

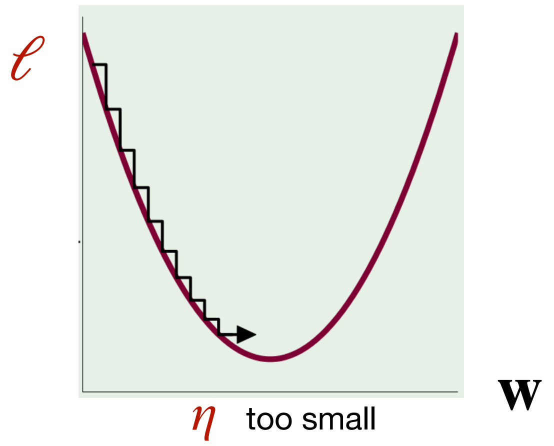
$$\Delta\ell = \ell(\mathbf{w}_t + \eta\hat{\mathbf{v}}) - \ell(\mathbf{w}_t)$$

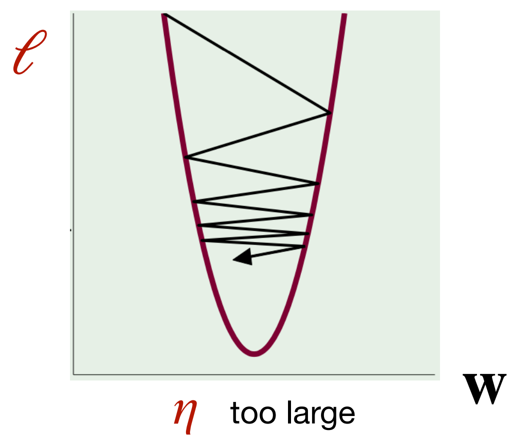
$$\approx \eta\nabla\ell(\mathbf{w}_t)^T\hat{\mathbf{v}} + O\left(\eta^2\right)$$

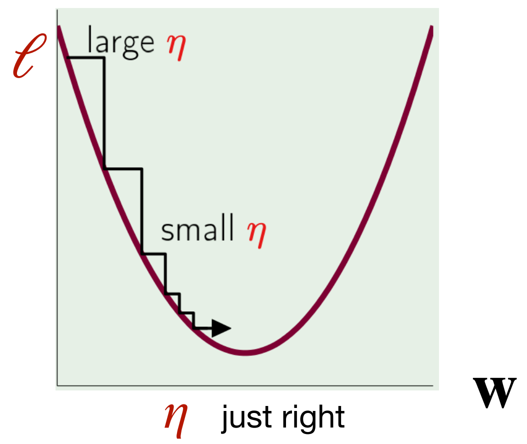
Thus the steepest downhill direction is

$$\boxed{\hat{\mathbf{v}} = -\frac{\nabla\ell(\mathbf{w}_t)}{\|\nabla\ell(\mathbf{w}_t)\|}}$$

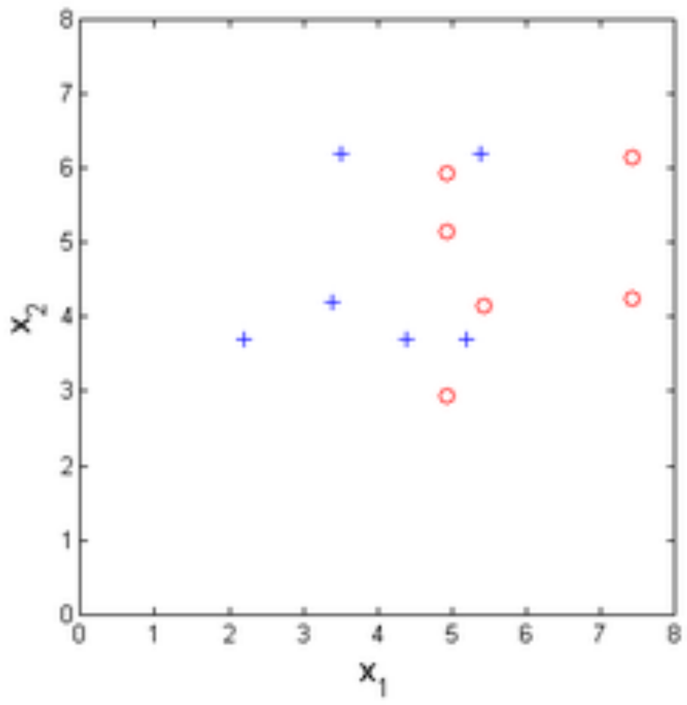
η should increase with slope.



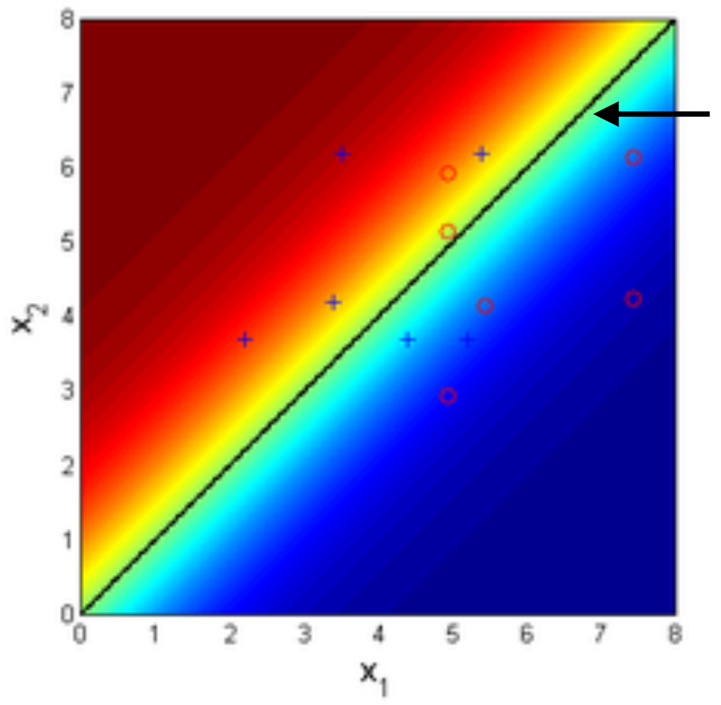




Example of Trained Logistic Binary Classifier (MLE)



$$P\left(y = +1 \mid \mathbf{x}, \hat{\boldsymbol{\theta}}_{MLE}\right)$$



$$\mathbf{w}_{MLE}^{T}\mathbf{x} + b_{MLE} = 0$$

We can also perform MAP estimate to learn the discriminative model

Suppose we assume a Gaussian Prior of the weight $w_j \sim \mathcal{N}(0, \lambda^2)$ so

$$P(\mathbf{w}) = \prod_j \frac{1}{\lambda\sqrt{2\pi}} \exp\left(\frac{-w_j^2}{2\lambda^2}\right)$$

Performing MAP estimate gives

$$\hat{\mathbf{w}}_{MAP} = \arg\max_{\mathbf{w}} \log P(\mathbf{w} \mid D, \lambda) = \arg\max_{\mathbf{w}} \left[\ell(\mathbf{w}) + \log P(\mathbf{w} \mid \lambda)\right]$$
$$= \arg\max_{\mathbf{w}} \left(\ell(\mathbf{w}) - \frac{1}{2\lambda^2}\mathbf{w}^T\mathbf{w}\right)$$

Here we can perform gradient descent again to find the global optima, since the objective function is the sum of two convex functions of $\mathbf{w}$.

We expect that the learned model will be less sensitive to sampling noise,
since the regularization term shall reduce the variance of the model.

Multinomial Logistic Regression (SoftMax Classification)

Generalising to the case when the outcome is not just binary, but has *K* classes, is simple.

We can model the conditional probability to be in class *k* for the linear discriminative model as

$$\boxed{P(Y = k \mid \mathbf{x}, \mathbf{w}) = \frac{\exp\left(\mathbf{w}_k^T \mathbf{x}\right)}{\sum_{k'=1}^{K} \exp\left(\mathbf{w}_{k'}^T \mathbf{x}\right)}, \quad \text{for} \quad k = 1, \ldots, K}$$

This function is also known as **SoftMax**.

We can train the model to find the empirically optimal hypothesis from MLE or MAP based on gradient descent.

In homework 2, you'll play around with the first useful application of machine learning! Handwriting digit classification!