Classification

Sewoong Oh

CSE446
University of Washington

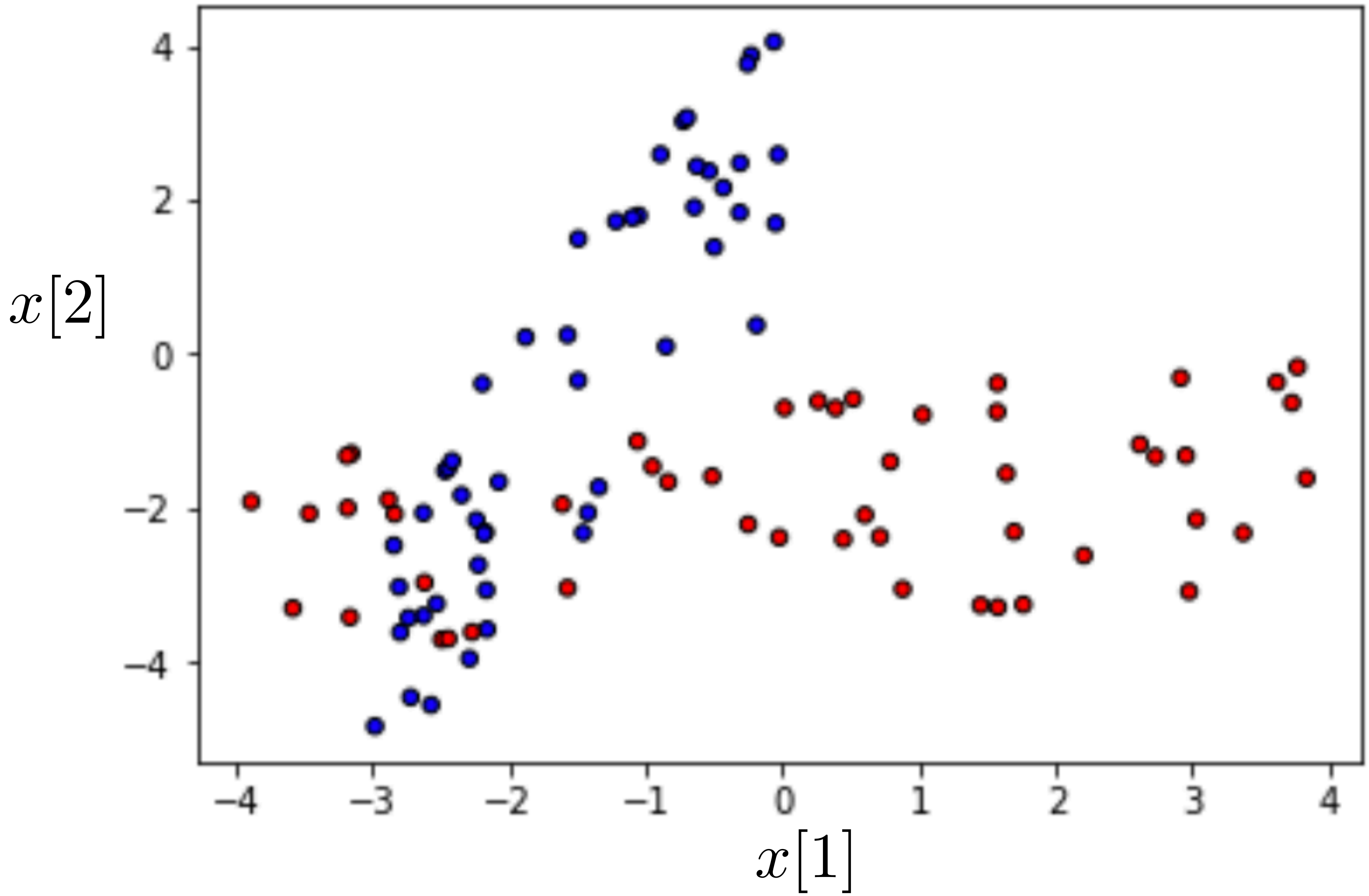
Boolean Classification

Boolean classification

- **Supervised learning** is training a predictor from labelled examples:
- There are two types of supervised learning
  - 1. **Regression**: the output variable ***y*** to be predicted is **real valued** scalar or a vector
  - 2. **Classification**: the output variable ***y*** to be predicted is categorical
    - 2.1 **Boolean classification**: there are two classes
    - 2.2 **Multi-class classification**: multiple classes

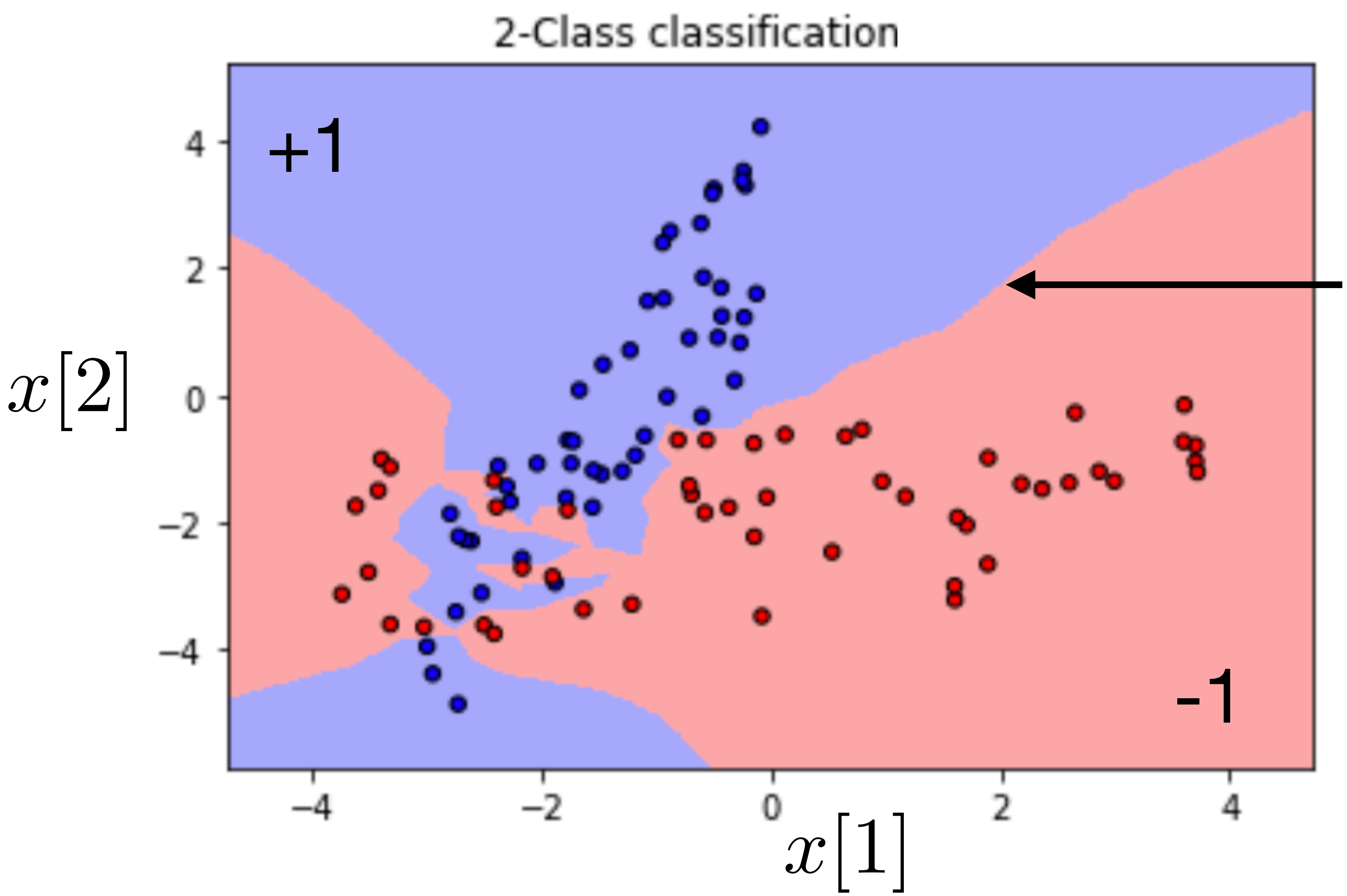
- We study Boolean classification in this chapter
- We denote two classes by -1 and 1, often corresponding to {FALSE,TRUE}
- for a data point (x_i, y_i), the value $y_i \in \{-1, 1\}$ is called the **class** or **label**
- A **Boolean classifier** predicts label ***y*** given input ***x***

Training data for a Boolean classification problem



- in this example, each input is $x_i \in \mathbb{R}^2$
- Red points have label y_i=-1, blue points have label y_i=1
- We want a predictor that maps any $x \in \mathbb{R}^2$ to a prediction $\hat{y} \in \{-1, +1\}$

Example: nearest neighbor classifier trained on 100 samples



when overfitting happens, we learned that prediction $f(x)$ is sensitive to changes in x, and this results in complicated **decision boundaries**

- 1-nearest neighbor classifier:
  - given x, let $\hat{i} \in \{1,\ldots,n\}$ be the closest training sample, i.e.
  $$\hat{i} = \arg \min_{i \in \{1,\ldots,n\}} \|x - x_i\|_2^2$$
  - prediction is the label of the nearest neighbor: $f(x) = y_{\hat{i}}$
- **Red** region is the set of x for which prediction is -1
- **Blue** region is the set of x for which prediction is +1
- zero training error (all training data correctly classified), but likely to be overfitting

Empirical risk minimization (ERM) with quadratic loss

- expanding on what we know from linear regression (in particular linear least squares regression), a straightforward approach for classification is the following
  - use a linear model:

$$\hat{y} \;=\; f_w(x) = w_0 + w_1 x[1] + w_2 x[2] + \cdots$$

  - train on Empirical Risk Minimization with L2 loss

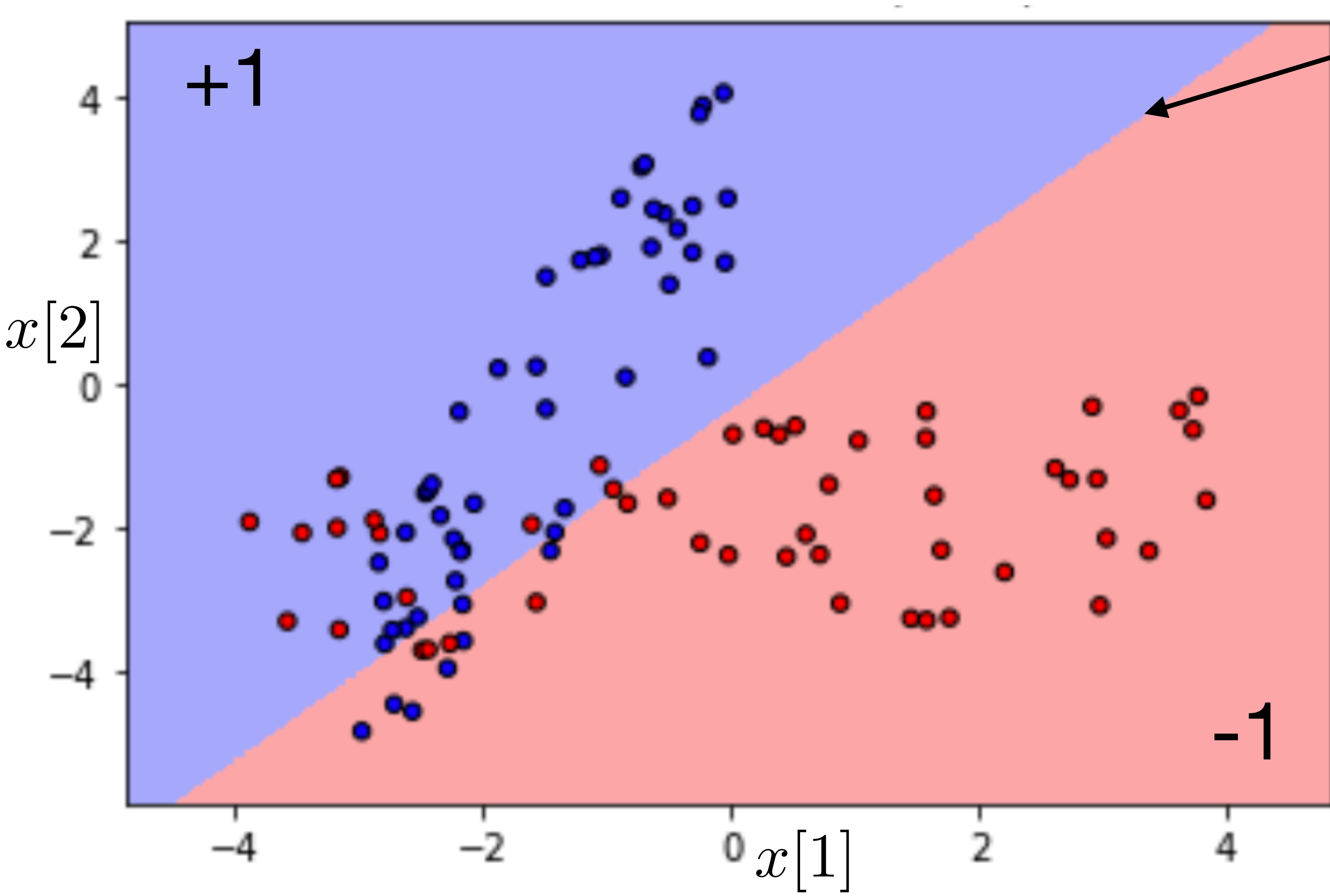
$$\mathcal{L}(w) \;=\; \sum_{i=1}^{n} (\underbrace{w^T x_i}_{\hat{y}_i} - y_i)^2$$

- Note that this is exactly linear least squares regression, just applied to a discrete valued y_i's
- to make a **hard prediction** in $\{-1,1\}$,

$$\begin{aligned}\hat{v} \;&=\; \text{sign}(\, f_w(x)\,) \\ &= \text{sign}(\, w_0 + w_1 x[1] + \cdots\,)\end{aligned}$$

- general recipe:
  - train linear model on ERM
  - make **hard prediction** by taking the $\text{sign}(\,\cdot\,)$
- **significantly better to choose the right loss tailored for discrete y_i's**

Example: linear classifier trained on 100 samples



simple decision boundary at $w^T x = 0$

- linear model: $\hat{y} = f(x) = w_0 + w_1 x[1] + w_2 x[2]$
- predict using $\hat{v} = \text{sign}(\hat{y}) = \text{sign}(w^T x)$
- 20% mis-classified in training data
- true positive C_{tp} =42, false positive C_{fp} =12,
- true negative C_{tn} =38, false negative C_{fn} =8

Empirical risk minimization

- given a choice of a loss function $\ell(\hat{y}, y)$, the empirical risk is

$$\mathcal{L}(w) \;=\; \frac{1}{n}\sum_{i=1}^{n} \ell(\,\hat{y}_i, y_i\,)$$

- using a linear model:

$$\hat{y} \;=\; f_w(x) = w_0 + w_1 x[1] + w_2 x[2] + \cdots$$

  the empirical risk is now

$$\mathcal{L}(w) \;=\; \frac{1}{n}\sum_{i=1}^{n} \ell(\,w^T x_i, y_i\,)$$

- to make a **hard** prediction in $\{-1,1\}$,

$$\begin{aligned}\hat{v} \;&=\; \operatorname{sign}(\,f_w(x)\,)\\ &= \operatorname{sign}(\,w_0 + w_1 x[1] + \cdots\,)\end{aligned}$$

- ERM minimizes this empirical risk
- Regularized ERM minimizes $\mathcal{L}(w) + \lambda\, r(w)$

Loss function for Boolean classification

- We need to design loss function $\ell(\hat{y}, y_i)$
- Note that
  - $\hat{y} = f_w(x) = w^T x \quad \in \quad \mathbb{R}$ can take **any real value**
  - But $y_i's$ only take values in $\{-1, +1\}$
- so in order to specify $\ell(\hat{y}, y_i)$
  we only need to give two functions (of scalar $\hat{y}$)
  - $\ell(\hat{y}, -1)$ is how much $\hat{y}$ irritates us when $y = -1$
  - $\ell(\hat{y}, +1)$ is how much $\hat{y}$ irritates us when $y = +1$

- a natural choice of the empirical risk is
  the **average number of mis-classified samples** in the training data
- where $\ell(\hat{y}, y_i)$ is the 0-1 loss:

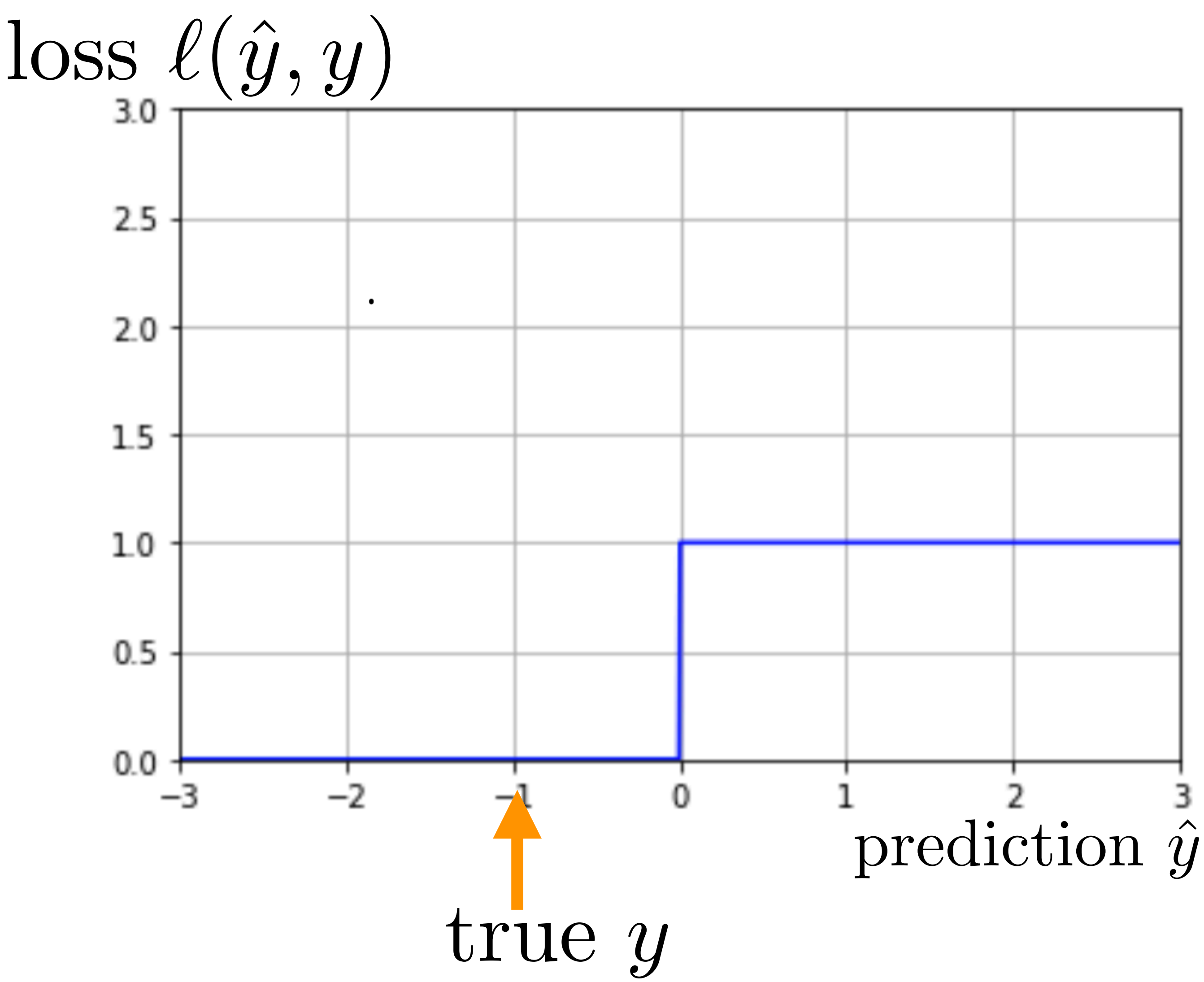
$$\ell(\hat{y}, y) = \begin{cases} 0 & \text{if sign}(\hat{y}) = y \\ +1 & \text{otherwise} \end{cases}$$

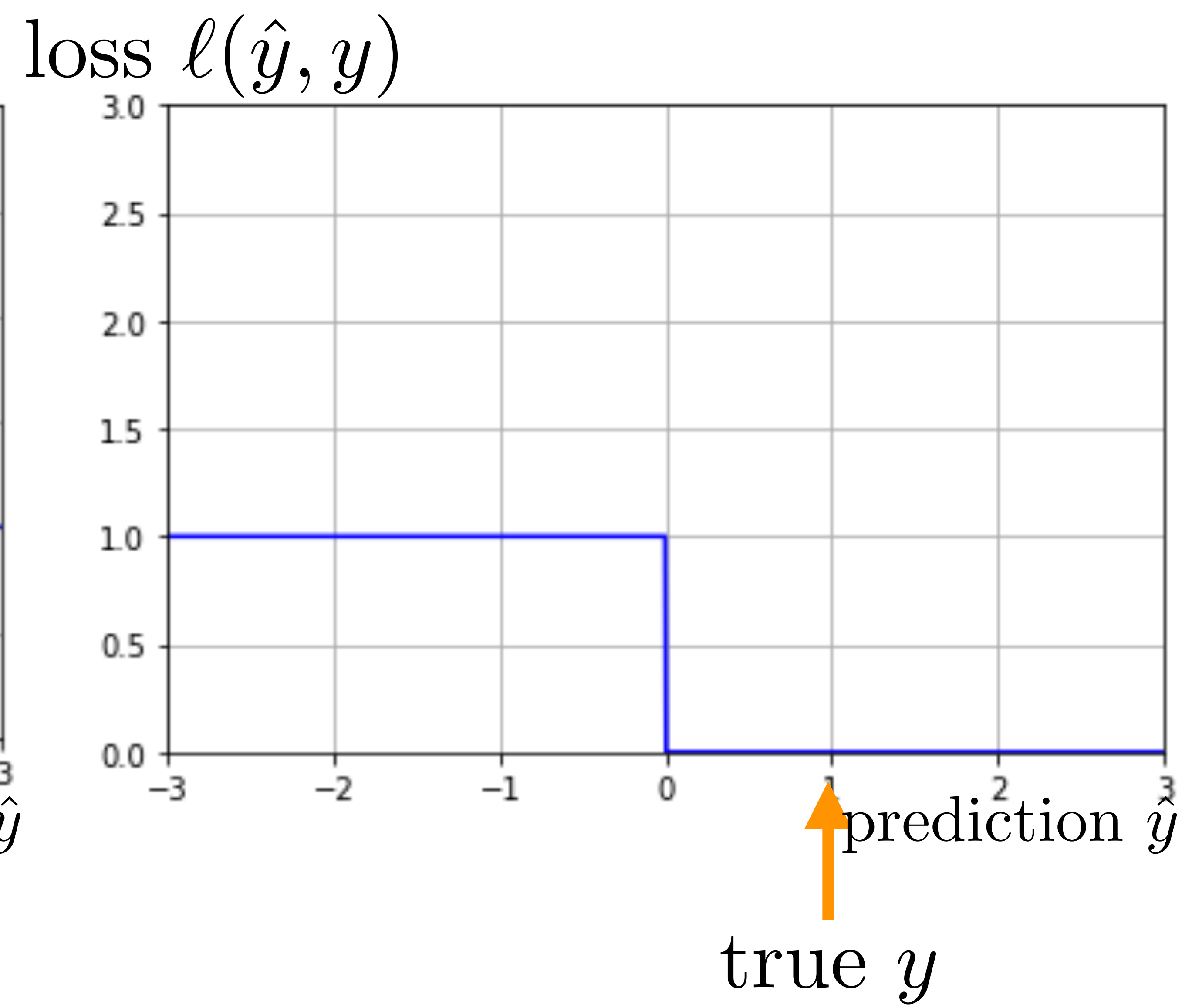
$$\mathcal{L}(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(\hat{y}_i, y_i)$$

0-1 loss

- 0-1 loss is

$$\ell(\hat{y}, -1) = \begin{cases} 0 & \hat{y} < 0 \\ +1 & \hat{y} \geq 0 \end{cases} \qquad \ell(\hat{y}, +1) = \begin{cases} 0 & \hat{y} > 0 \\ +1 & \hat{y} \leq 0 \end{cases}$$





Problem with 0-1 loss

- 0-1 loss is not differentiable, or even continuous (and certainly not convex)

- its gradient is zero or does not exist

- Gradient based optimizer does not know how to improve the model

Ideas of proxy loss

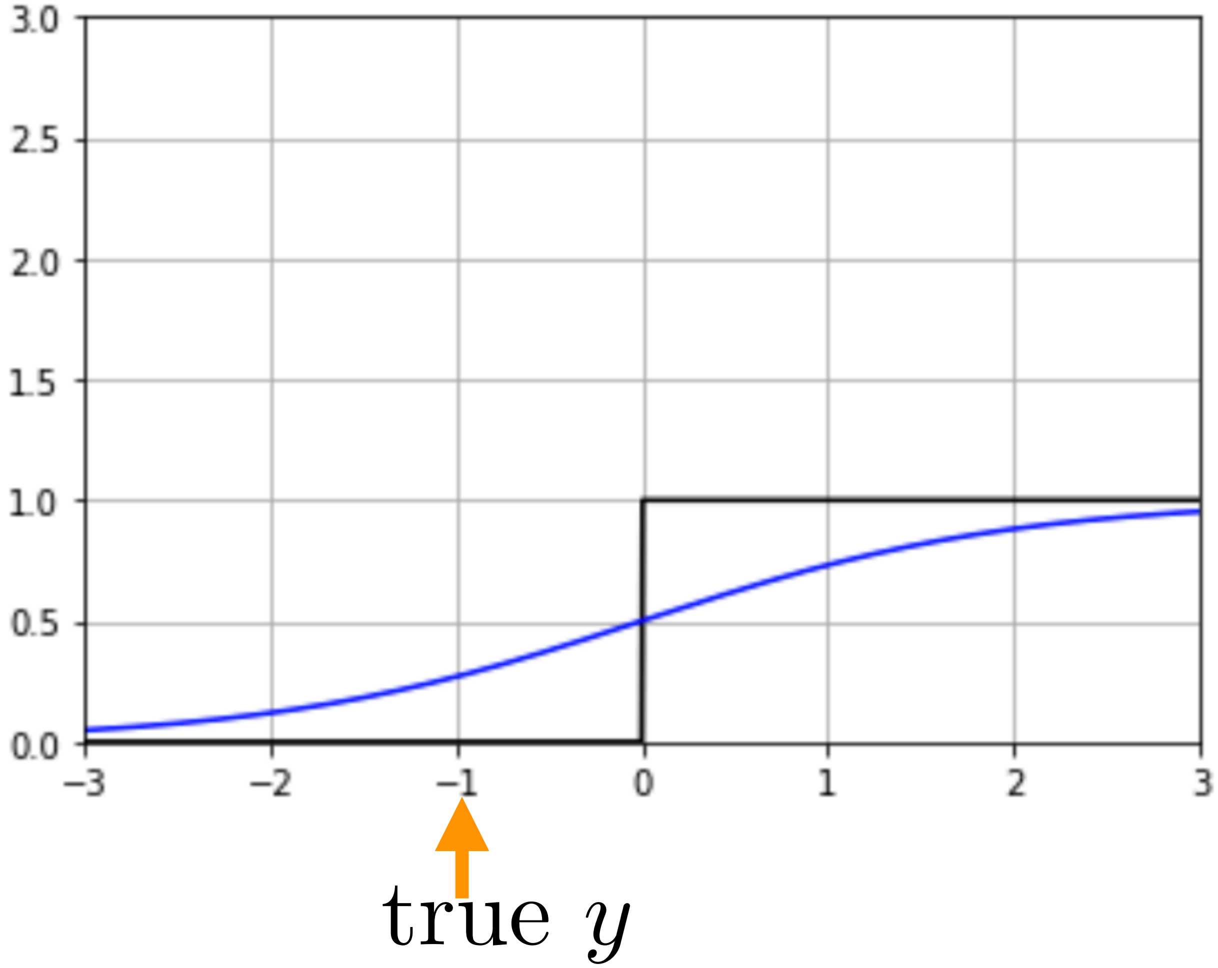
- we get better results using proxy losses that
  - approximate, or captures the flavor of, the 0-1 loss
  - is more easily optimized (e.g. convex and/or non-zero derivatives)

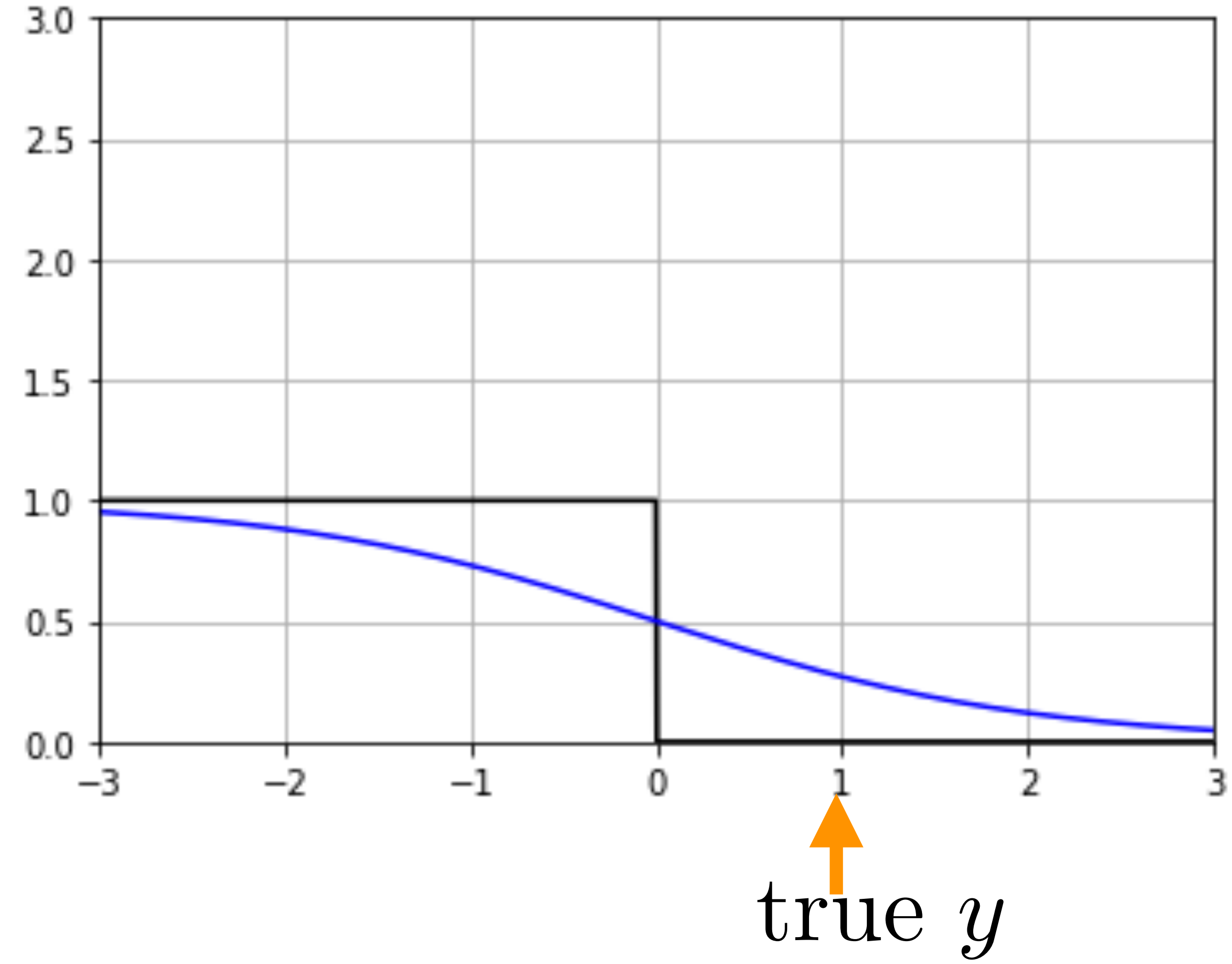
- concretely, we want **proxy loss function**
  - with $\ell(\hat{y}, -1)$ small when $\hat{y} < 0$ and larger when $\hat{y} > 0$
  - with $\ell(\hat{y}, 1)$ small when $\hat{y} > 0$ and larger when $\hat{y} < 0$
  - Which has other nice characteristics, e.g., differentiable or convex

Sigmoid loss (also known as logistic function)

$$\ell(\hat{y}, -1) = \frac{1}{1+e^{-\hat{y}}}$$

$$\ell(\hat{y}, +1) = \frac{1}{1+e^{\hat{y}}}$$





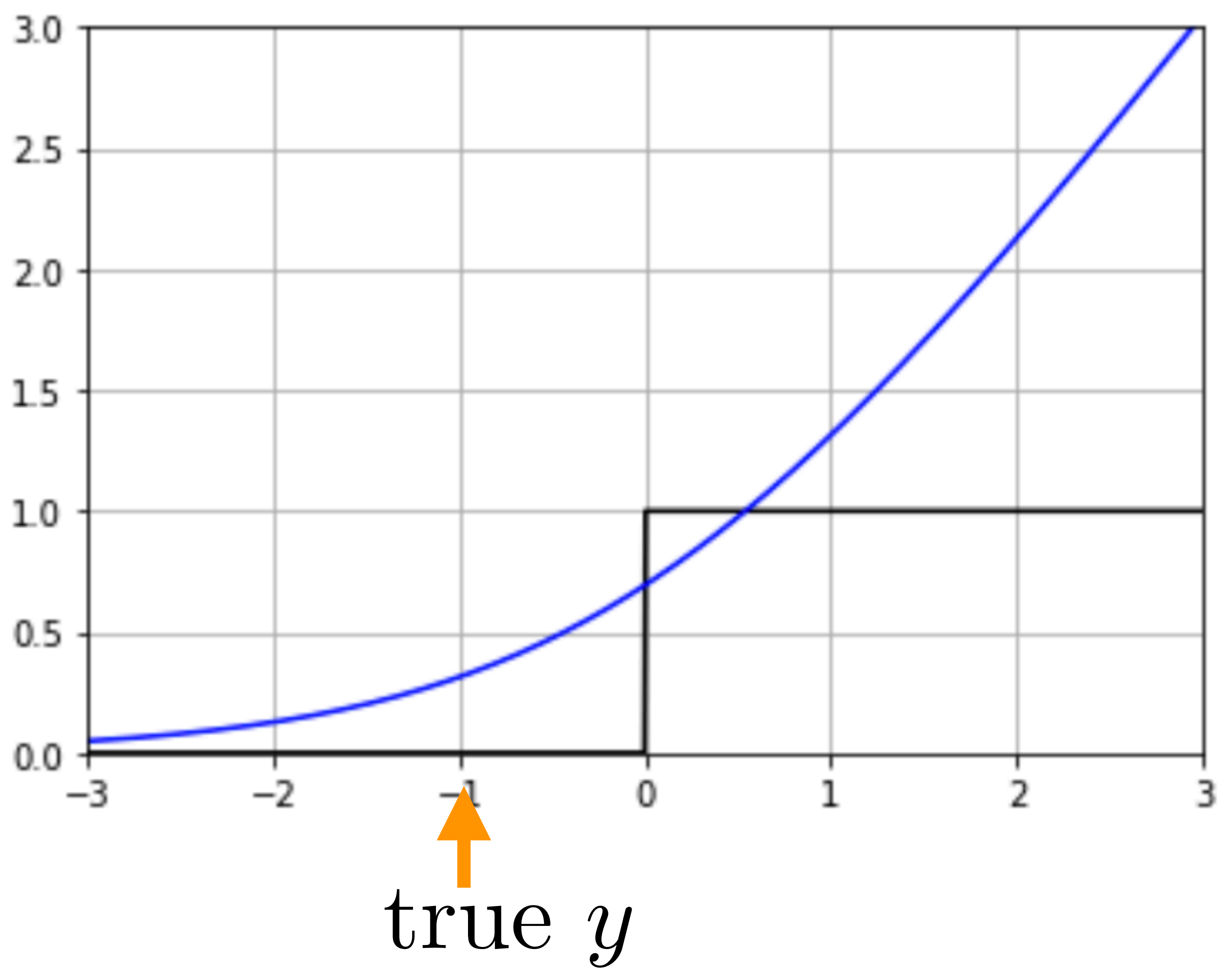
- differentiable approximation of 0-1 loss
- but not convex in $\hat{y}$
- the two losses sum to one

$$\frac{1}{1+e^{-\hat{y}}} + \frac{1}{1+e^{\hat{y}}} = \frac{e^{\hat{y}}}{e^{\hat{y}}+1} + \frac{1}{1+e^{\hat{y}}} = 1$$

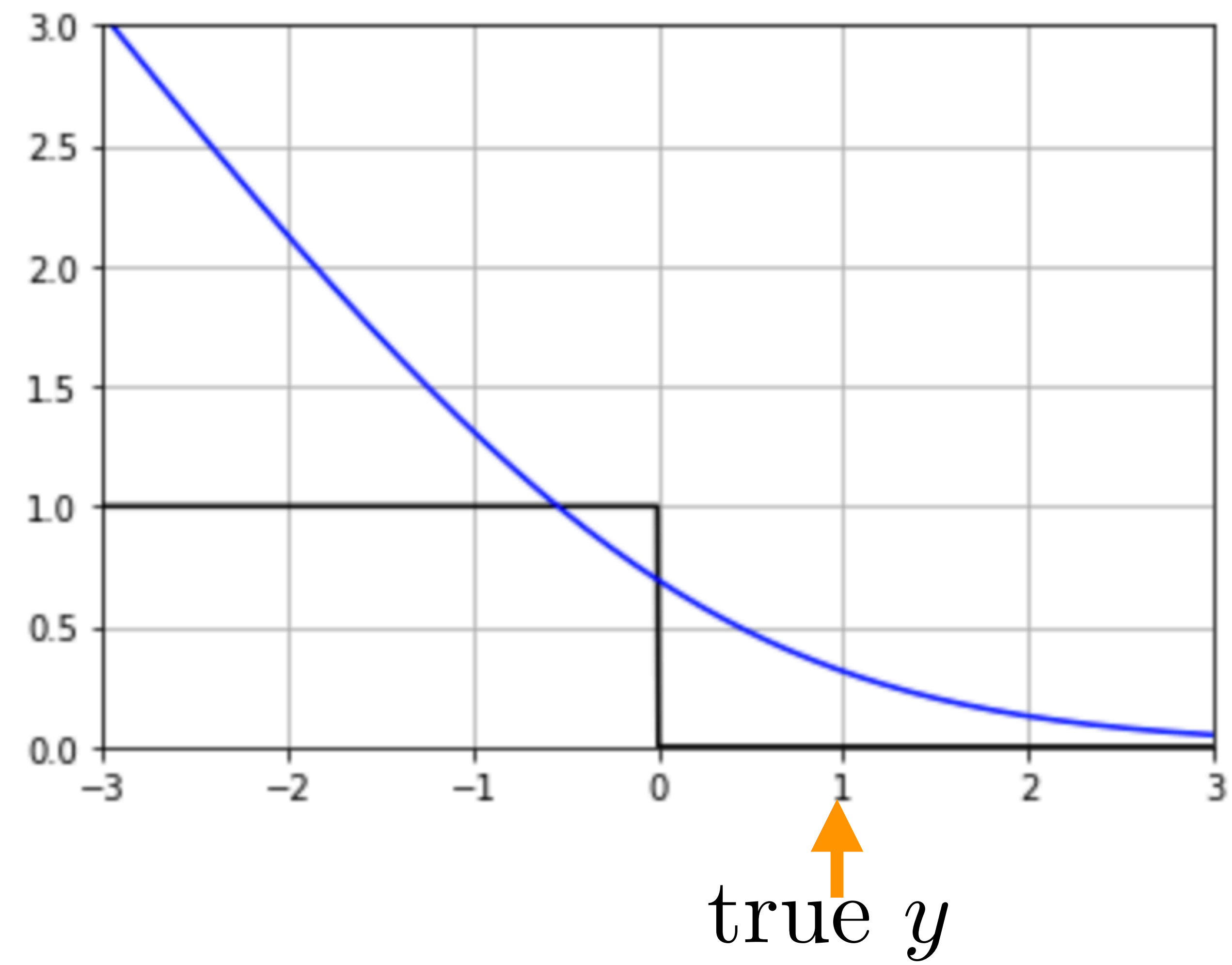
- softer (or smoothed) version of the 0-1 loss

Logistic loss

$$\ell(\hat{y}, -1) = \log(1 + e^{\hat{y}})$$



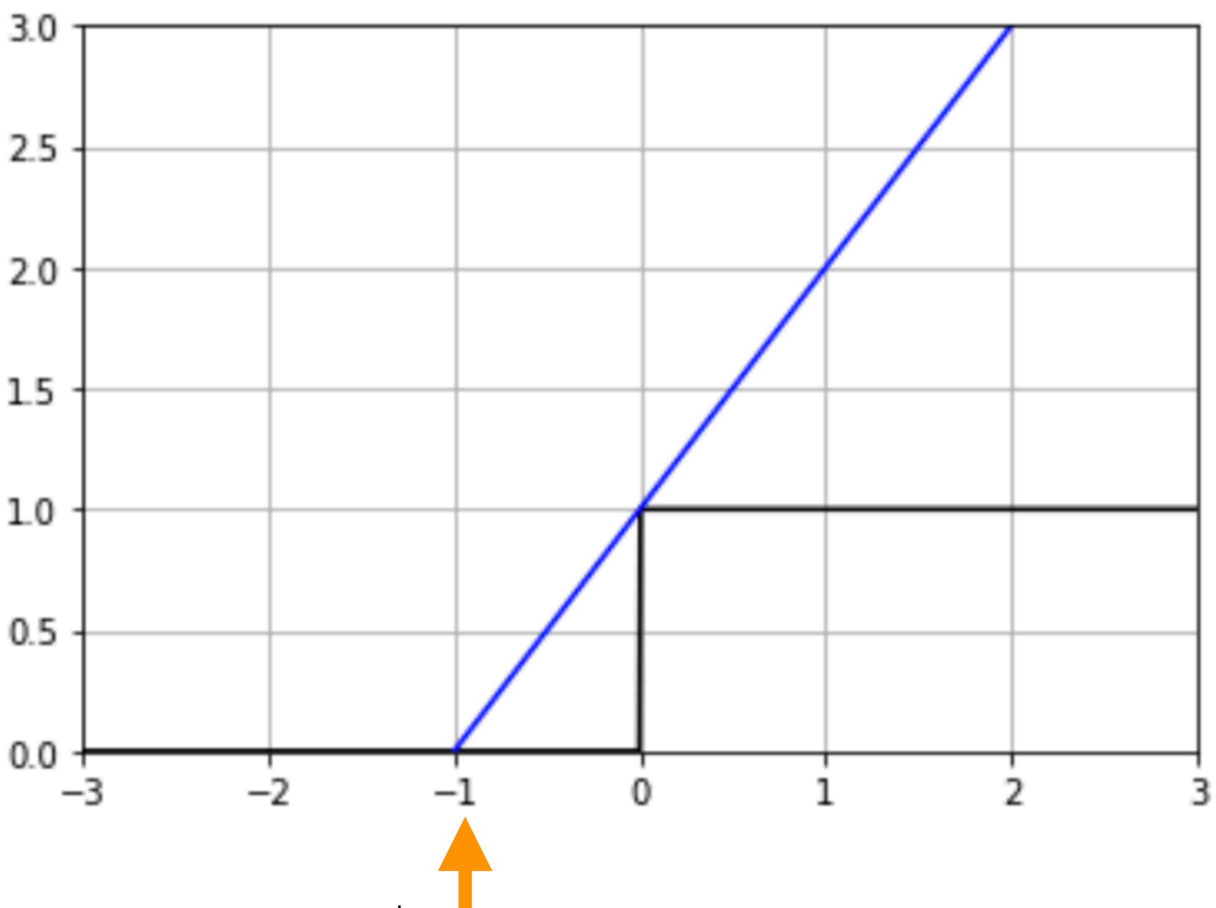
$$\ell(\hat{y}, +1) = \log(1 + e^{-\hat{y}})$$



- differentiable and convex in $\hat{y}$
- approximation of 0-1
- don’t get confused between **logistic loss** (which is the function above) and **logistic function** (which is the sigmoid loss)

Hinge loss

$$\ell(\hat{y}, -1) = [1 + \hat{y}]^+ \qquad \ell(\hat{y}, +1) = [1 - \hat{y}]^+$$



true y

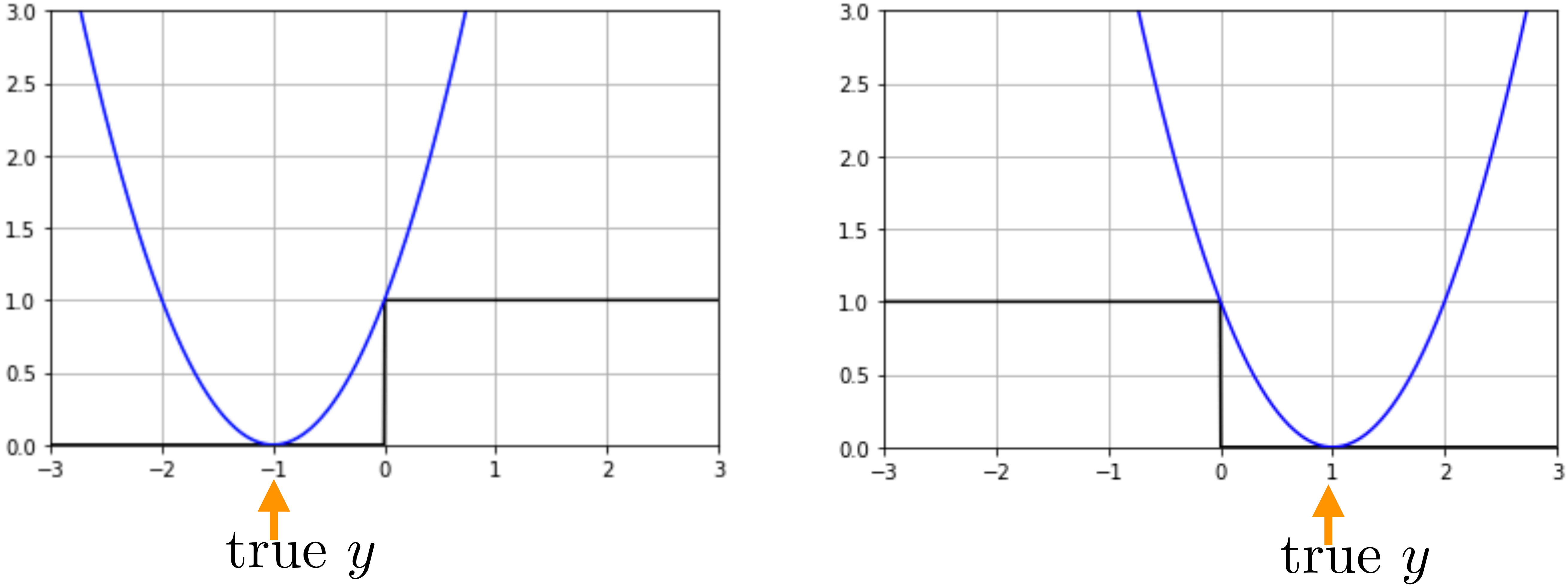
true y

where $[x]^+ = \max\{0, x\}$

- non-differentiable but convex approximation of 0-1 loss

Square loss

$$\ell(\hat{y}, -1) \;=\; (\hat{y}+1)^2 \qquad\qquad \ell(\hat{y}, +1) \;=\; (\hat{y}-1)^2$$



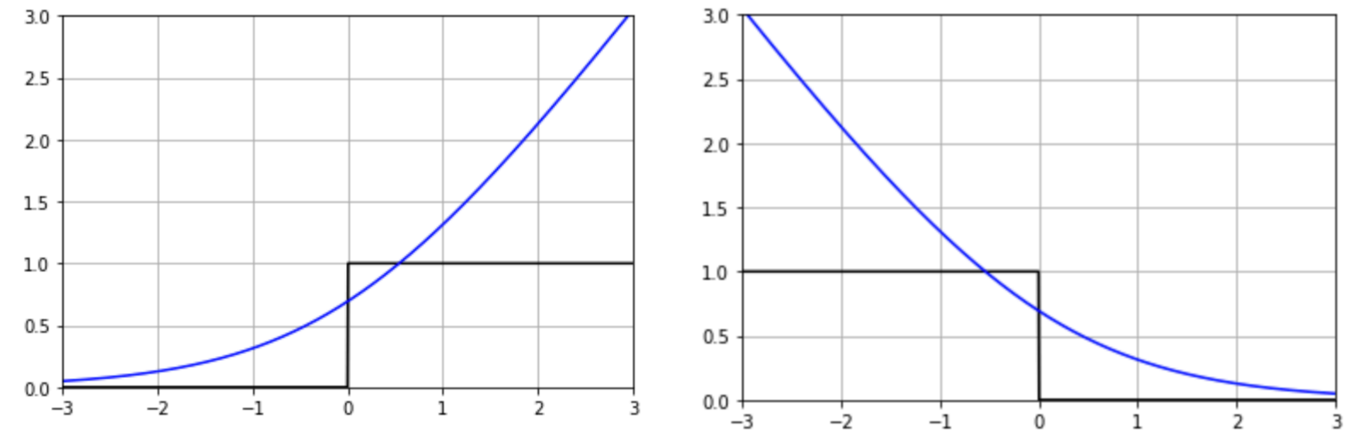
- not only is it convex, square loss is easy to minimize (has a closed form solution)

Logistic regression:

it is called regression but is just classification with logistic loss

Logistic regression

- uses **logistic loss**

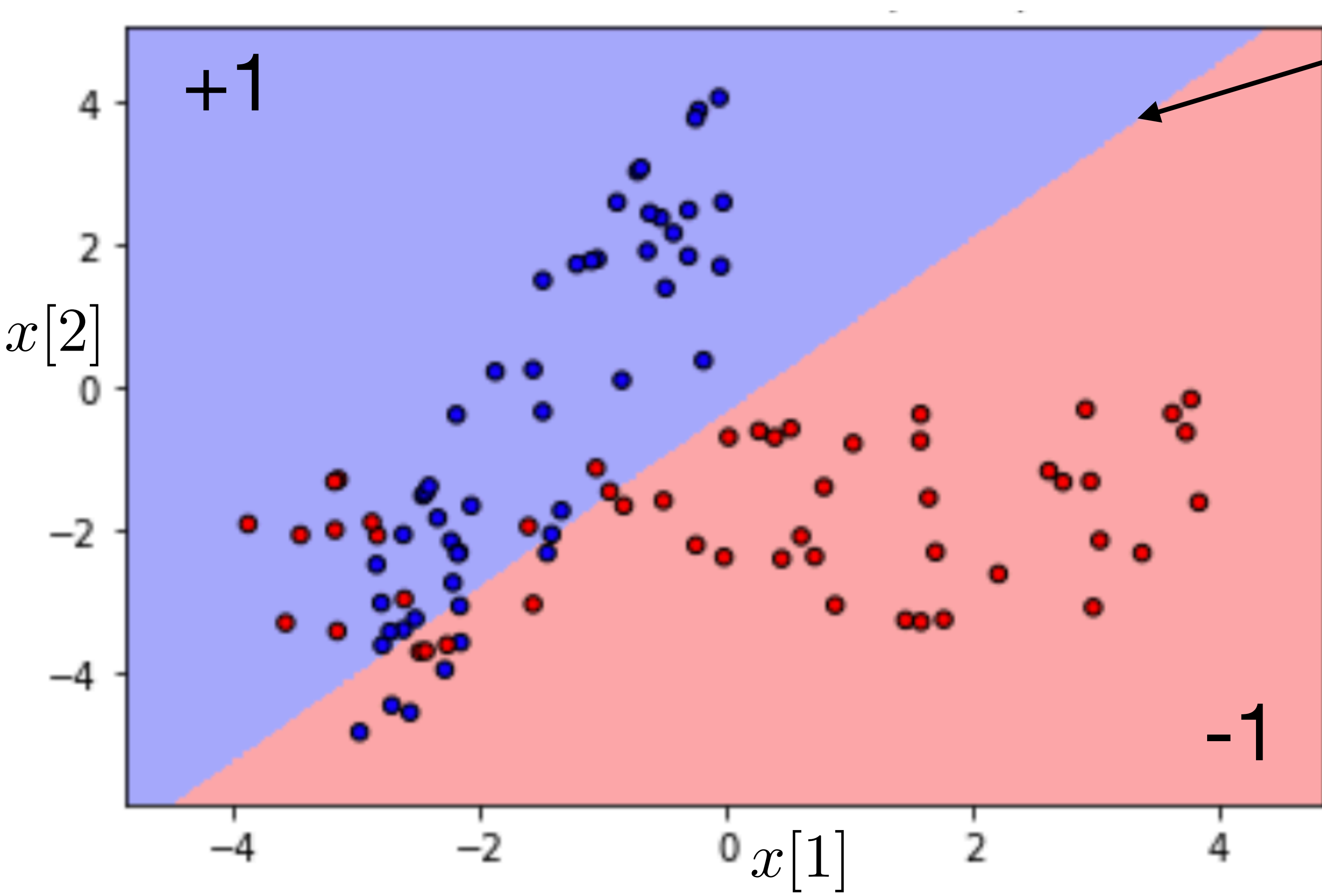


$$\hat{w}_{\text{logistic}} = \arg\min_{w} \mathscr{L}(w) = \frac{1}{n}\Big(\sum_{i:y_i=-1} \log(1+e^{w^T x_i}) + \sum_{i:y_i=+1} \log(1+e^{-w^T x_i}) \Big)$$

with a choice of a regularizer $r(w)$

- can minimize $\mathscr{L}(w) + \lambda r(w)$
- is a convex optimization if the regularizer is convex, and the minimizer can be found efficiently
- this follows from the fact that $f(z) = \log(1+e^{z})$ is convex in $z \in \mathbb{R}$ (and $f(z) = \log(1+e^{-z})$ is also a convex function in $z \in \mathbb{R}$)

Example: linear classifier trained on 100 samples



simple decision boundary at $w^T x = 0$

- linear model: $\hat{y} = f(x) = w_0 + w_1 x[1] + w_2 x[2]$
- predict using $\hat{v} = \text{sign}(\hat{y})$
- 20% mis-classified in training data
- true positive C_{tp} =42, false positive C_{fp} =12,
- true negative C_{tn} =38, false negative C_{fn} =8

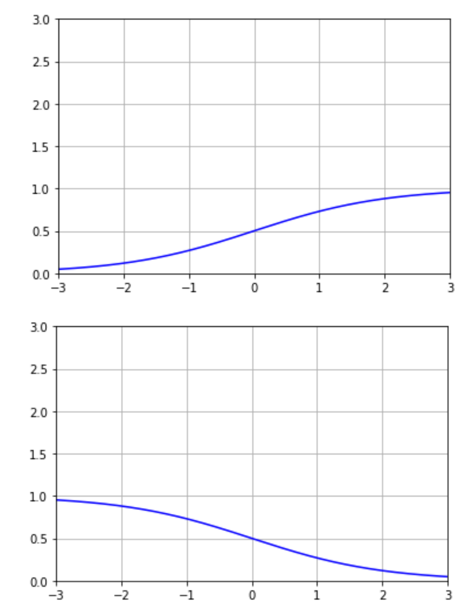
Probabilistic interpretation of logistic regression

- just as Maximum Likelihood Estimator (MLE) under linear model and additive Gaussian noise model recovers **linear least squares**,
- we study a particular noise model that recovers **logistic regression**

- a probabilistic noise model for Boolean labels:

$$\mathbb{P}(y_i = +1 \mid x_i) = \frac{1}{1 + e^{-w^T x_i}}$$

$$\mathbb{P}(y_i = -1 \mid x_i) = \frac{1}{1 + e^{w^T x_i}}$$



$w^T x_i$

with a ground truth model parameter $w \in \mathbb{R}^d$

- this function $\sigma(z) = \dfrac{1}{1 + e^{-z}}$ is called a **logistic function** (not to be confused with logistic loss, which is different) or a **sigmoid function**
- if we know that the data came from such a model, but do not know the ground truth parameter $w \in \mathbb{R}^d$, we can apply MLE to find the best w
- this MLE recovers the logistic regression algorithm, exactly

Maximum Likelihood Estimator (MLE)

- if the data came from a probabilistic model model:

$$(\underbrace{\frac{1}{1+e^{-w^T x}}}_{\mathbb{P}(y_i=+1|x_i)}, \underbrace{\frac{1}{1+e^{w^T x}}}_{\mathbb{P}(y_i=-1|x_i)})$$

- log-likelihood of observing a data point (x_i, y_i) is

$$\text{log-likelihood} = \log\Big(\mathbb{P}(y_i|x_i)\Big) = \begin{cases} \log\left(\frac{1}{1+e^{-w^T x_i}}\right) & \text{if } y_i = +1 \\ \log\left(\frac{1}{1+e^{w^T x_i}}\right) & \text{if } y_i = -1 \end{cases}$$

- Maximum Likelihood Estimator is the one that maximizes the sum of all log-likelihoods on training data points

$$\hat{w}_{\text{MLE}} = \arg\max_{w} \ \mathbb{P}(\{y_1, \ldots, y_n\} \,|\, \{x_1, \ldots, x_n\})$$

$$= \arg\max_{w} \prod_{i=1} \mathbb{P}(y_i \,|\, x_i)$$

(independence)

$$= \arg\max_{w} \sum_{i:y_i=-1} \log\left(\frac{1}{1+e^{w^T x_i}}\right) + \sum_{i:y_i=1} \log\left(\frac{1}{1+e^{-w^T x_i}}\right)$$

(substitution)

- notice that this is exactly the **logistic regression:**

$$\hat{w}_{\text{logistic}} = \arg\min_{w} \frac{1}{n}\Big(\sum_{i:y_i=-1} \log(1+e^{w^T x_i}) + \sum_{i:y_i=1} \log(1+e^{-w^T x_i}) \Big)$$

- once we have trained a model $\hat{w}_{\text{logistic}}$, we can make a hard prediction $\hat{v}$ of the label at an input example x

$$\hat{v} = \begin{cases} +1 & \text{if } \mathbb{P}(+1|x) \geq \mathbb{P}(-1|x) \\ -1 & \text{otherwise} \end{cases}$$

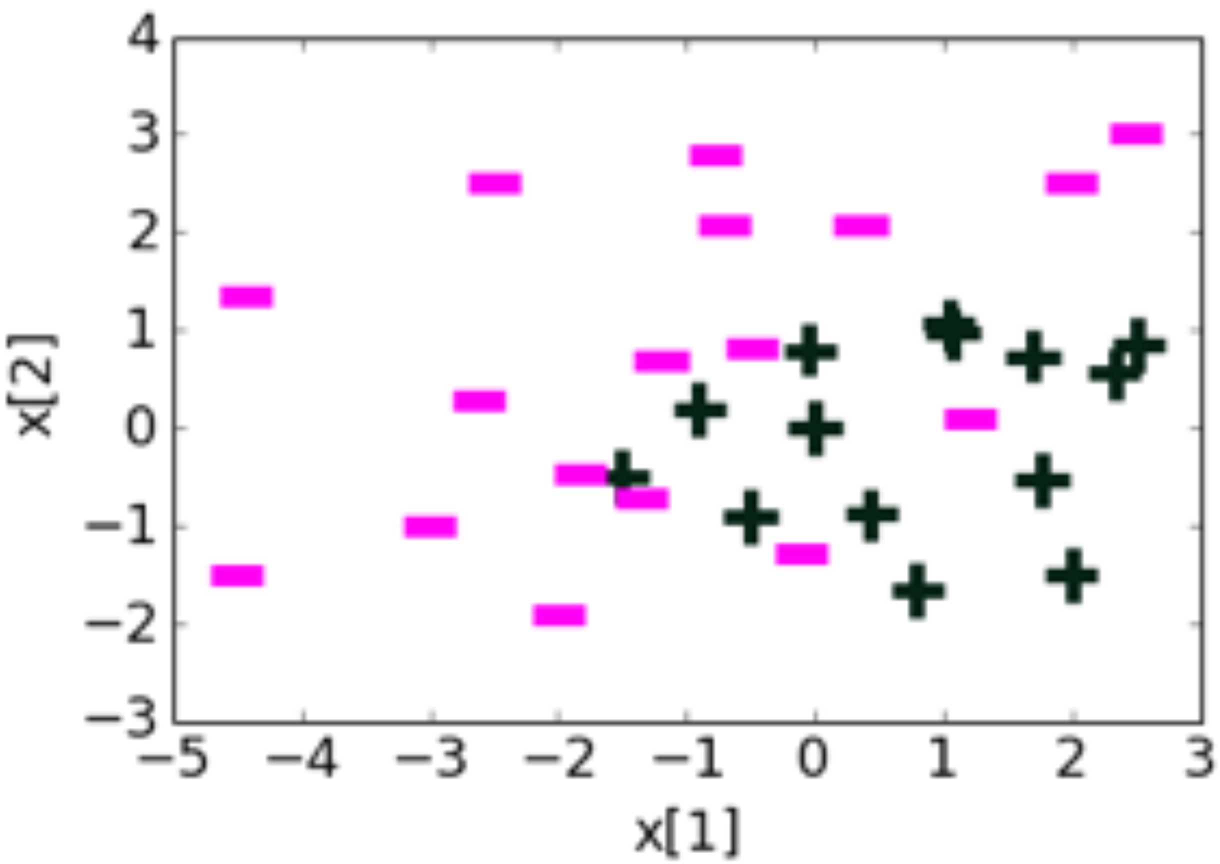
$$= \begin{cases} +1 & \text{if } \frac{1}{1+e^{-w^T x}} \geq \frac{1}{1+e^{w^T x}} \\ -1 & \text{otherwise} \end{cases}$$

$$= \begin{cases} +1 & \text{if } 1 \leq e^{2w^T x} \\ -1 & \text{otherwise} \end{cases}$$

$$= \text{sign}(w^T x)$$

Overfitting in classification

Example: adding more polynomial features

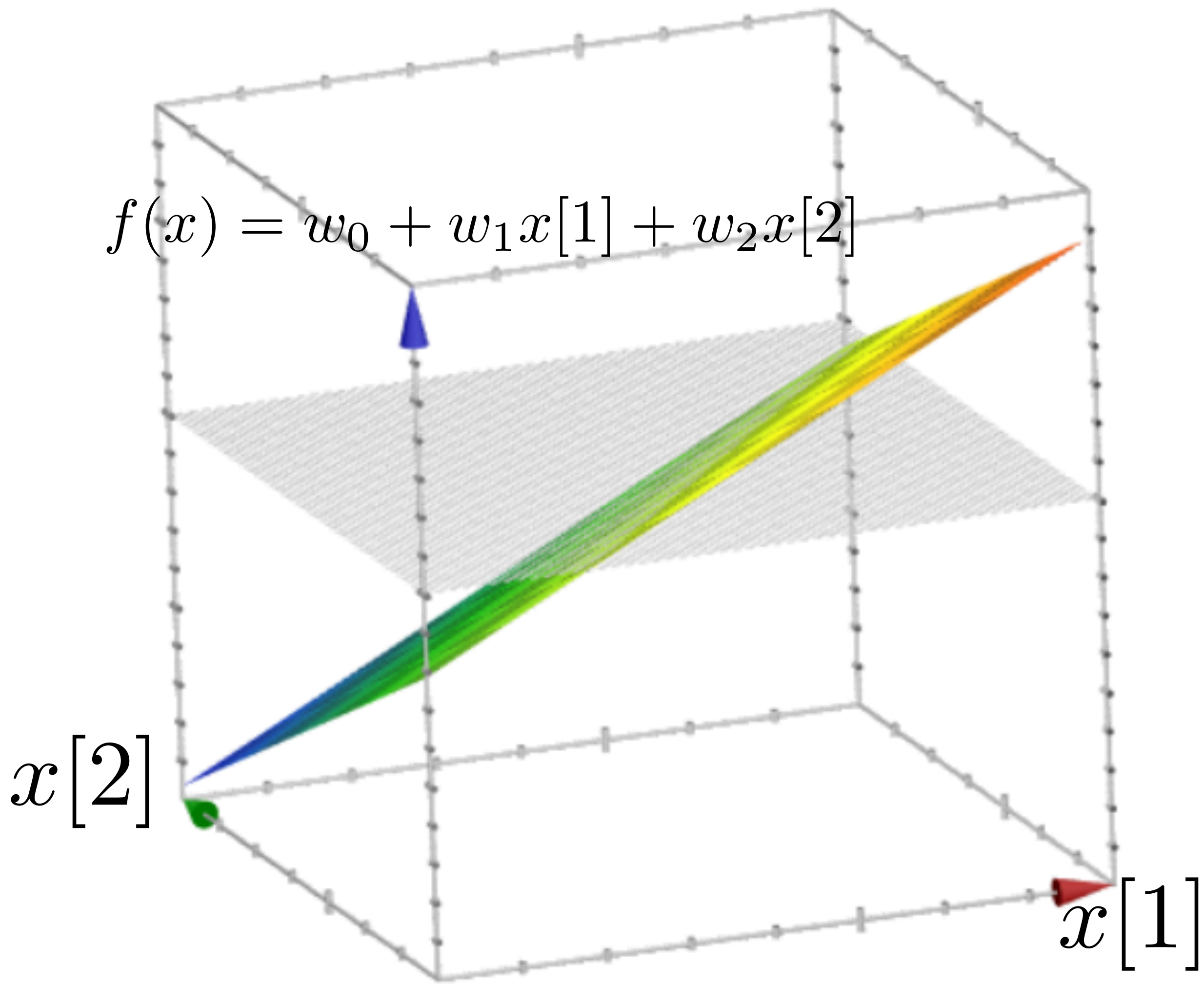


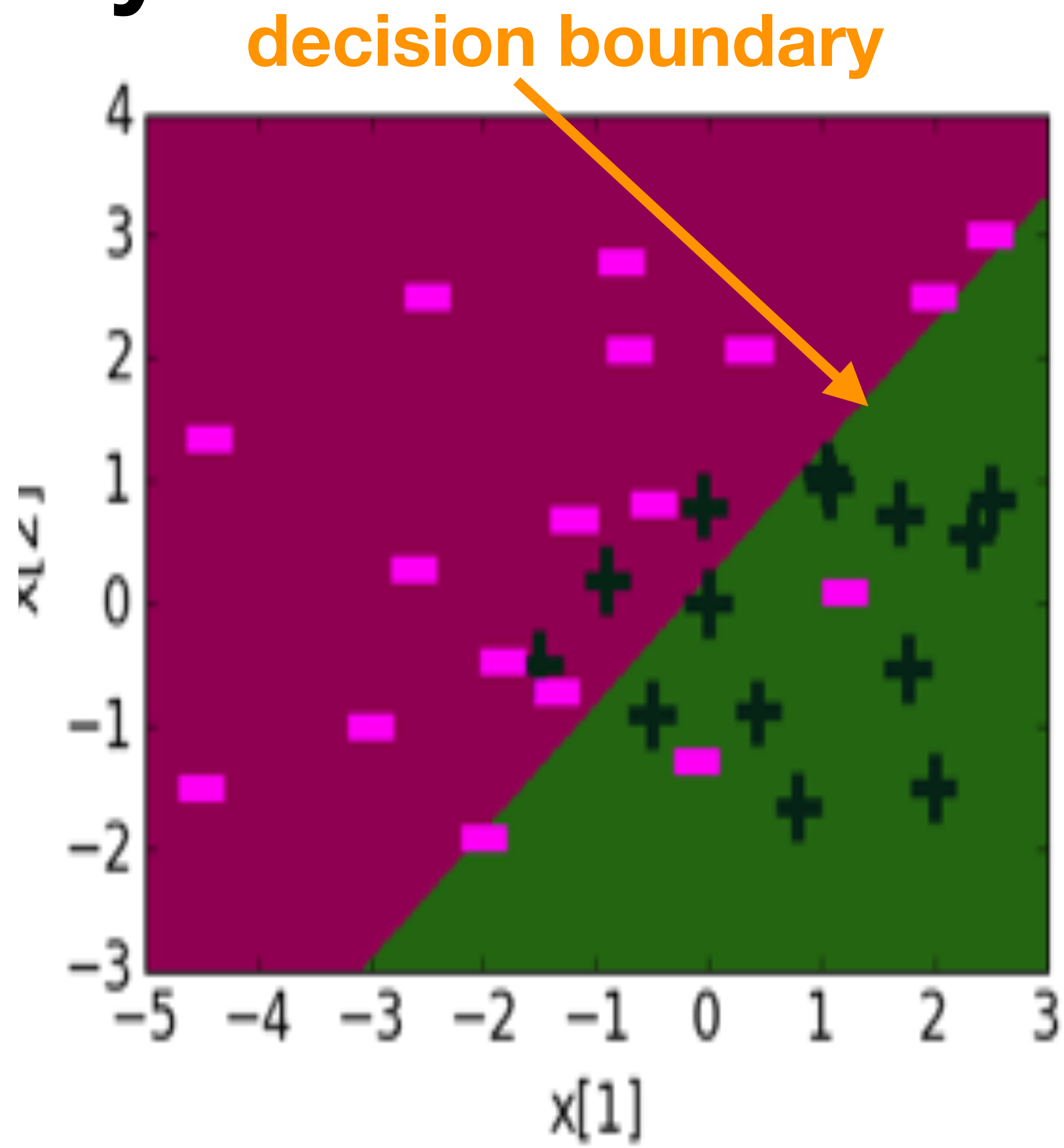
Polynomial features

$$\begin{bmatrix} h_0(x) = 1 \\ h_1(x) = x[1] \\ h_2(x) = x[2] \\ h_3(x) = x[1]^2 \\ h_4(x) = x[2]^2 \\ \vdots \end{bmatrix}$$

- data: ***x*** in 2-dimensions, ***y*** in {+1,-1}
- features: polynomials
- model: linear
- $$f(x) \;=\; w_0 h_0(x) + w_1 h_1(x) + w_2 h_2(x) + \cdots$$

Learned decision boundary



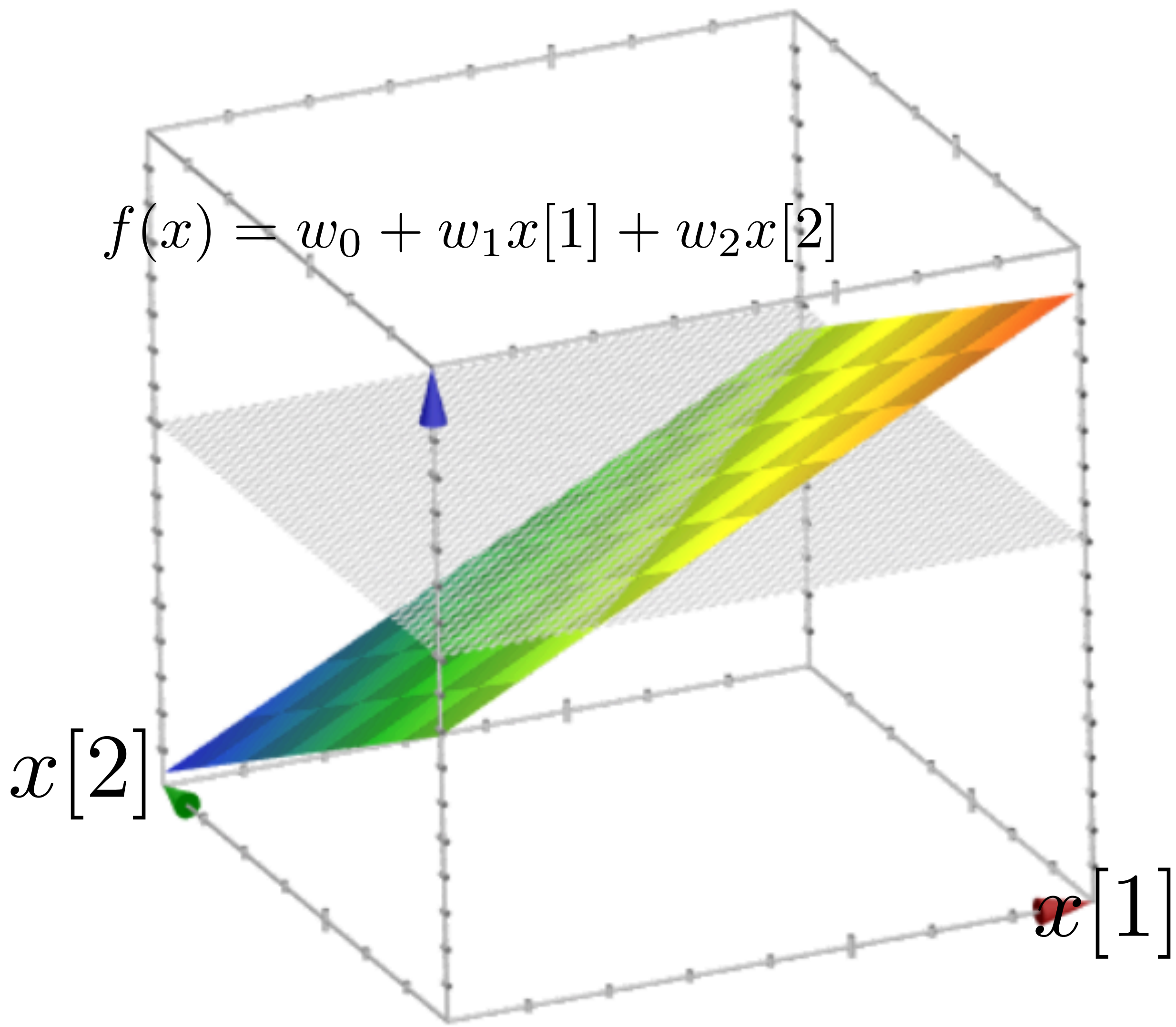


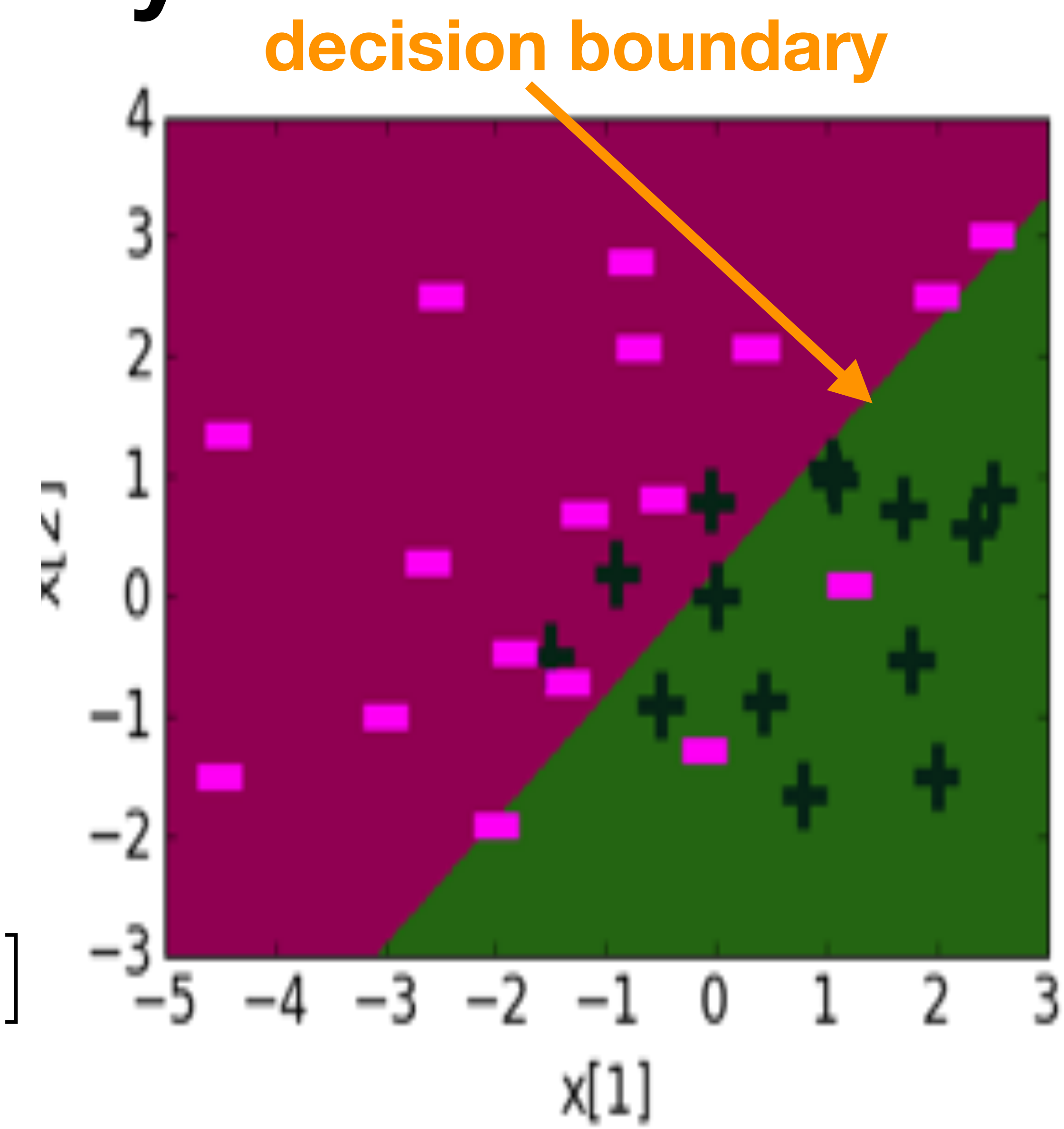
3-d view

Feature	Value	Coefficient
$h_0(x)$	1	0.23
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

Learned decision boundary



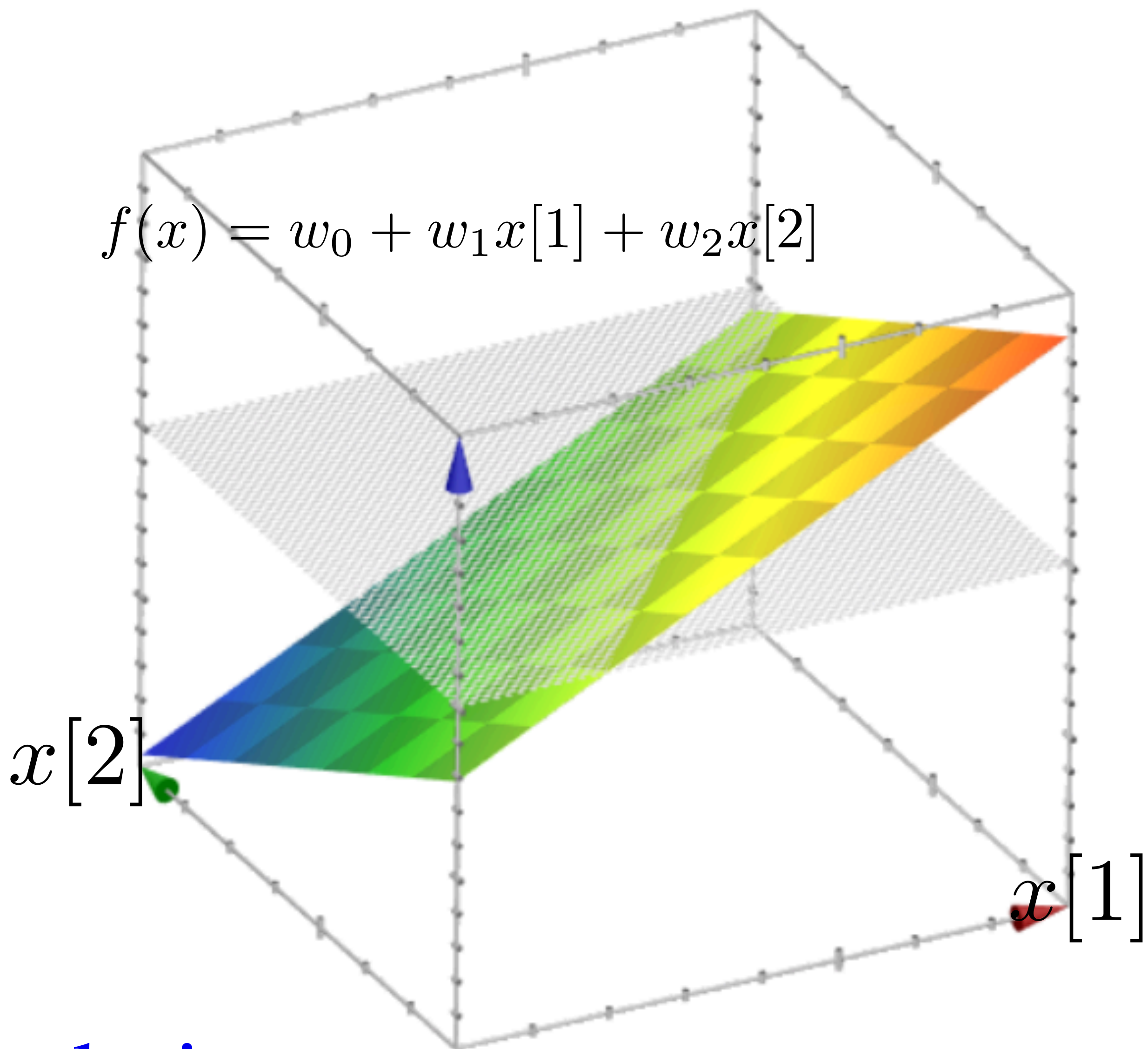


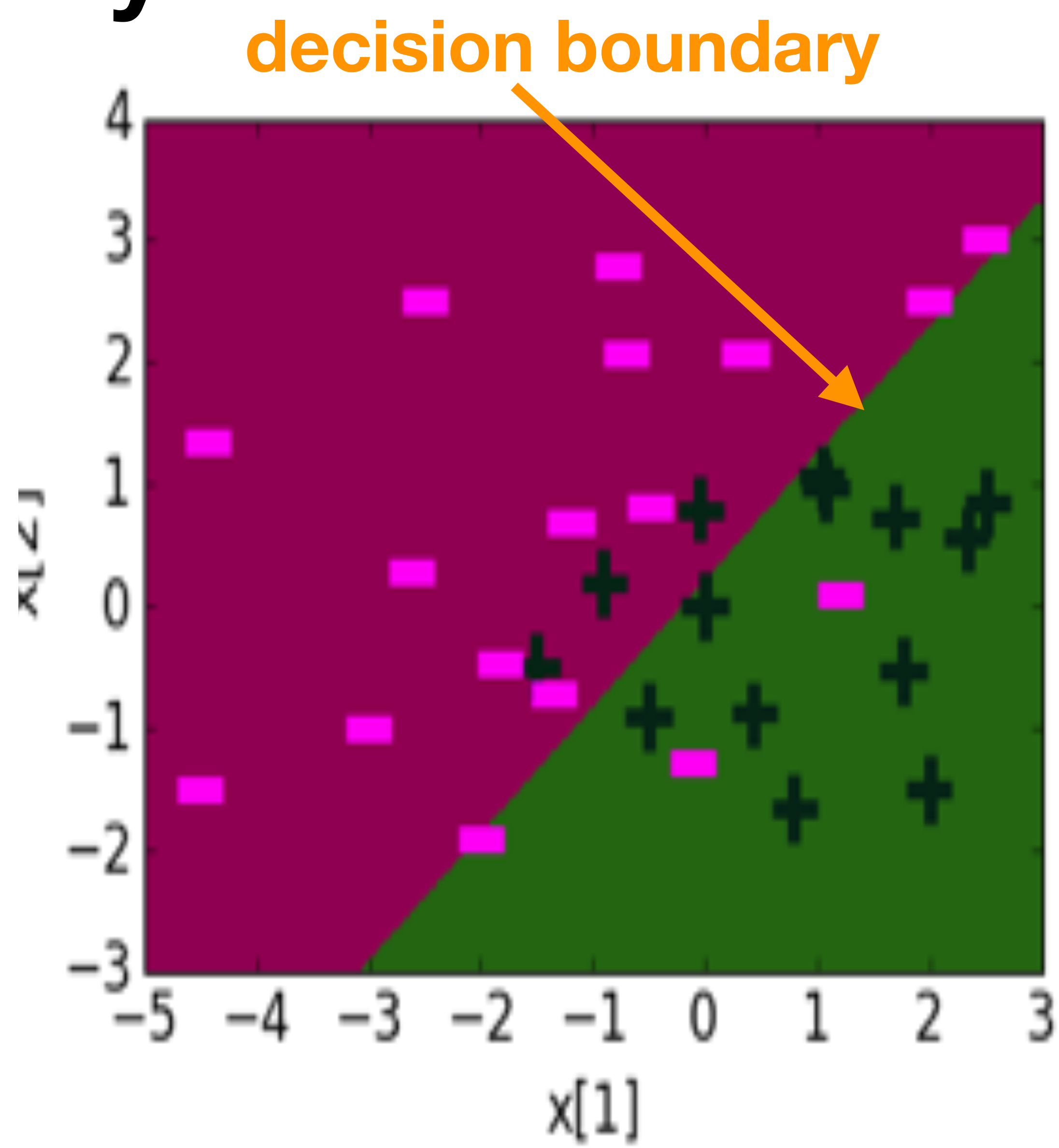
[3-d view]

Feature	Value	Coefficient
$h_0(x)$	1	0.23
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

Learned decision boundary



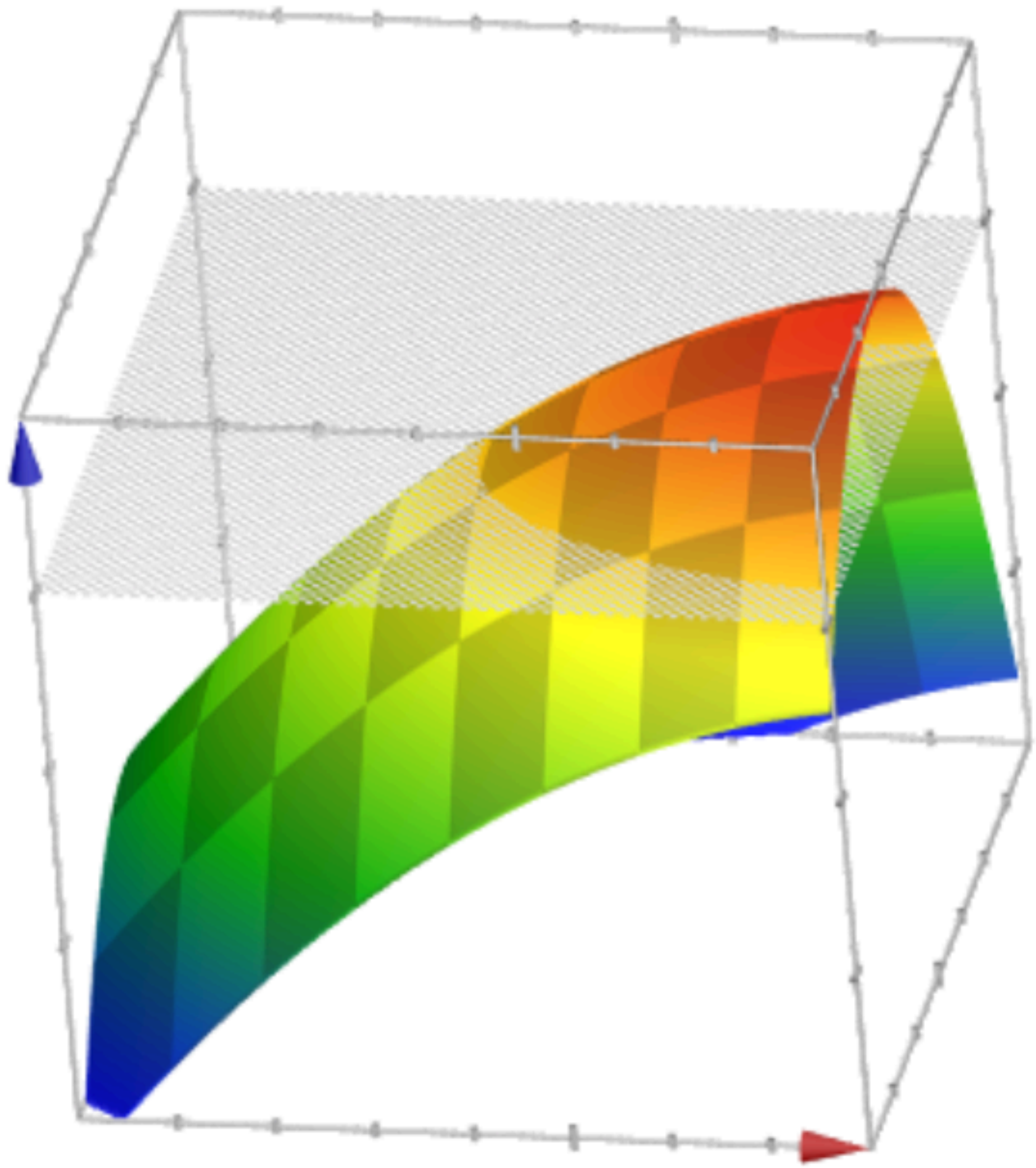


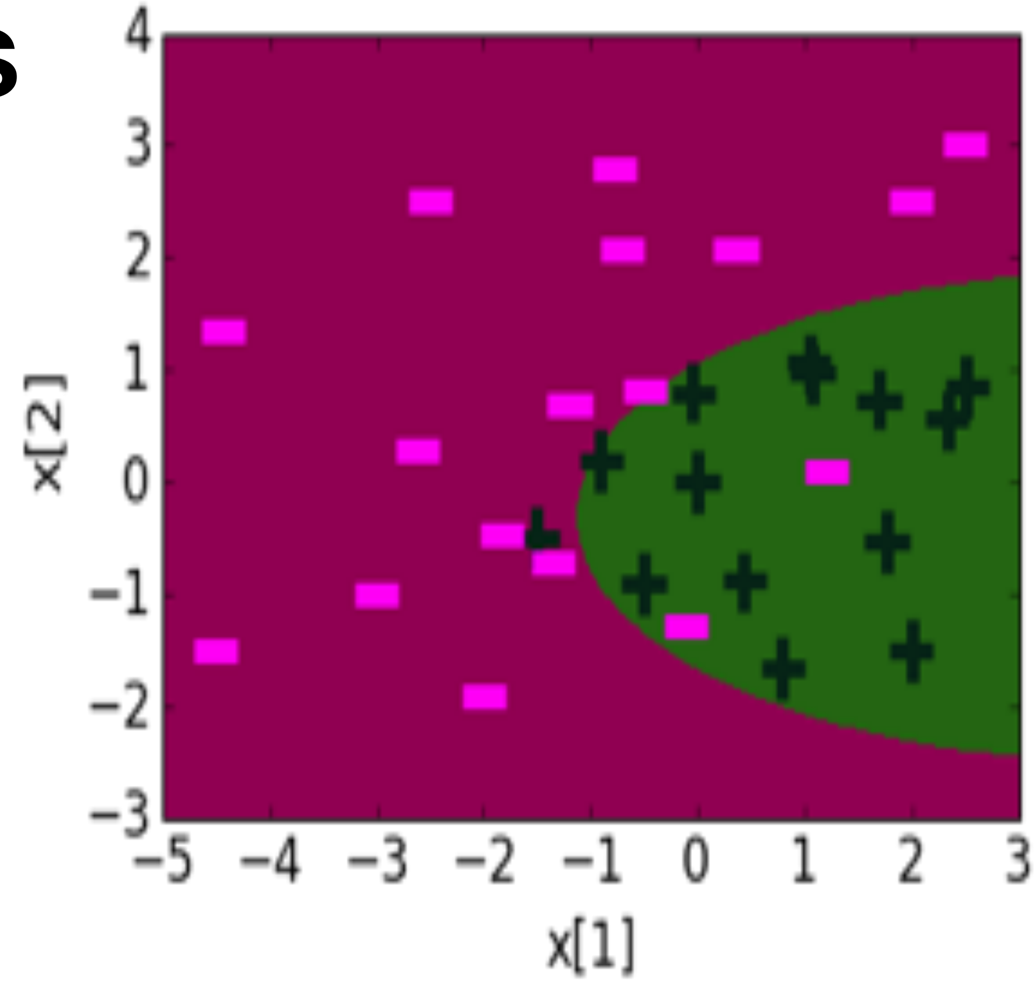
[3-d view]

Feature	Value	Coefficient
$h_0(x)$	1	0.23
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07

- Simple **regression** models had **smooth** predictors
- Simple **classifier** models have **smooth** decision boundaries

Adding quadratic features

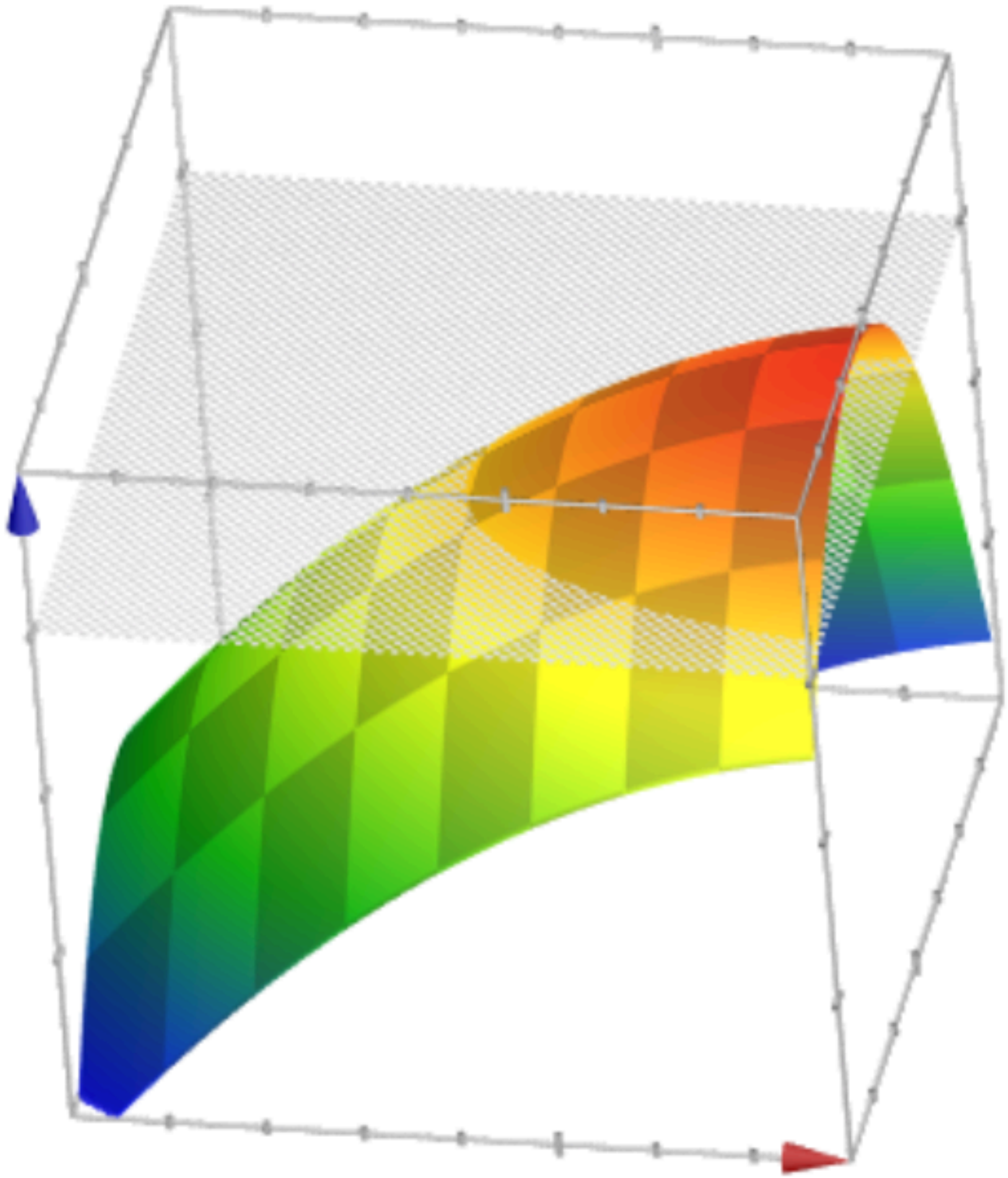


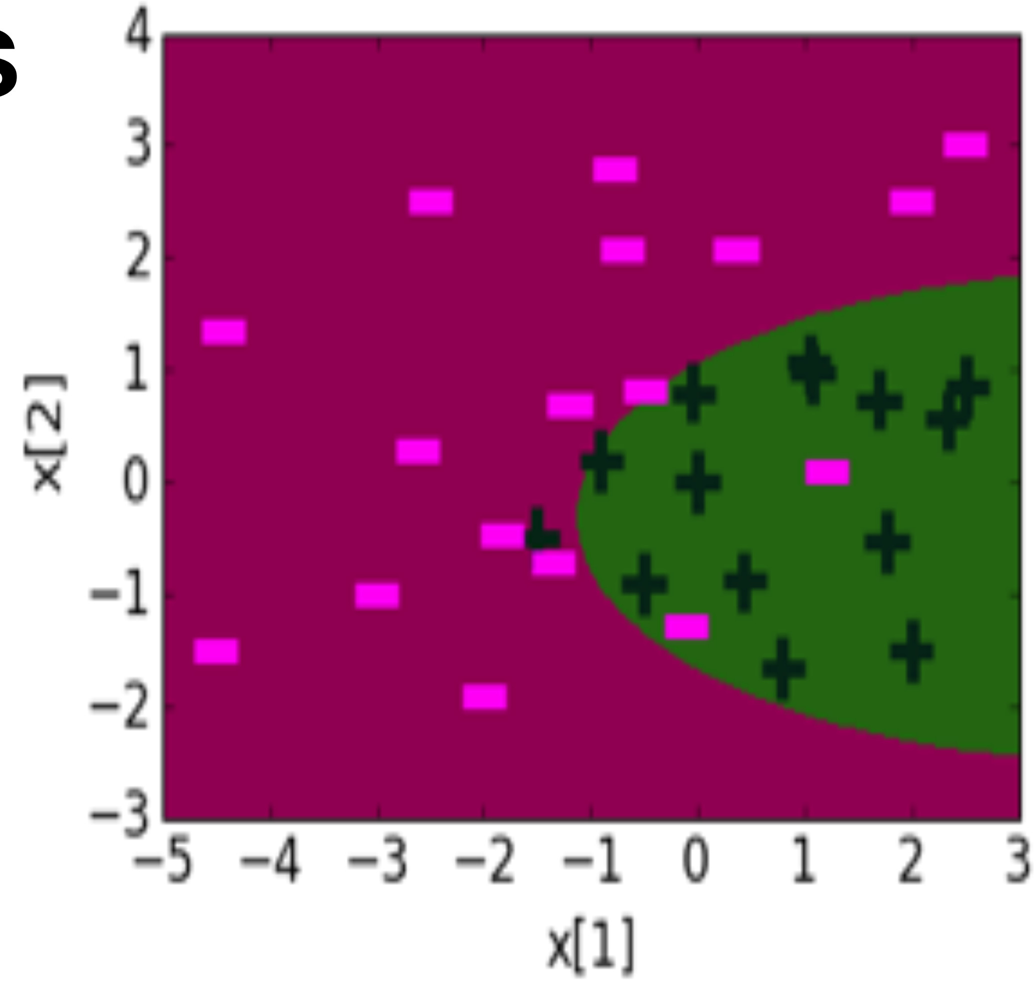


Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

- Adding more features gives more complex models
- Decision boundary becomes more complex

Adding quadratic features

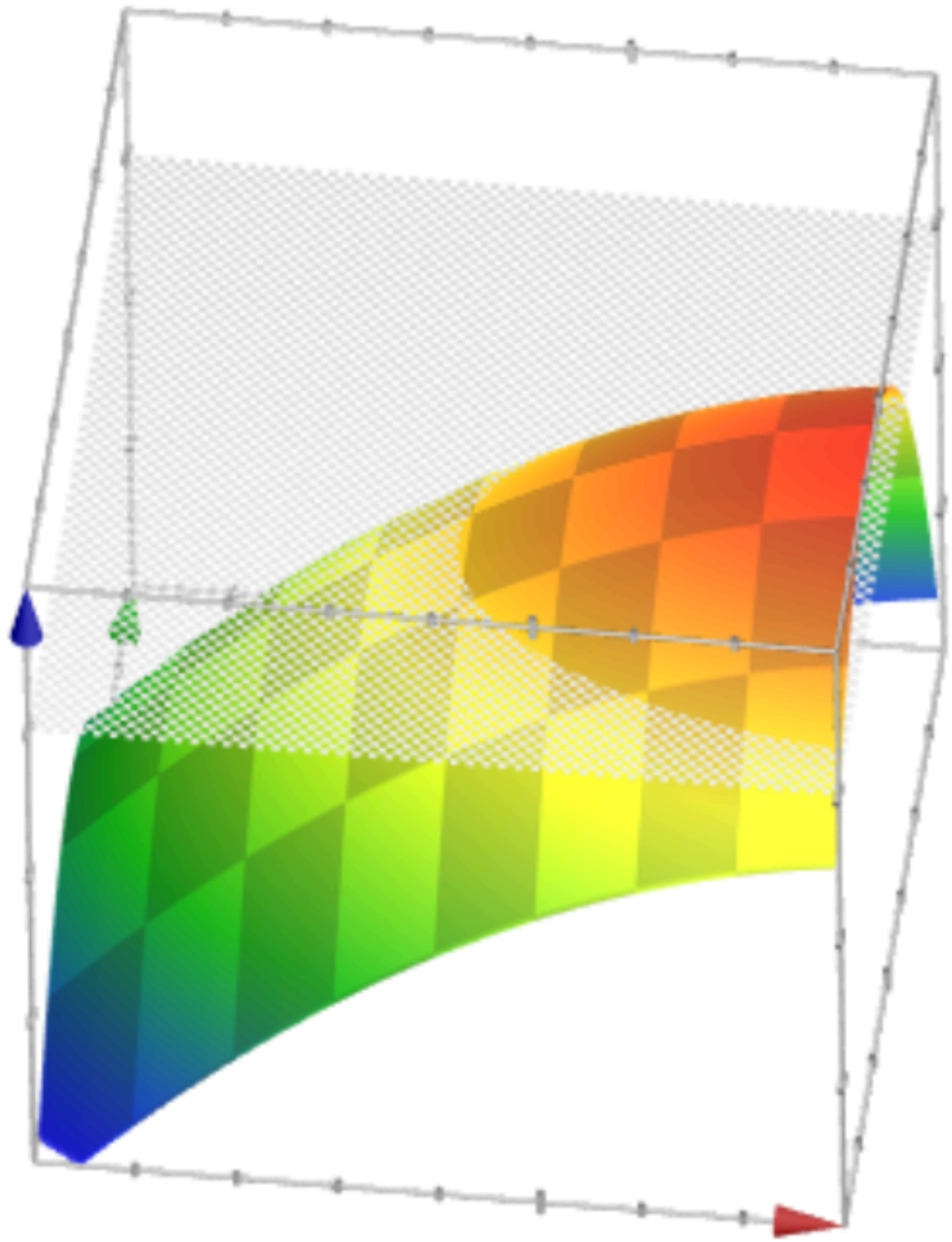


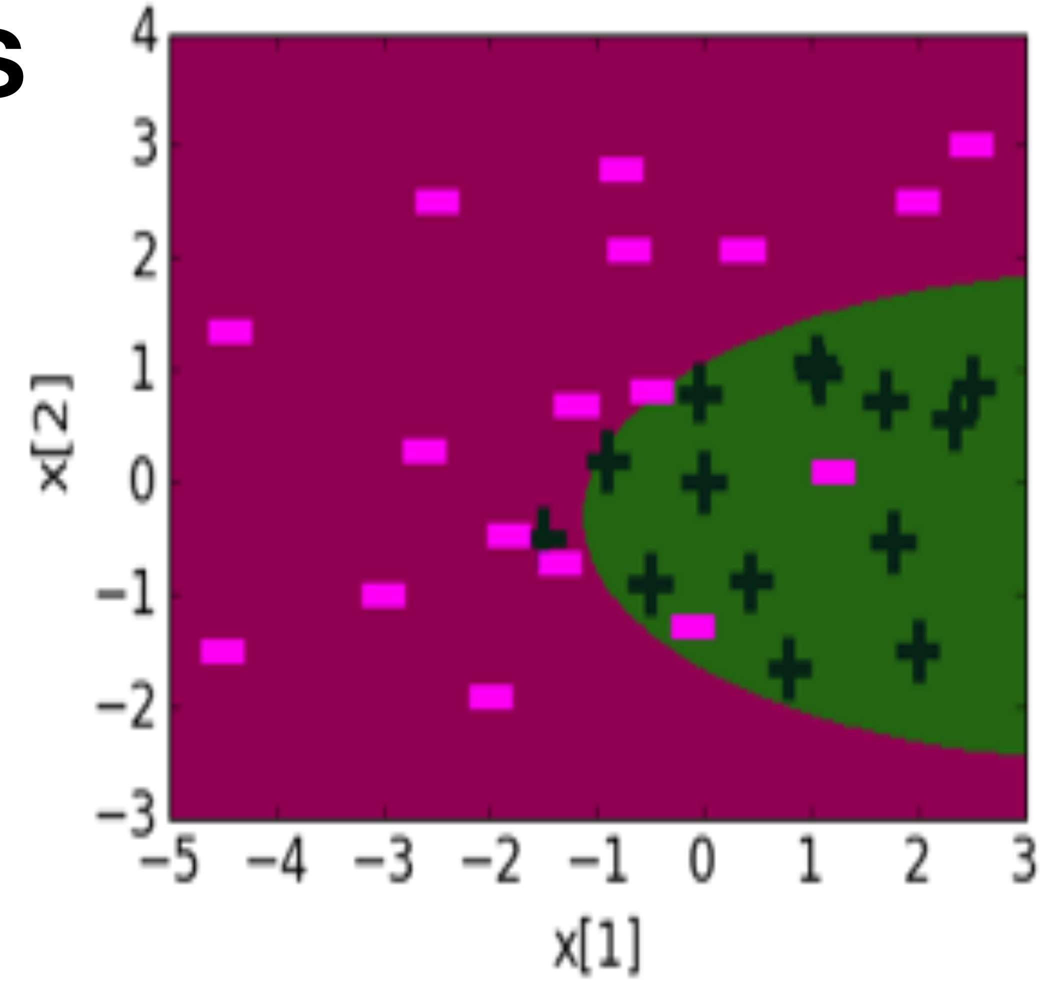


Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

- Adding more features gives more complex models
- Decision boundary becomes more complex

Adding quadratic features

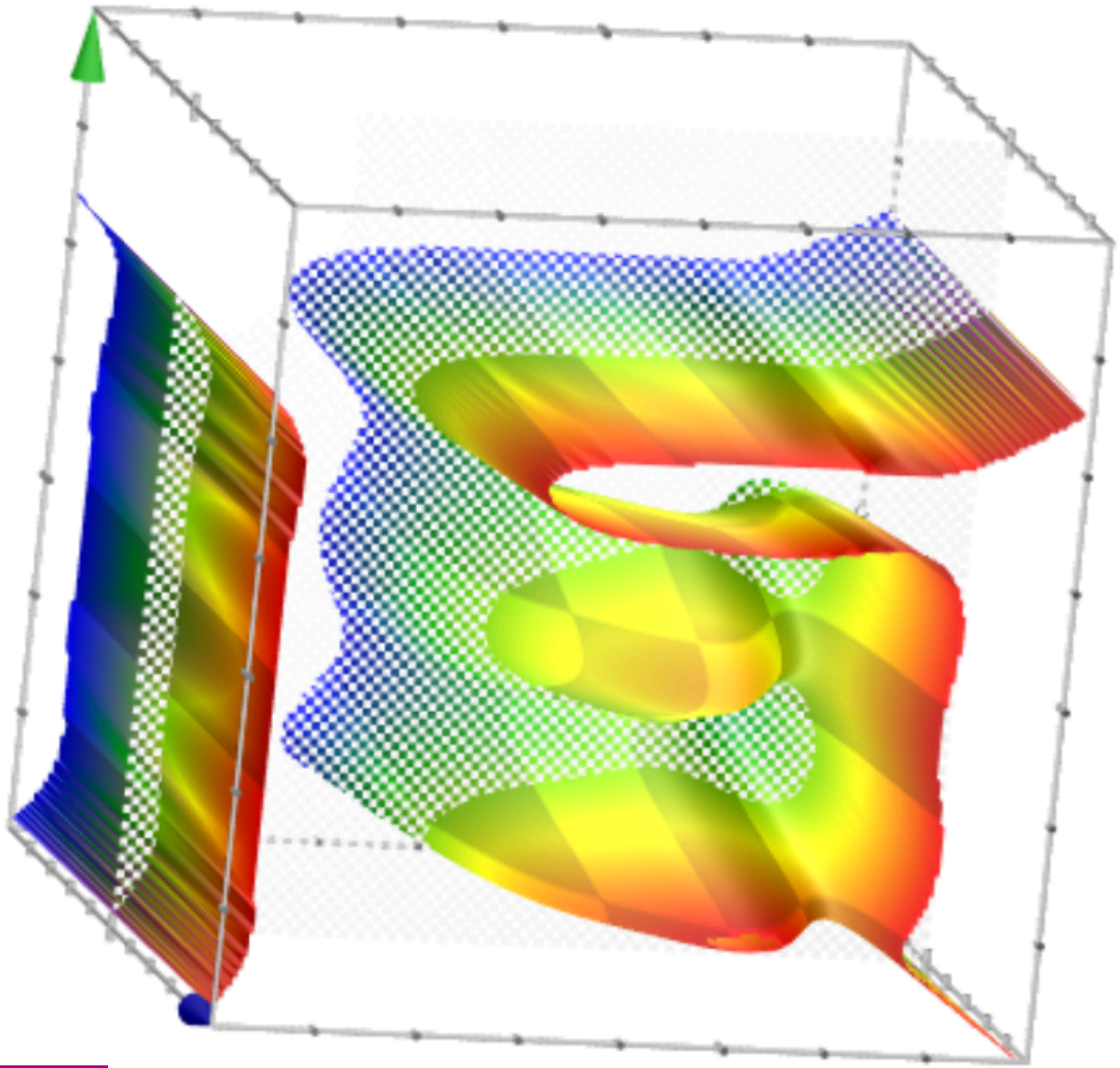




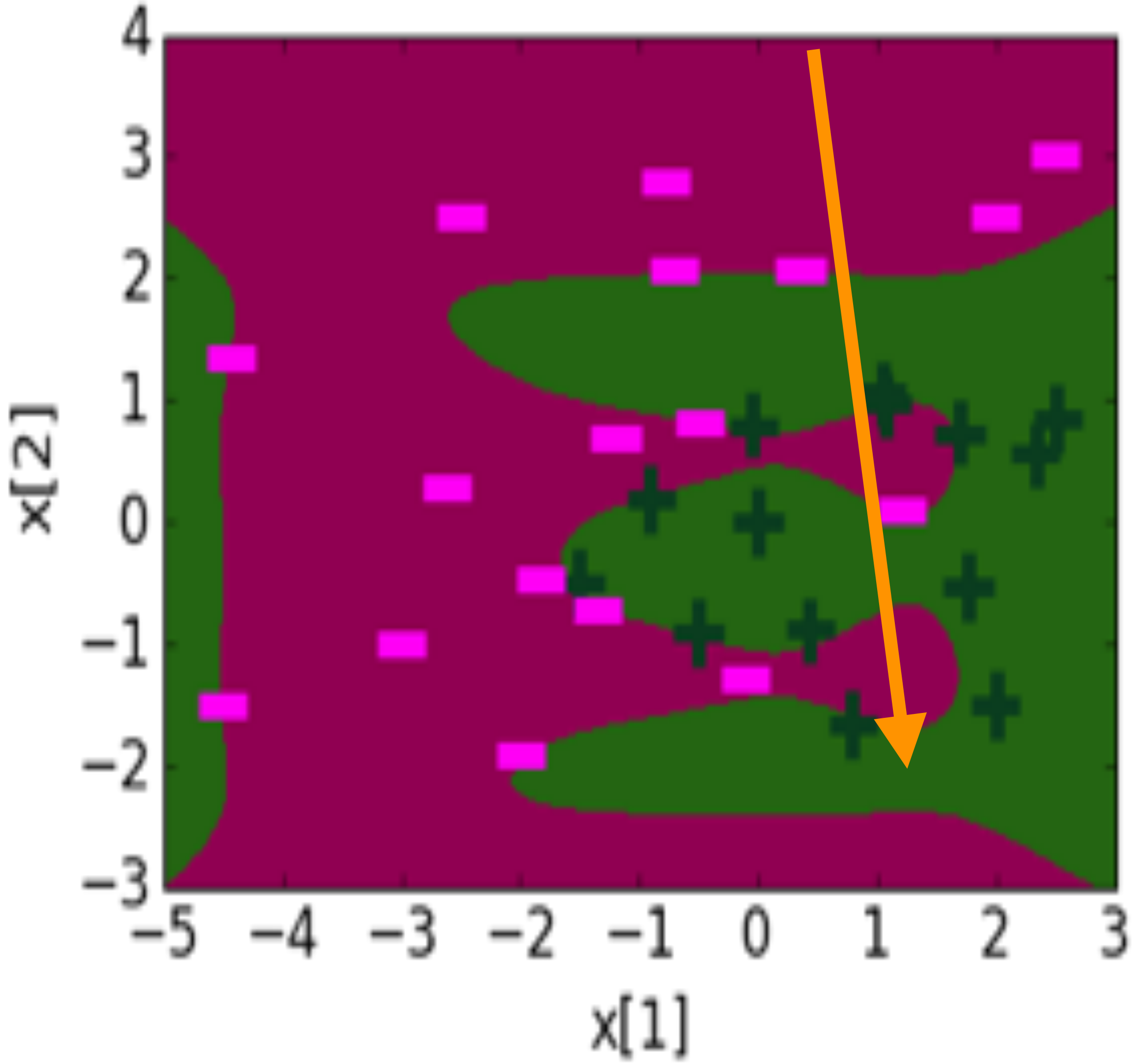
Feature	Value	Coefficient
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96
$h_5(x)$	x[1]x[2]	Omitted

- Adding more features gives more complex models
- Decision boundary becomes more complex

Adding higher degree polynomial features



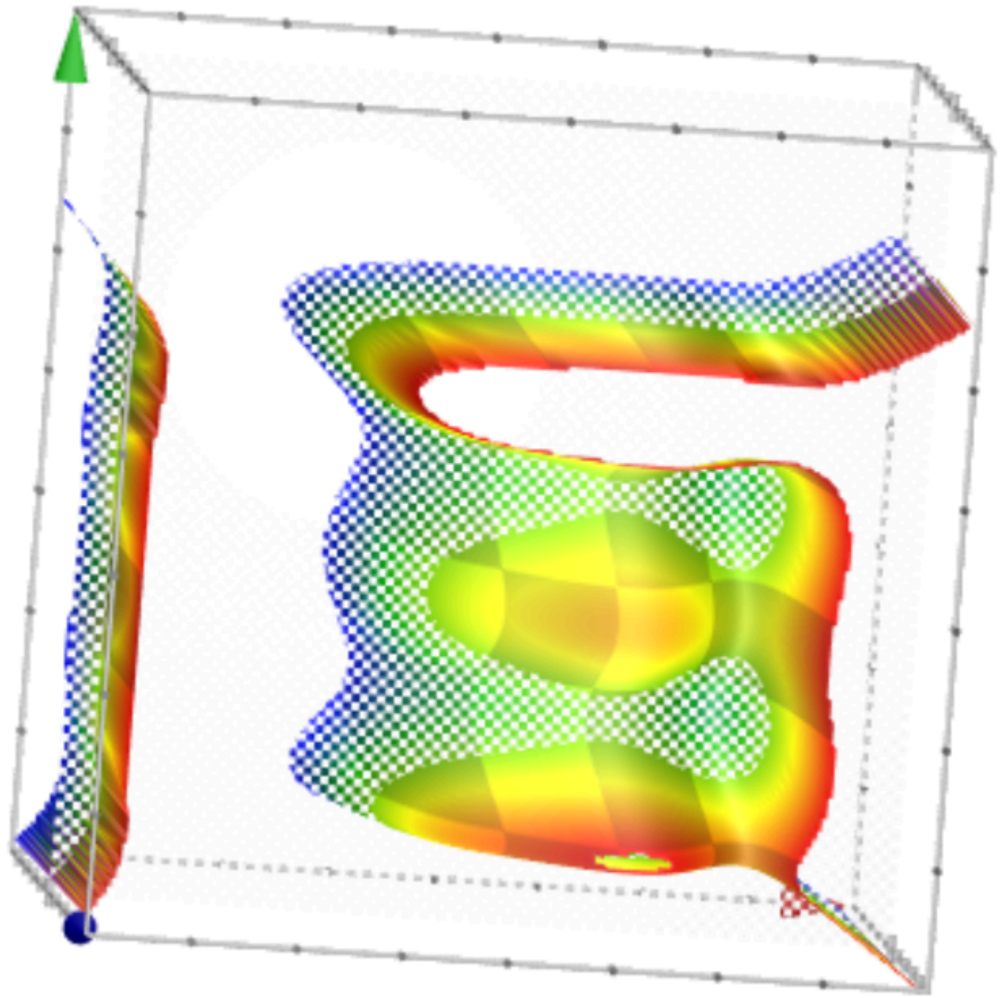
Overfitting leads to non-generalization



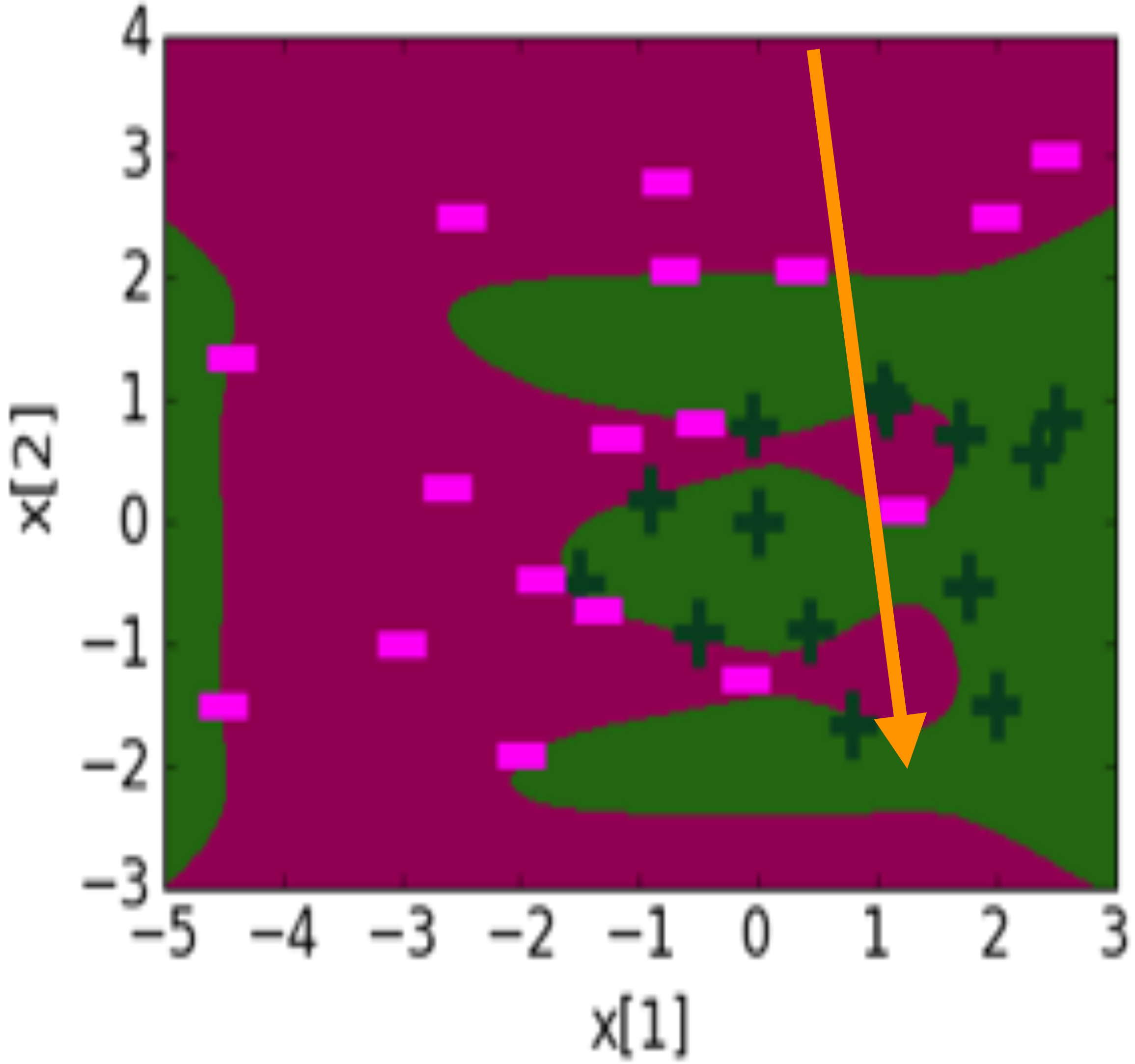
Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	x[1]	5.3
$h_2(x)$	x[2]	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$	$(x[1])^6$	0.8
$h_{12}(x)$	$(x[2])^6$	-8.6

Coefficient values getting large

Adding higher degree polynomial features



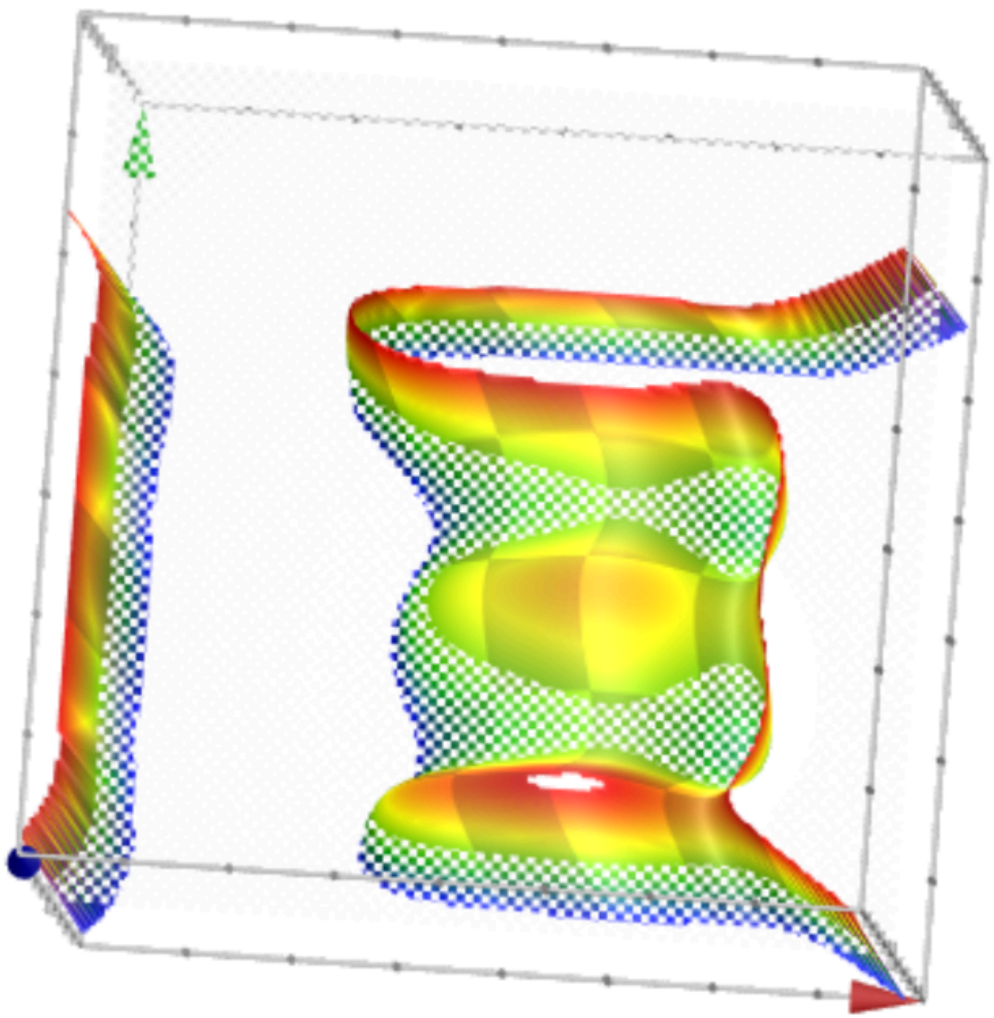
Overfitting leads to non-generalization



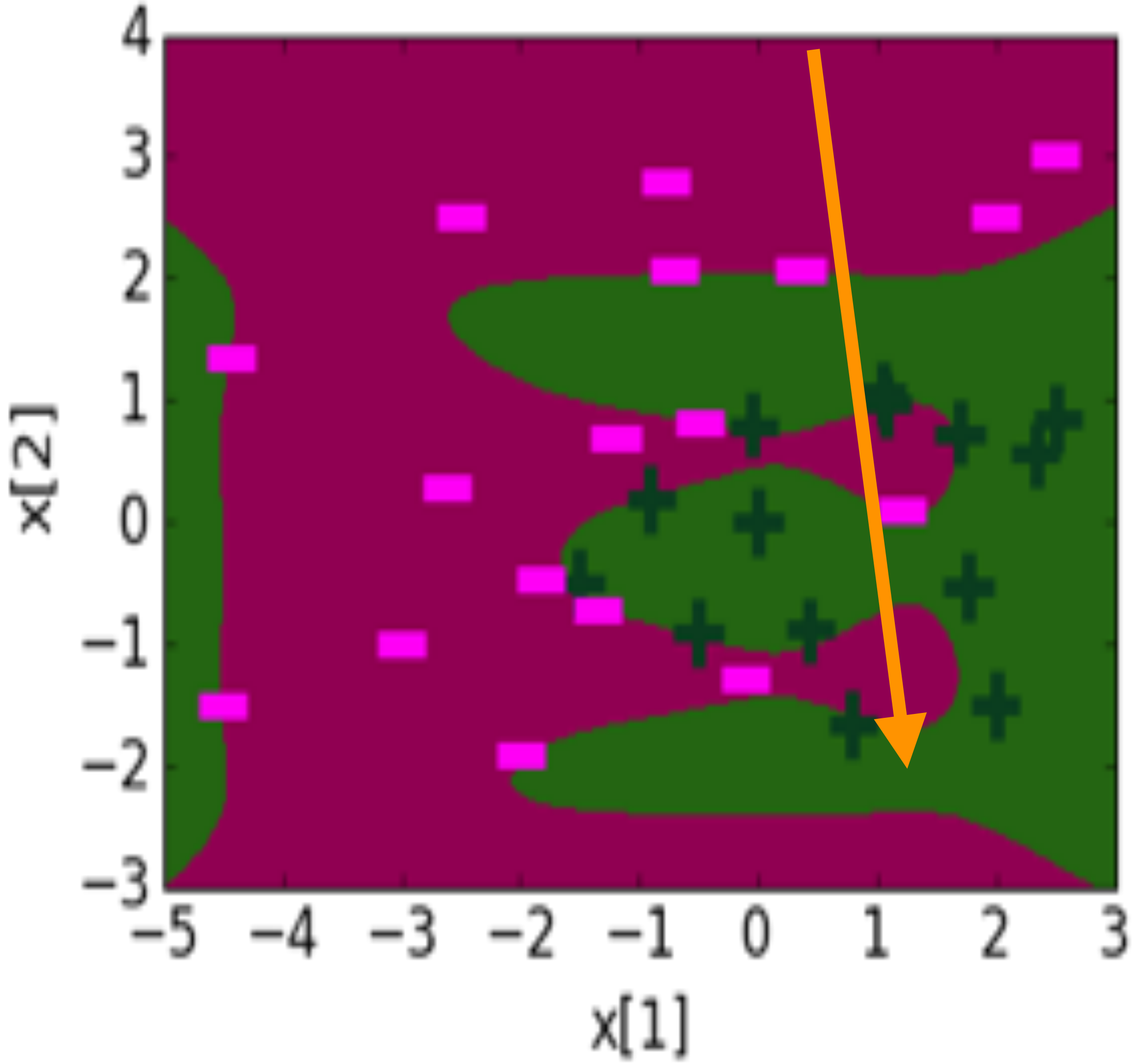
Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	x[1]	5.3
$h_2(x)$	x[2]	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$	$(x[1])^6$	0.8
$h_{12}(x)$	$(x[2])^6$	-8.6

Coefficient values getting large

Adding higher degree polynomial features



Overfitting leads to non-generalization



Feature	Value	Coefficient learned
$h_0(x)$	1	21.6
$h_1(x)$	x[1]	5.3
$h_2(x)$	x[2]	-42.7
$h_3(x)$	$(x[1])^2$	-15.9
$h_4(x)$	$(x[2])^2$	-48.6
$h_5(x)$	$(x[1])^3$	-11.0
$h_6(x)$	$(x[2])^3$	67.0
$h_7(x)$	$(x[1])^4$	1.5
$h_8(x)$	$(x[2])^4$	48.0
$h_9(x)$	$(x[1])^5$	4.4
$h_{10}(x)$	$(x[2])^5$	-14.2
$h_{11}(x)$	$(x[1])^6$	0.8
$h_{12}(x)$	$(x[2])^6$	-8.6

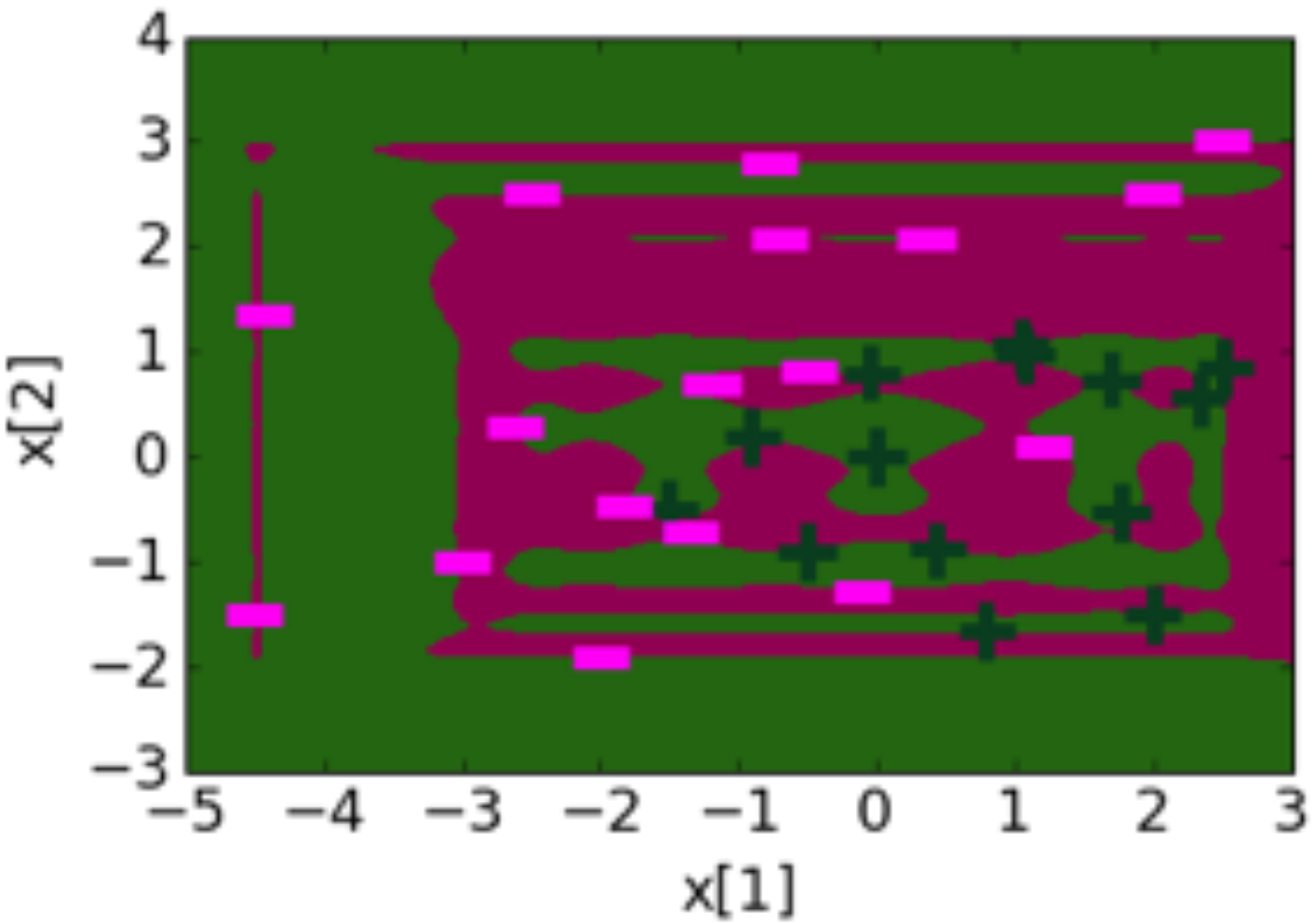
Coefficient values getting large

- Overfitting leads to very large values of

$$f(x) = w_0 h_0(x) + w_1 h_1(x) + w_2 h_2(x) + \cdots$$

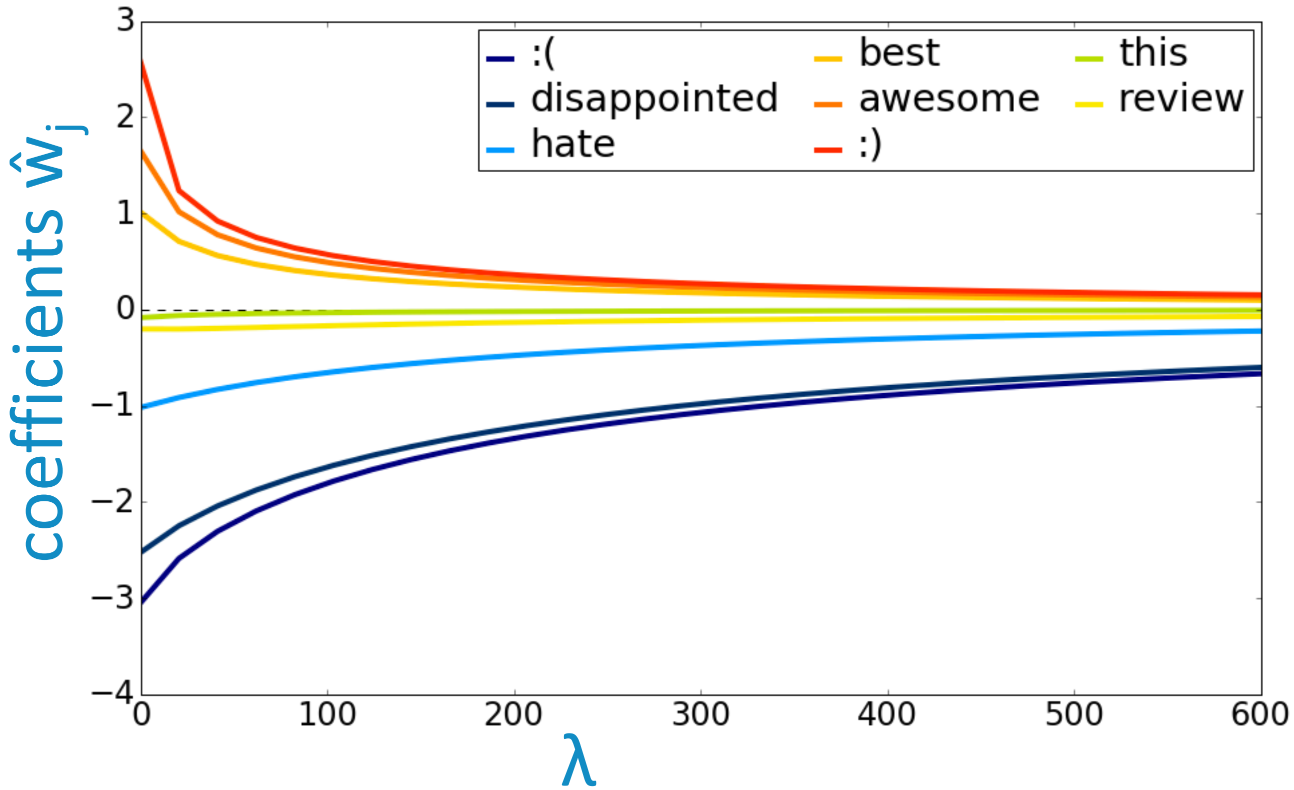
Even higher degree polynomial features

Feature	Value	Coefficient
$h_0(x)$	1	8.7
$h_1(x)$	x[1]	5.1
$h_2(x)$	x[2]	78.7
…	…	…
$h_{11}(x)$	$(x[1])^6$	-7.5
$h_{12}(x)$	$(x[2])^6$	3803
$h_{13}(x)$	$(x[1])^7$	21.1
$h_{14}(x)$	$(x[2])^7$	-2406
…	…	…
$h_{37}(x)$	$(x[1])^{19}$	$-2*10^{-6}$
$h_{38}(x)$	$(x[2])^{19}$	-0.15
$h_{39}(x)$	$(x[1])^{20}$	$-2*10^{-8}$
$h_{40}(x)$	$(x[2])^{20}$	0.03

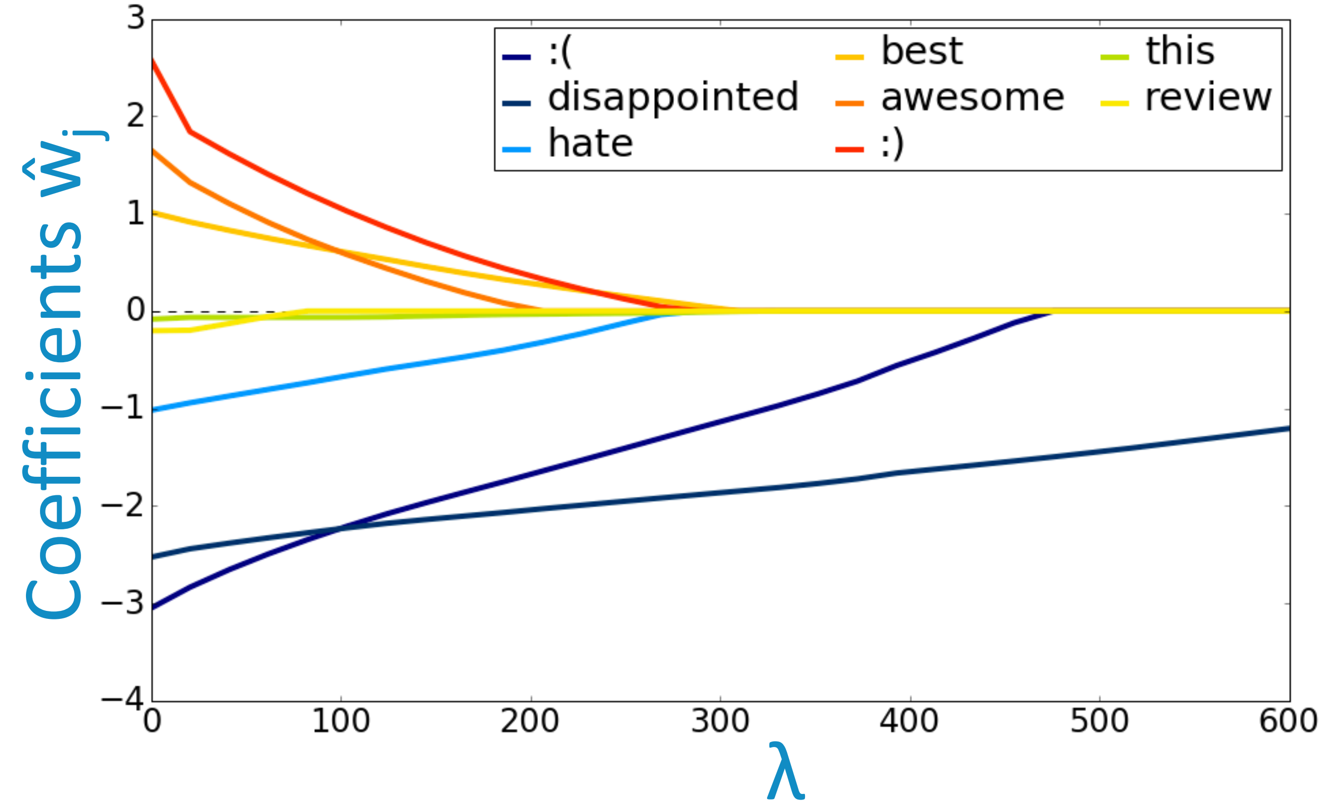


Regularization path

L2 regularizer: $\|W\|_2^2 = |w_1|^2 + \cdots + |w_d|^2$



absolute regularizer: $\|w\|_1 = |w_1| + \cdots + |w_d|$



- Absolute regularizer (a.k.a L1 regularizer) gives sparse parameters, which is desired for interpretability, feature selection, and efficiency

Gradient descent

Iterative algorithms for Empirical Risk Minimization

- $$\text{minimize}_w \quad \underbrace{\sum_{i=1}^{n} \ell(w^T x_i, y_i)}_{\mathcal{L}(w)}$$
- for some convex loss function $\ell(\hat{y}, y)$, which is convex in $\hat{y}$
- we want to find $\hat{w}$ that minimizes the objective function
- if there is no analytical solution (which is the case for logistic regression), we resort to **iterative algorithms** that compute sequence of parameters $w^{(0)}, w^{(1)}, \cdots, w^{(t)}$ each in $\mathbb{R}^d$, hoping that it converges to the minimizer of the objective function
- $w^{(t)}$ is called the ***t*-th iterate**
- $w^{(0)}$ is called the **starting point**
- an algorithm is a **descent method** if
$$\mathcal{L}(w^{(t+1)}) \;\leq\; \mathcal{L}(w^{(t)})$$
each iterate is better than the previous one

Gradient Descent

$\mathcal{L}(w)$ - differentiable $\longleftrightarrow$ $\nabla_w \mathcal{L}(w)$

at time $t+1$-th iterate,

make an affine Taylor expansion of $\mathcal{L}(w)$ around current iterate $w^{(t)}$.

$$\hat{\mathcal{L}}(w; w^{(t)}) = \mathcal{L}(w^{(t)}) + \nabla \mathcal{L}(w^{(t)})^T \cdot (w - w^{(t)})$$

① $\hat{\mathcal{L}}(w; w^{(t)}) \simeq \mathcal{L}(w)$

② if $\|w - w^{(t)}\|_2^2$ is small

$$w^{(t+1)} \longleftarrow \underset{w}{\operatorname{argmin}} \; \hat{\mathcal{L}}(w; w^{(t)}) + \frac{1}{2h^{(t)}} \|w - w^{(t)}\|_2^2$$

$h^{(t)} > 0$ scalar. learning rate, step size.

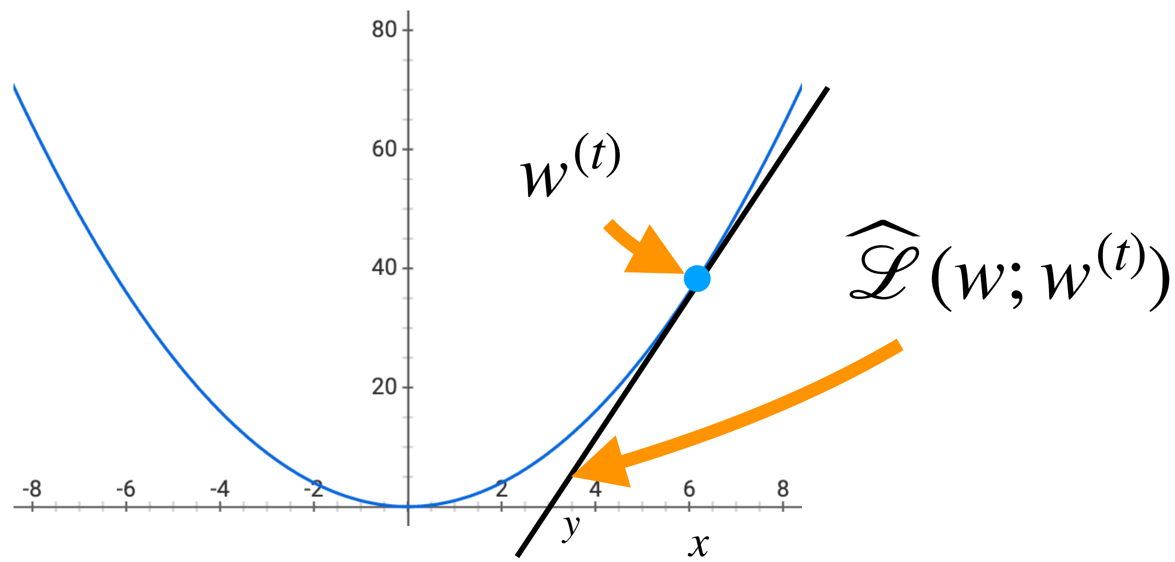
$$w^{(t+1)} \longleftarrow w^{(t)} - h^{(t)} \cdot \nabla \mathcal{L}(w^{(t)})$$

Gradient descent

- suppose $\mathscr{L}(w)$ is differentiable, so gradient exists every $w \in \mathbb{R}^d$
- at (t+1)-th iteration, create **affine Taylor approximation** of $\mathscr{L}(w)$ around current iterate $w^{(t)}$

$$\widehat{\mathscr{L}}(w; w^{(t)}) = \mathscr{L}(w^{(t)}) + \nabla \mathscr{L}(w^{(t)})^T (w - w^{(t)})$$

- this approximation is more accurate, $\widehat{\mathscr{L}}(w; w^{(t)}) \approx \mathscr{L}(w)$, for w near $w^{(t)}$
- hence, we choose $w^{(t+1)}$ that
    - makes $\widehat{\mathscr{L}}(w^{(t+1)}; w^{(t)})$ small
    - while keeping $\|w^{(t+1)} - w^{(t)}\|_2^2$



$$w^{(t+1)} \leftarrow \arg\min_w \widehat{\mathscr{L}}(w; w^{(t)}) + \frac{1}{2h^{(t)}} \|w - w^{(t)}\|_2^2$$

- where $h^{(t)} > 0$ is a trust parameter or step length or learning rate
- the optimal solution of the above update rule is

$$w^{(t+1)} \leftarrow w^{(t)} - h^{(t)} \nabla \mathscr{L}(w^{(t)})$$

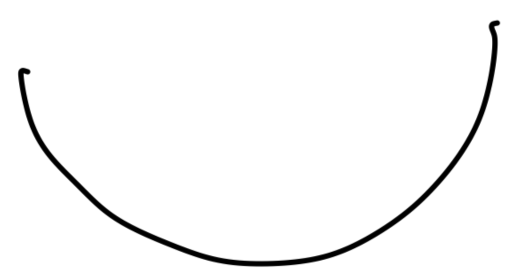
- roughly, take a step in the direction of negative gradient

$$w^{(t+1)} \leftarrow \arg\min_{w} \nabla L(w^{(t)})^T (w - w^{(t)}) + \frac{1}{2h^{(t)}} \|w - w^{(t)}\|_2^2$$

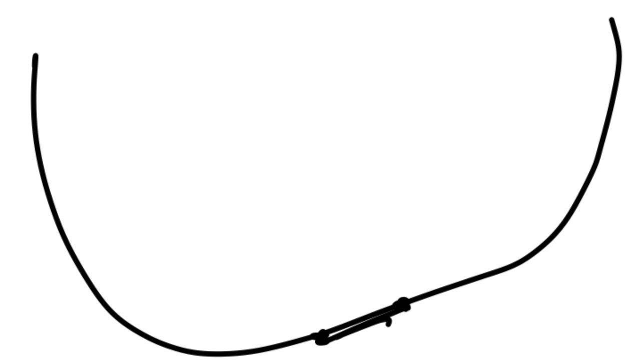
$$\leftarrow \arg\min_{w} \frac{1}{2h^{(t)}} \left\| \underbrace{(w - w^{(t)}) + h^{(t)} \nabla L(w^{(t)})}_{=0} \right\|_2^2 + \text{Constant}$$

$$w^{(t+1)} \leftarrow w^{(t)} - h^{(t)} \cdot \nabla L(w^{(t)}).$$

$h^{(t)} \leftarrow$ is related to $L(w)$



strong convex



non-strongly convex

$$h^{(t+1)} \leftarrow \frac{1}{2} h^{(t)}$$

Gradient descent update

- at each iteration, we want update $w^{(t+1)}$ as the minimizer of

$$\mathcal{L}(w^{(t)}) + \nabla \mathcal{L}(w^{(t)})^T (w - w^{(t)}) + \frac{1}{2h^{(t)}} \|w - w^{(t)}\|_2^2$$

- this can be re-written as

$$\mathcal{L}(w^{(t)}) + \frac{1}{2h^{(t)}} \left\| (w - w^{(t)}) + h^{(t)} \nabla \mathcal{L}(w^{(t)}) \right\|_2^2 - \frac{h^{(t)}}{2} \|\nabla \mathcal{L}(w^{(t)})\|_2^2$$

- as the first and third terms don't depend on w
- middle term is minimized (and made zero) by choosing

$$w^{(t+1)} \quad \leftarrow \quad w^{(t)} - h^{(t)} \nabla \mathcal{L}(w^{(t)})$$

- this is how we update iterates in gradient descent
- in practice, $h^{(t)}$ is fixed as a constant until no progress is being made and then decreased by $h^{(t+1)} = h^{(t)}/2$

Gradient descent convergence

- (under some technical conditions) we have
$$\|\nabla \mathscr{L}(w^{(t)})\|_2^2 \to 0 \text{ as } t \to \infty$$
- i.e., the gradient descent method always finds a global minimum of a **differentiable convex** function

Gradient descent for ERM

- to implement gradient descent on a given ERM, one needs to compute the gradient (which is typically done automatically via auto differentiation) and choose hyper-parameters
- we can manually compute the gradient as

$$\mathscr{L}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(w^T x_i, y_i)$$

$\nabla_w \ell(w^T x_i, y_i) = \ell'(w^T x_i, y_i) x_i$

$$\nabla \mathscr{L}(w) = \frac{1}{n}\sum_{i=1}^{n} \ell'(w^T x_i, y_i) x_i$$

where $\ell'(\hat{y}, y)$ is derivative of $\ell(\hat{y}, y)$ with respect to its first argument $\hat{y}$

- this can be done via
  - first, compute n-dim vector $\hat{y}^{(t)} = \mathbf{X} w^{(t)}$ **2nd operations**
  - next, compute n-dim vector $z^{(t)}$ with each entry $z_i^{(t)} = \ell'(\hat{y}_i^{(t)}, y_i)$ **n operations**
  - finally, compute d-dim vector $\nabla \mathscr{L}(w^{(t)}) = \frac{1}{n}\mathbf{X}^T z^{(t)}$ **2nd operations**

Gradient descent for logistic regression

- the logistic loss is (for $\hat{y} = w^T x$)
  $\ell(\hat{y}, y) \;=\; \log(1 + e^{-y\hat{y}}) = \log(1 + e^{-y(w^T x)})$
- the derivative is $\ell'(\hat{y}, y) \;=\; \dfrac{\partial \ell(\hat{y}, y)}{\partial \hat{y}} \;=\; \dfrac{-y\, e^{-y\hat{y}}}{1 + e^{-y\hat{y}}}$
- the gradient is

$$\nabla \mathscr{L}(w^{(t)}) \;=\; \frac{1}{n}\sum_{i=1}^{n} \ell'(w^T x_i, y_i) x_i \;=\; \frac{1}{n}\sum_{i=1}^{n} \frac{-y_i\, e^{-y_i w^T x_i}}{1 + e^{-y_i w^T x_i}} x_i$$

- $4nd + n \approx 4nd$ operations per iteration

Stochastic gradient descent for logistic regression

- recall the **gradient descent** for ERM is

$$\mathcal{L}(w) \;=\; \frac{1}{n}\sum_{i=1}^{n} \ell(w^T x_i, y_i)$$

$$\nabla \mathcal{L}(w) \;=\; \frac{1}{n}\sum_{i=1}^{n} \ell'(w^T x_i, y_i) x_i$$

$$w^{(t+1)} \;\leftarrow\; w^{(t)} - h^{(t)}\, \nabla \mathcal{L}(w^{(t)})$$

- as gradient computation can be slow (4nd operations) for large training data with large n,
- **stochastic gradient descent (SGD)** approximates the gradient by a **minibatch** of sampled gradients
  - choose the size m of minibatches to be used
  - at each iteration, randomly sample a minibatch of size m

$$S^{(t)} = \{i_1^{(t)}, \ldots, i_m^{(t)}\}$$

  - compute stochastic gradient update

$$w^{(t+1)} \;\leftarrow\; w^{(t)} - h^{(t)} \frac{1}{m} \sum_{i \in S^{(t)}} \ell'(w^T x_i, y_i) x_i$$

Stochastic gradient descent

- each update requires 4md operations
- this is a **stochastic (random)** approximation of the actual full gradient

- this is an unbiased estimate of the full gradient

$$
\begin{aligned}
\mathbb{E}_{S^{(t)}}\Big[\frac{1}{m}\sum_{i\in S^{(t)}} \ell'(w^T x_i, y_i)x_i\Big] &= \frac{1}{m}\sum_{i=1}^{m} \mathbb{E}_{i\sim\text{Uniform}\{1,\ldots,n\}}[\ell'(w^T x_i, y_i)x_i] \\
&= \mathbb{E}_{i\sim\text{Uniform}\{1,\ldots,n\}}[\ell'(w^T x_i, y_i)x_i] \\
&= \frac{1}{n}\sum_{i=1}^{n} \ell'(w^T x_i, y_i)x_i
\end{aligned}
$$

- choosing a small batch size m is faster, but has large variance
- choosing a large batch size m is slower, but has small variance

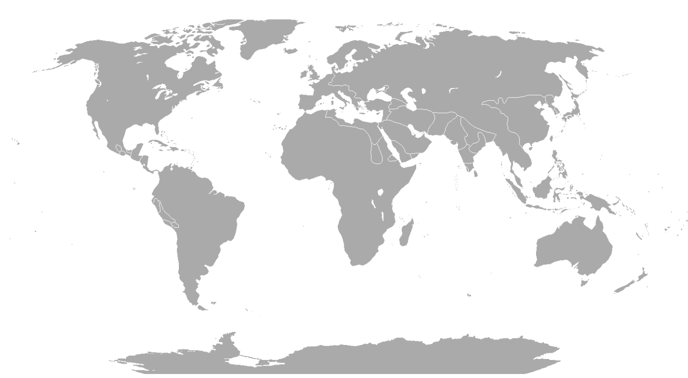
- This is another hyper-parameter you tune, in practice

Multi-class classification

How do we encode categorical data *y*?

- so far, we considered Boolean case where there are two categories
- encoding y is simple: {+1,-1}, as there is not much difference

- multi-class classification predicts categorial y
- taking values in $C = \{c_1, \ldots, c_k\}$
- c_j's are called **classes** or **labels**
- examples:



Country of birth

(Argentina, Brazil, USA,...)

Zipcode

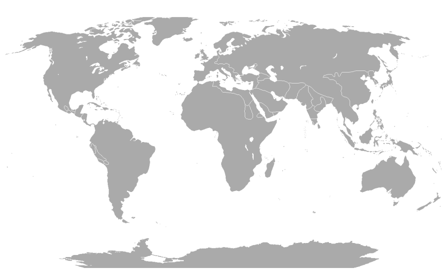
(10005, 98195,...)

All English words

- a **k-class classifier** predicts y given x

Embedding c_j's in real values

- for optimization we need to **embed** raw categorical c_j's into real valued vectors
- there are many ways to embed categorial data
  - True->1, False->-1
  - Yes->1, Maybe->0, No->-1
  - Yes->(1,0), Maybe->(0,0), No->(0,1)
  - Apple->(1,0,0), Orange->(0,1,0), Banana->(0,0,1)
  - Ordered sequence:
    (Horse 3, Horse 1, Horse 2) -> (3,1,2)
- we use **one-hot embedding** (a.k.a. **one-hot encoding**)
  - each class is a standard basis vector in $k-$dimension



Country of birth
(Argentina, Brazil, USA,...)

196 categories

1-hot encoding

x	$h_1(x)$	$h_2(x)$	…	$h_{195}(x)$	$h_{196}(x)$
Brazil		1			
Zimbabwe					1

196 features

Multi-class logistic regression

- data: categorical y in $\{c_1, \ldots, c_k\}$ with k categories

  we use one-hot encoding, s.t. $y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ implies that $y = c_1$

- model: linear vector-function makes a linear prediction $\hat{y} \in \mathbb{R}^k$

$$\hat{y}_i \; = \; f(x_i) \; = \; w^T x_i$$

with model parameter matrix $w \in \mathbb{R}^{d \times k}$ and sample $x_i \in \mathbb{R}^d$

$$f(x_i) \; = \; \begin{bmatrix} f_1(x_i) \\ f_2(x_i) \\ \vdots \\ f_k(x_i) \end{bmatrix} \; = \; \underbrace{\begin{bmatrix} w_{1,0} & w_{1,1} & w_{1,2} & \cdots \\ w_{2,0} & w_{2,1} & w_{2,2} & \cdots \\ \vdots & & & \\ w_{k,0} & w_{k,1} & w_{k,2} & \cdots \end{bmatrix}}_{w^T} \underbrace{\begin{bmatrix} 1 \\ x_i[1] \\ \vdots \\ x_i[d] \end{bmatrix}}_{x_i} \; = \; \begin{bmatrix} w_{1,0} + w_{1,1}x_i[1] + w_{1,2}x_i[2] + \cdots \\ w_{2,0} + w_{2,1}x_i[1] + w_{2,2}x_i[2] + \cdots \\ \vdots \\ w_{k,0} + w_{k,1}x_i[1] + w_{k,2}x_i[2] + \cdots \end{bmatrix}$$

$$w = \begin{bmatrix} w[:,1] & w[:,2] & \cdots & w[:,k] \end{bmatrix} \in \mathbb{R}^{(d+1) \times k}, \quad f_j(x_i) = W[:,j]^T \cdot x_i$$

- Logistic regression

2 classes

$$\mathbb{P}(y_i = -1 \mid x_i) = \frac{1}{1 + e^{w^T x_i}}$$

$$\mathbb{P}(y_i = +1 \mid x_i) = \frac{1}{1 + e^{-w^T x_i}}$$

k classes

$$\mathbb{P}(y_i = c_1 \mid x_i) = \frac{e^{w[:,1]^T x_i}}{e^{w[:,1]^T x_i} + \cdots + e^{w[:,k]^T x_i}}$$

$$\vdots$$

$$\mathbb{P}(y_i = c_k \mid x_i) = \frac{e^{w[:,k]^T x_i}}{e^{w[:,1]^T x_i} + \cdots + e^{w[:,k]^T x_i}}$$

Maximum Likelihood Estimator

$$\text{maximize}_w \quad \frac{1}{n} \sum_{i=1}^{n} \log(\mathbb{P}(y_i \mid x_i))$$

$$\text{maximize}_{w \in \mathbb{R}^d} \quad \frac{1}{n} \sum_{i=1}^{n} \log\left(\frac{1}{1 + e^{-y_i w^T x_i}} \right)$$

$$\text{maximize}_{w \in \mathbb{R}^{d \times k}} \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbf{I}\{y_i = c_j\} \log\left(\frac{e^{w[:,j]^T x_i}}{\sum_{j'=1}^{k} e^{w[:,j']^T x_i}} \right)$$

$\mathbf{I}\{y_i = j\}$ is an indicator that is one only if $y_i = j$