Linear Classification

Rina BUOY, PhD

ChatGPT 4.0

Disclaimer

Adopted from

6.390

Introduction to Machine Learning (Fall 2024)

https://introml.mit.edu/fall24

Outline

- Recap, classification setup
- Linear classifiers
  - Separator, normal vector, and separability
- Linear logistic classifiers
  - Motivation, sigmoid, and negative log-likelihood loss
- Multi-class classifiers
  - One-hot encoding, softmax, and cross-entropy

Recap:

Data $\rightarrow$

Learning algorithm

Hypothesis class
Hyperparameters
Objective (loss) functions
Regularization

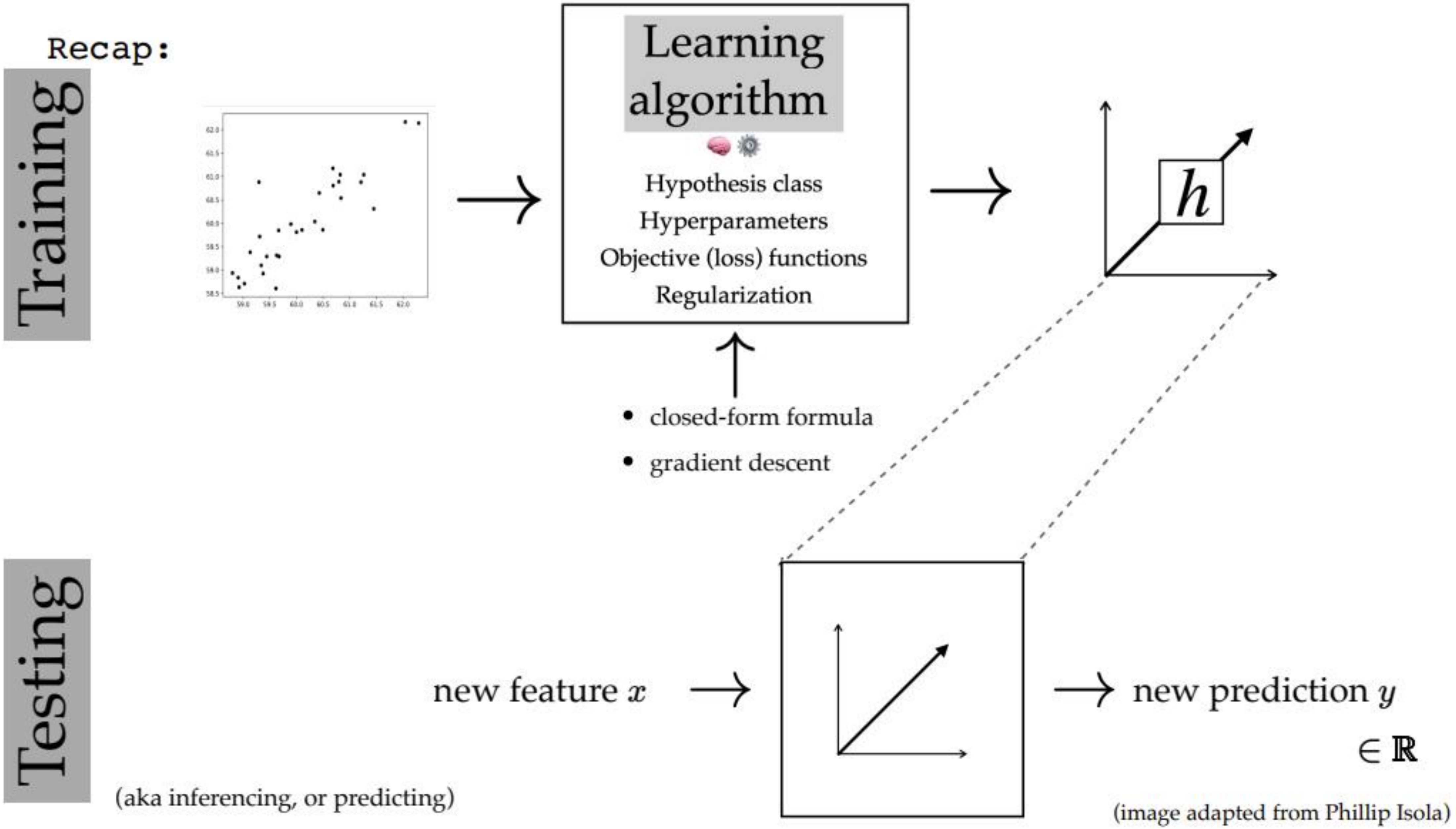
$\rightarrow$ h

$\uparrow$

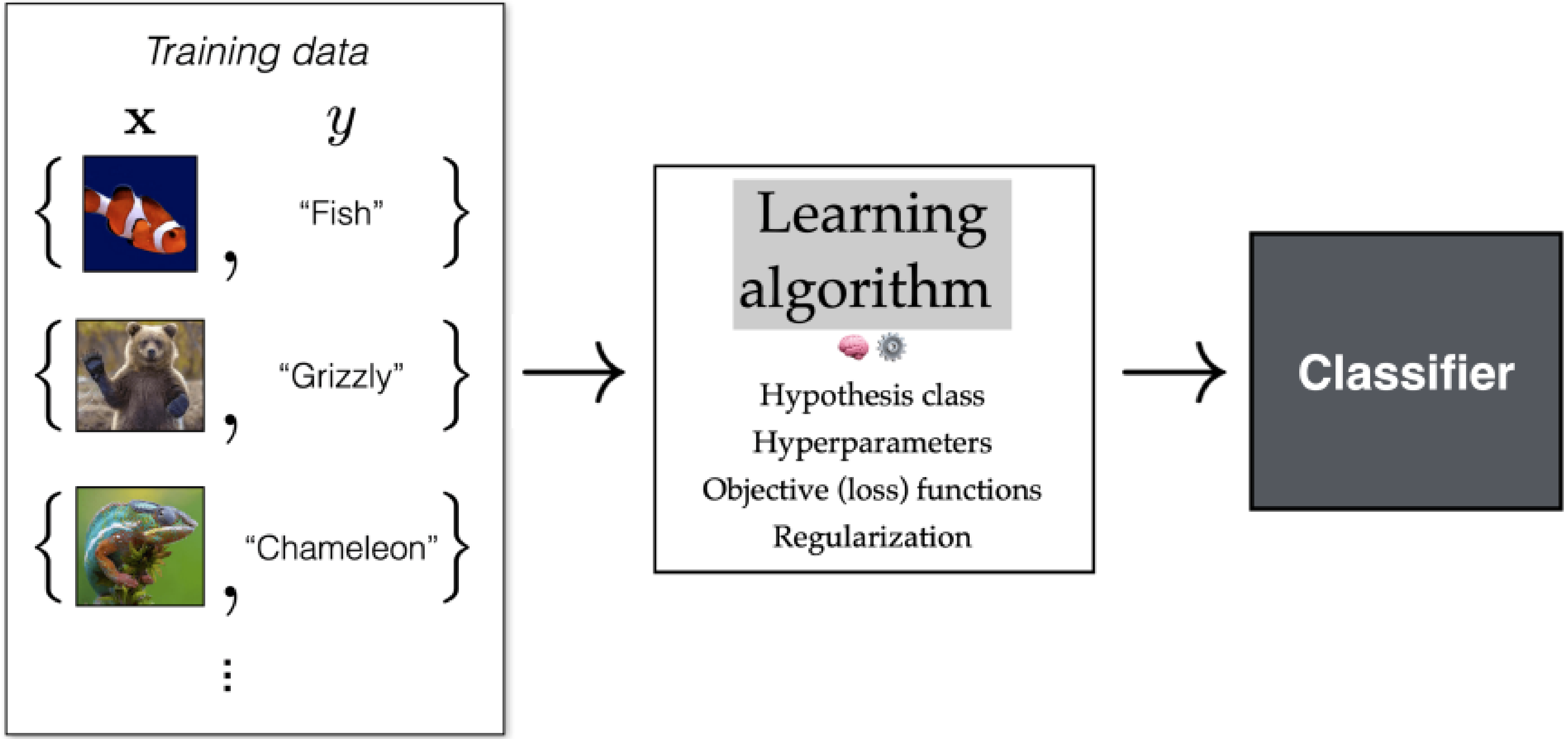
Compute/optimize/ train

(image adapted from Phillip Isola)

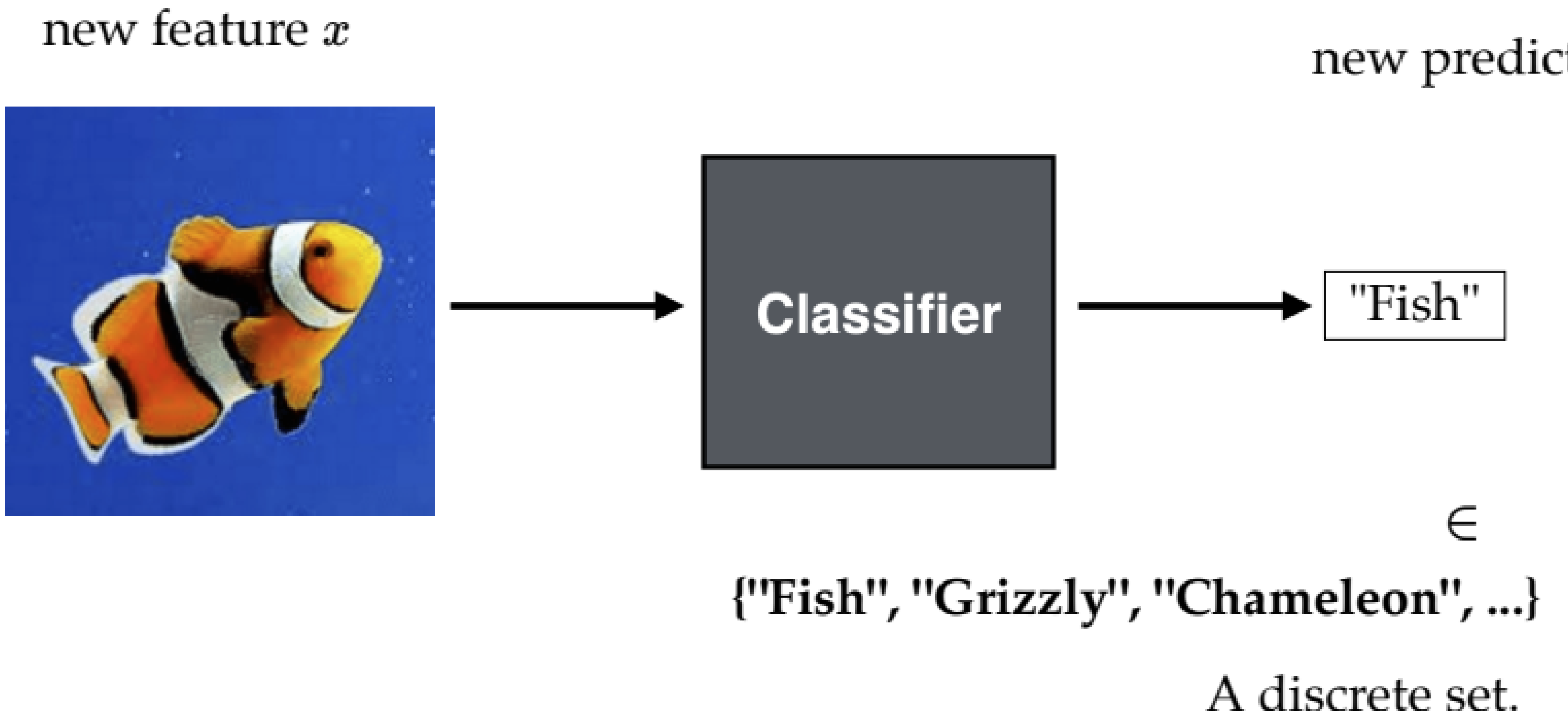
Recap:


Training
Learning algorithm
Hypothesis class
Hyperparameters
Objective (loss) functions
Regularization
h
• closed-form formula
• gradient descent
Testing
(aka inferencing, or predicting)
new feature x
new prediction y
$\in \mathbb{R}$
(image adapted from Phillip Isola)

Classification Setup

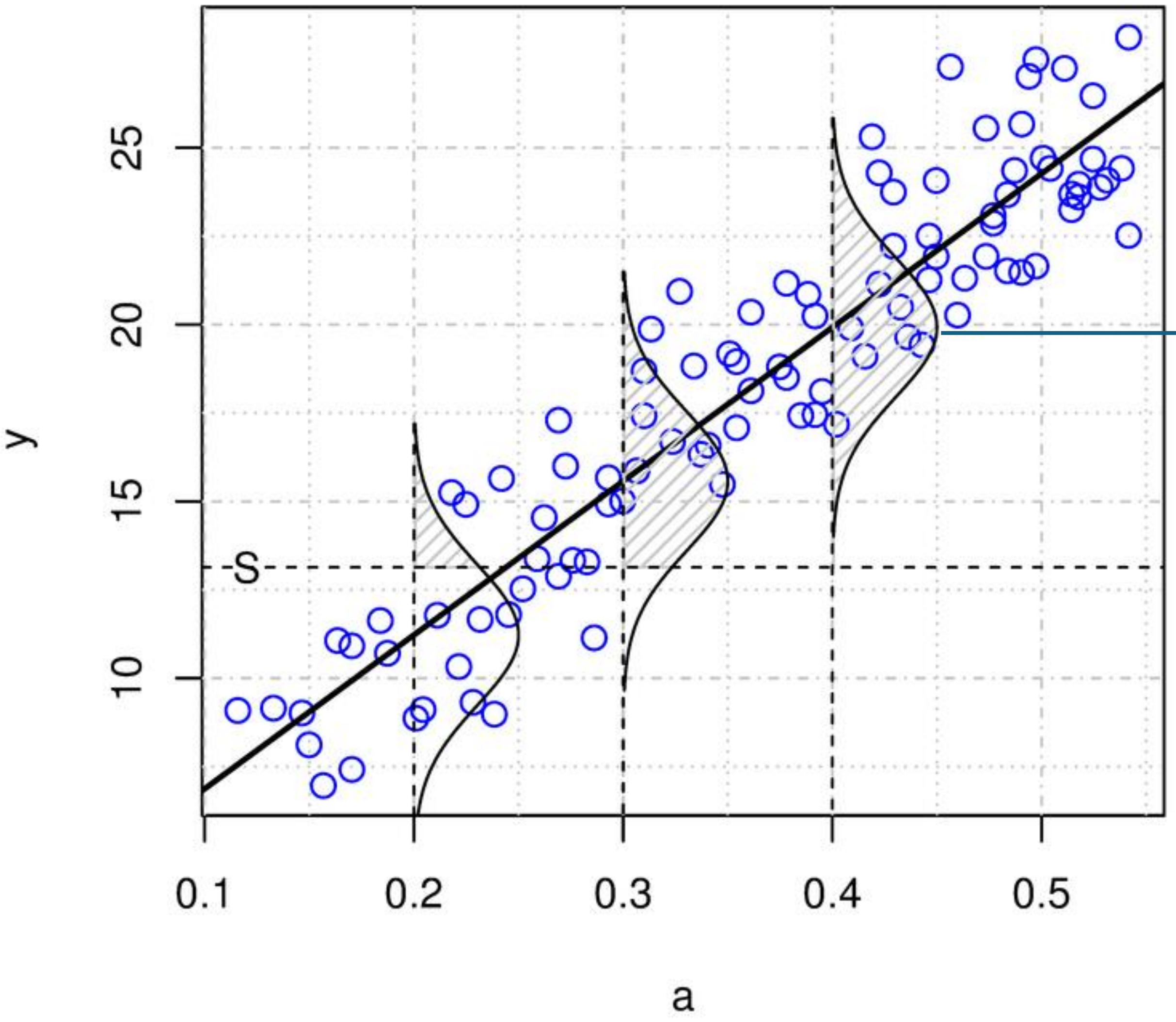


(image adapted from Phillip Isola)



(image adapted from Phillip Isola)

Linear Regression from Probability Perspective



$$p(y_i \mid \mathbf{x}_i, \theta)$$

Maximum Log-Likelihood

$$\hat{\theta} = \operatorname{argmax}_{\theta} \log p(\mathcal{D} \mid \theta)$$

https://www.quantstart.com/articles/Maximum-Likelihood-Estimation-for-Linear-Regression/

Maximum Log-Likelihood

$$\hat{\theta} = \mathrm{argmax}_{\theta} \log p(\mathcal{D} \mid \theta)$$

$$\begin{aligned} l(\theta) &:= \log p(\mathcal{D} \mid \theta) \\ &= \log \left(\prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i, \theta) \right) \\ &= \sum_{i=1}^{N} \log p(y_i \mid \mathbf{x}_i, \theta) \end{aligned}$$

Negative Log-Likelihood

$$\mathrm{NLL}(\theta) = -\sum_{i=1}^{N} \log p(y_i \mid \mathbf{x}_i, \theta)$$

Negative Log-Likelihood

$$
\begin{aligned}
\mathrm{NLL}(\theta) &= -\sum_{i=1}^{N} \log p(y_i \mid \mathbf{x}_i, \theta) \\
&= -\sum_{i=1}^{N} \log \left[\left(\frac{1}{2\pi\sigma^2} \right)^{\frac{1}{2}} \exp \left(-\frac{1}{2\sigma^2} (y_i - \beta^T \mathbf{x}_i)^2 \right) \right] \\
&= -\sum_{i=1}^{N} \frac{1}{2} \log \left(\frac{1}{2\pi\sigma^2} \right) - \frac{1}{2\sigma^2} (y_i - \beta^T \mathbf{x}_i)^2 \\
&= -\frac{N}{2} \log \left(\frac{1}{2\pi\sigma^2} \right) - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - \beta^T \mathbf{x}_i)^2 \\
&= -\frac{N}{2} \log \left(\frac{1}{2\pi\sigma^2} \right) - \frac{1}{2\sigma^2} \mathrm{RSS}(\beta)
\end{aligned}
$$

Optimal Parameters

$$\frac{\partial NLL}{\partial \beta} = 0$$

$$\hat{\beta}_{\mathrm{OLS}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

Maximum Likelihood Formulation

1. Choose a suitable probability distribution $Pr(\mathbf{y}|\boldsymbol{\theta})$ that is defined over the domain of the predictions $\mathbf{y}$ and has distribution parameters $\boldsymbol{\theta}$.
2. Set the machine learning model $\mathbf{f}[\mathbf{x}, \phi]$ to predict one or more of these parameters so $\boldsymbol{\theta} = \mathbf{f}[\mathbf{x}, \phi]$ and $Pr(\mathbf{y}|\boldsymbol{\theta}) = Pr(\mathbf{y}|\mathbf{f}[\mathbf{x}, \phi])$.
3. To train the model, find the network parameters $\hat{\boldsymbol{\phi}}$ that minimize the negative log-likelihood loss function over the training dataset pairs $\{\mathbf{x}_i, \mathbf{y}_i\}$:

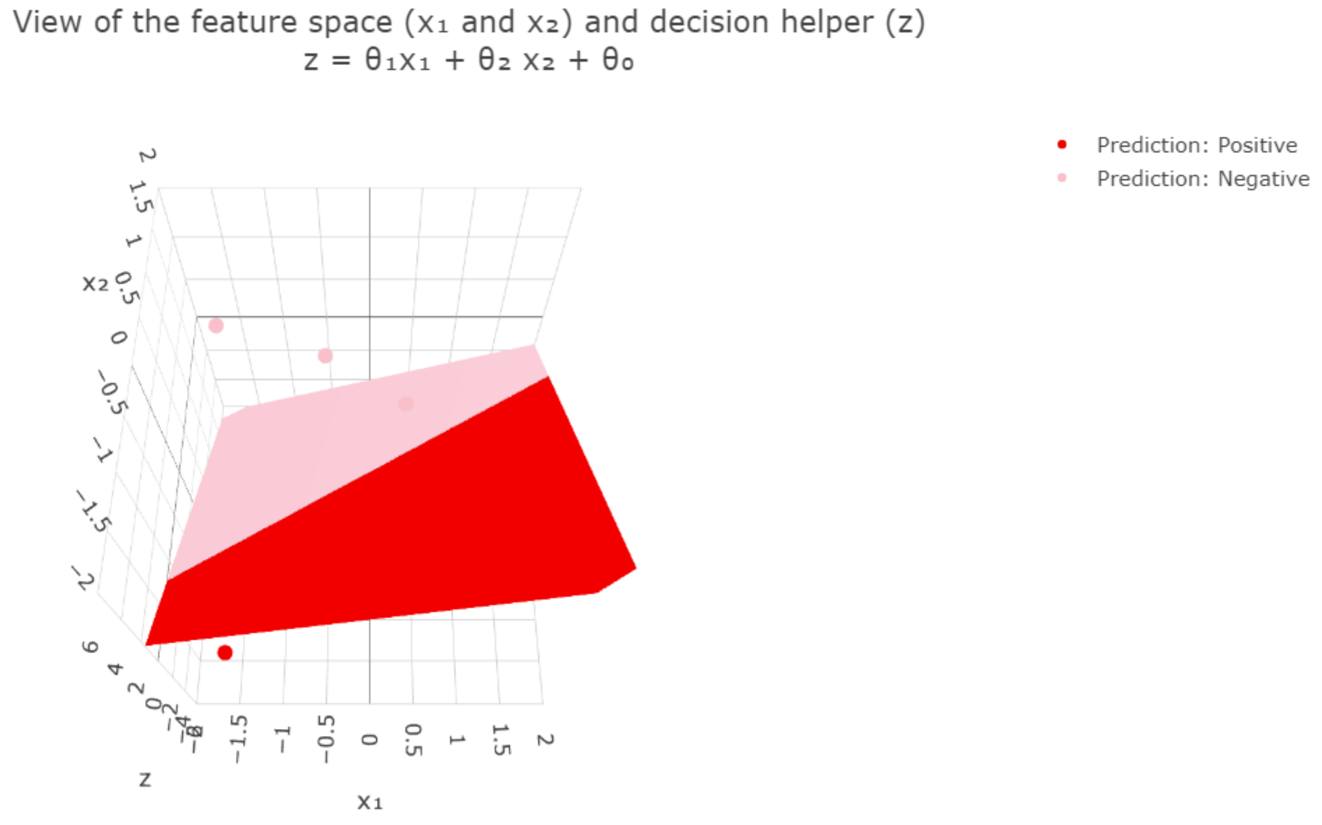
$$\hat{\boldsymbol{\phi}} = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[L[\boldsymbol{\phi}] \right] = \underset{\boldsymbol{\phi}}{\operatorname{argmin}} \left[-\sum_{i=1}^{I} \log \Big[Pr(\mathbf{y}_i | \mathbf{f}[\mathbf{x}_i, \boldsymbol{\phi}]) \Big] \right]. \tag{5.7}$$

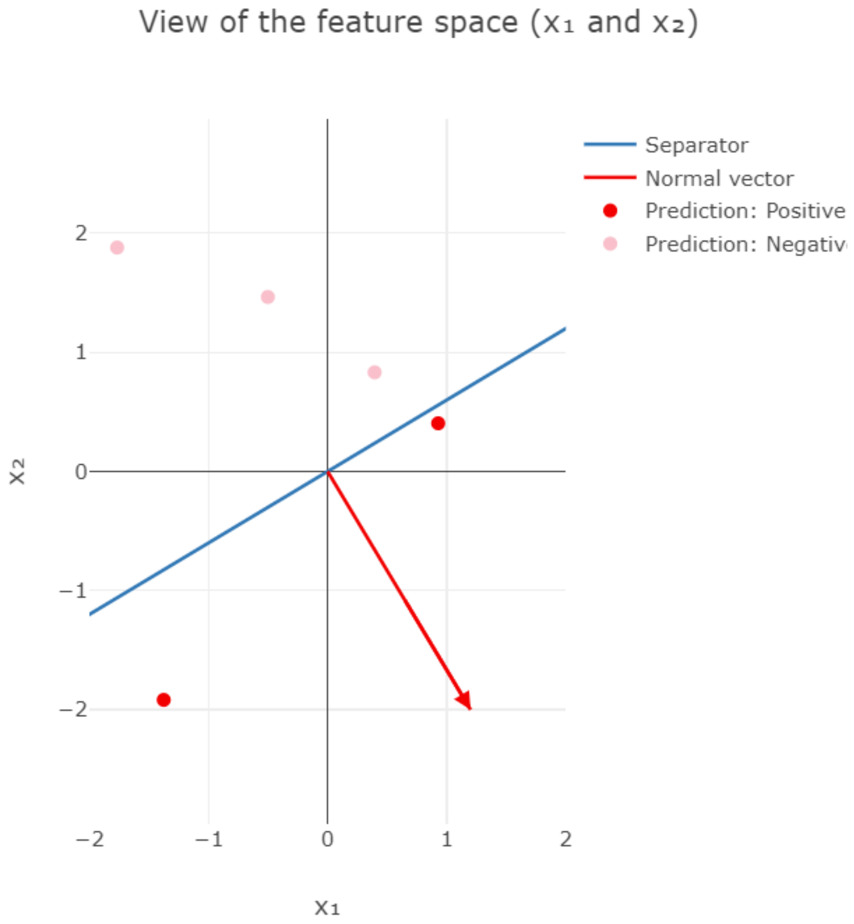
Outline

- Recap, classification setup
- Linear classifiers
  - Separator, normal vector, and separability
- Linear logistic classifiers
  - Motivation, sigmoid, and negative log-likelihood loss
- Multi-class classifiers
  - One-hot encoding, softmax, and cross-entropy

(vanilla, sign-based, binary) Linear Classifier

- Each data point:
  - features $[x_1, x_2, \ldots x_d]$
  - label $y \in$ {positive, negative} (or {dog, cat}, {pizza, not pizza}, {+1, 0})
- A (vanilla, sign-based, binary) linear classifier is parameterized by $[\theta_1, \theta_2, \ldots, \theta_d, \theta_0]$
- To ***use*** a given classifier make prediction:
  - do linear combination: $z = (\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d) + \theta_0$
  - predict positive label if $z > 0$, otherwise, negative label.





θ_1: 1.2

θ_2: -2.0

θ_0: 0.0

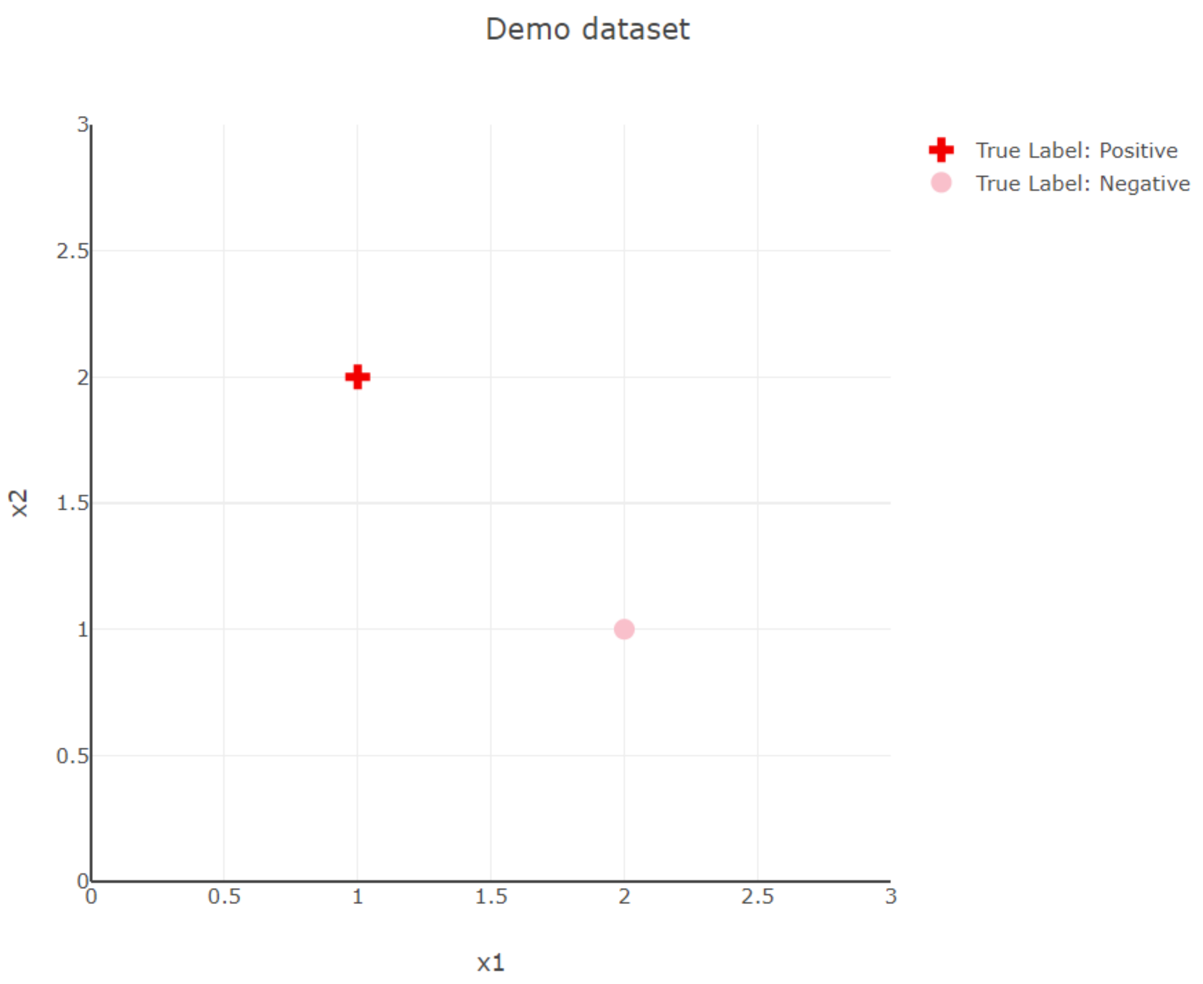
Toggle z=0 Surface

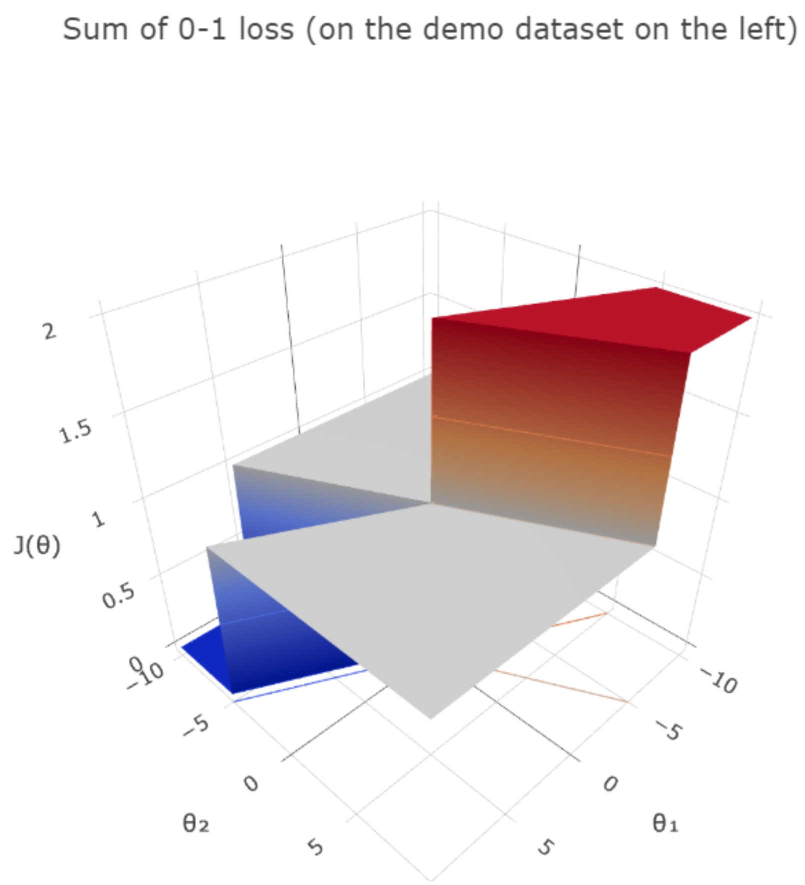
- Now let's try to *learn* a linear classifier
- One natural loss: $$\mathcal{L}_{01}(g, a) = \begin{cases} 0 & \text{if guess} = \text{actual} \\ 1 & \text{otherwise} \end{cases}$$
- Combined with the linear classifier hypothesis:

$$\mathcal{L}_{01}(x^{(i)}, y^{(i)}; \theta, \theta_0) = \begin{cases} 0 & \text{if sign}\left(\theta^\top x^{(i)} + \theta_0\right) = y^{(i)} \\ 1 & \text{otherwise} \end{cases}$$

- Very intuitive, and easy to evaluate 😍
  - Induced concept: separability
- Very hard to optimize (NP-hard) 🥺
  - "Flat" almost everywhere (zero gradient)
  - "Jumps" elsewhere (no gradient)

Adopted from : https://introml.mit.edu





Try to draw the separator and normal vector given by ($\theta_1 = -1$, and $\theta_2 = 1$) on the 2D plot, and make sense of the loss given in the 3D plot.

Outline

- Recap, classification setup
- Linear classifiers
  - Separator, normal vector, and separability
- Linear logistic classifiers
  - Motivation, sigmoid, and negative log-likelihood loss
- Multi-class classifiers
  - One-hot encoding, softmax, and cross-entropy

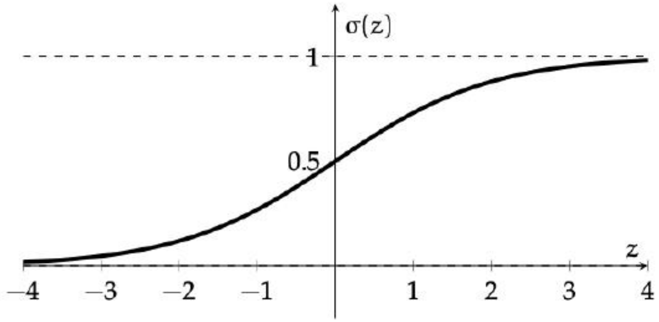
Linear Logistic Classifier

- Mainly motivated to address the gradient issue in *learning* a "vanilla" linear classifier
  - The gradient issue is caused by both the $0/1$ loss, and the sign functions nested in.

$$\mathcal{L}_{01}(x^{(i)}, y^{(i)}; \theta, \theta_0) = \begin{cases} 0 & \text{if sign}\left(\theta^\top x^{(i)} + \theta_0\right) = y^{(i)} \\ 1 & \text{otherwise} \end{cases}$$

- But has nice probabilistic interpretation too.
- As before, let's first look at how to make prediction with a *given* linear logistic classifier

(Binary) Linear Logistic Classifier



- Each data point:
  - features $[x_1, x_2, \ldots x_d]$
  - label $y \in$ {positive, negative}
- A (binary) linear **logistic** classifier is parameterized by $[\theta_1, \theta_2, \ldots, \theta_d, \theta_0]$
- To ***use*** a given classifier make prediction:
  - do linear combination: $z = (\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d) + \theta_0$
  - predict positive label if

$$\sigma(z) = \sigma\left(\theta^\top x + \theta_0\right) = \frac{1}{1+e^{-z}} = \frac{1}{1+e^{-(\theta^\top x + \theta_0)}} > 0.5$$

otherwise, negative label.

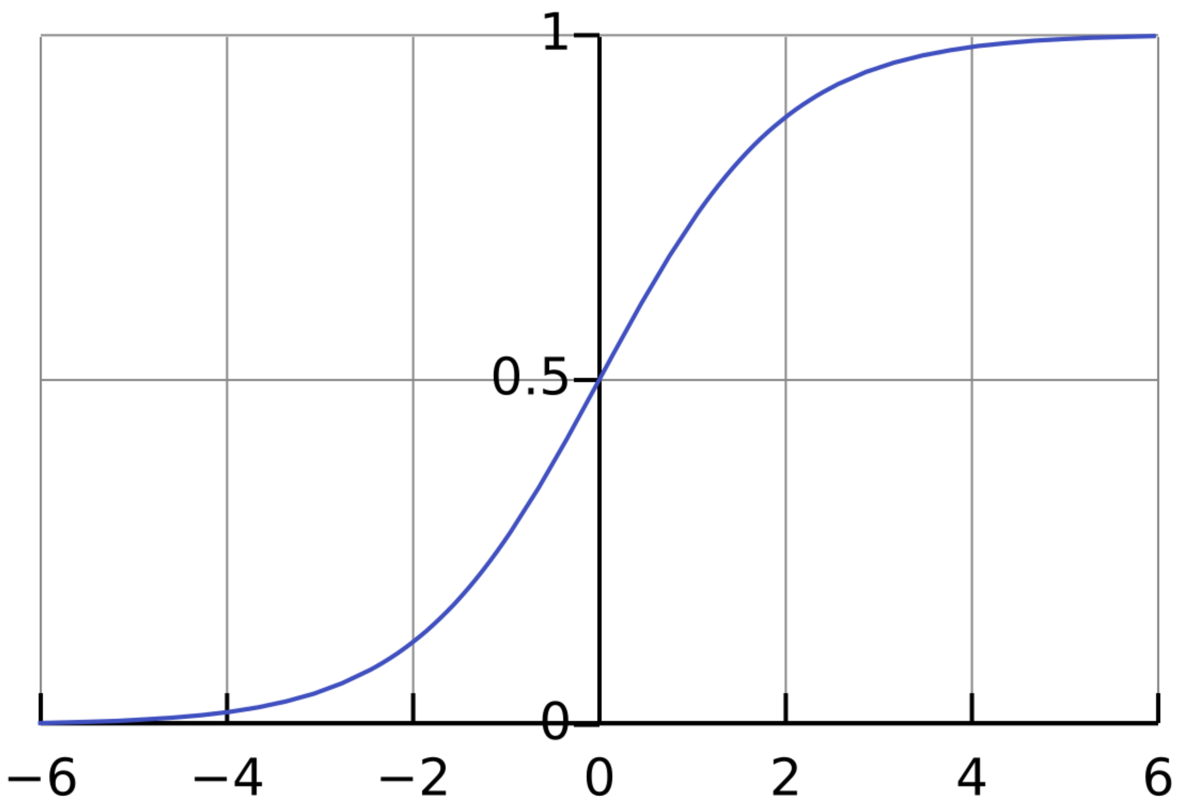
Sigmoid : a smooth step function

$$\sigma(z) = \sigma\left(\theta^\top x + \theta_0\right) = \frac{1}{1 + e^{-(\theta^\top x + \theta_0)}}$$

- "sandwiched" between 0 and 1 *vertically* (never 0 or 1 mathematically)
- θ, θ_0 can flip, squeeze, expand, shift *horizontally*
- $\sigma(\cdot)$ interpreted as the *probability / confidence* that feature x has positive label. Predict positive if

$$\sigma(z) = \sigma\left(\theta^\top x + \theta_0\right) > 0.5$$

- monotonic, very nice / elegant gradient (see recitation / hw)



Adopted from : https://introml.mit.edu

$$\sigma(z) = \sigma\left(\theta^{\top} x + \theta_0\right) = \frac{1}{1 + e^{-(\theta^{\top} x + \theta_0)}}$$ ➡ **Probability**

$$\theta^{\top} x + \theta_0$$ ➡ **Logit or log odd** ➡ $\log(\frac{p}{1-p})$

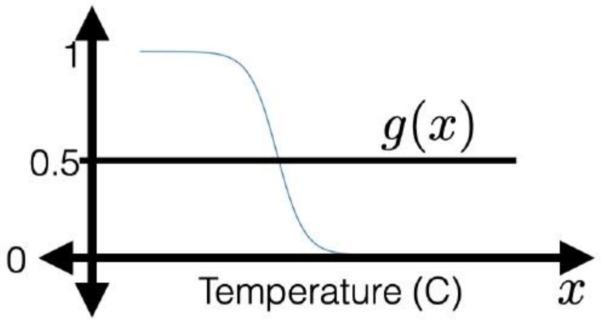
$$\frac{p}{1-p}$$ ➡ **Odd ratio**

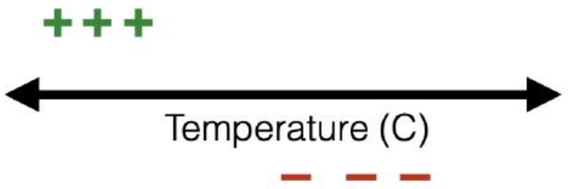
If the probability of rain is 0.5, the odd ratio of rain to no rain is 1 and the log odd is 0.

e.g. suppose, wanna predict whether to bike to school.

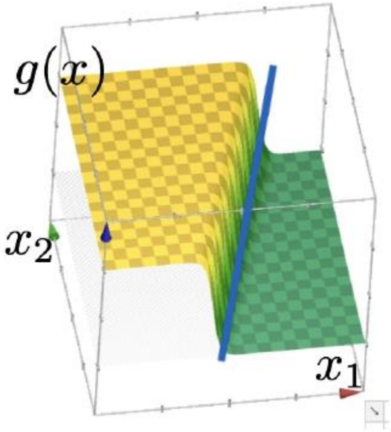
with **given** parameters, how do I make prediction?

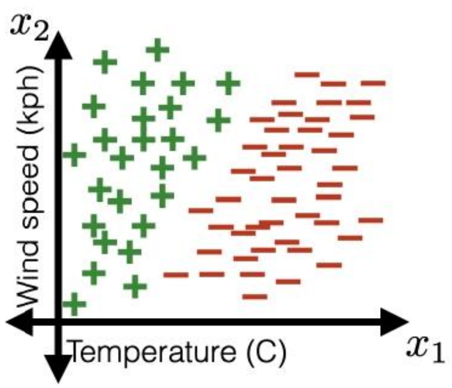
1 feature: $$g(x) = \sigma\left(\theta x + \theta_0\right) = \frac{1}{1 + \exp\left\{-\left(\theta x + \theta_0\right)\right\}}$$





2 features: $$g(x) = \sigma\left(\theta^\top x + \theta_0\right) = \frac{1}{1 + \exp\left\{-\left(\theta^\top x + \theta_0\right)\right\}}$$

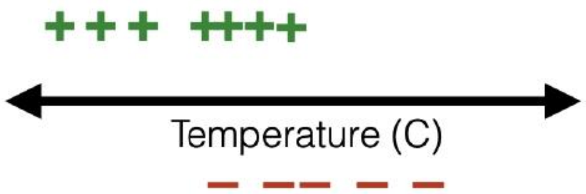




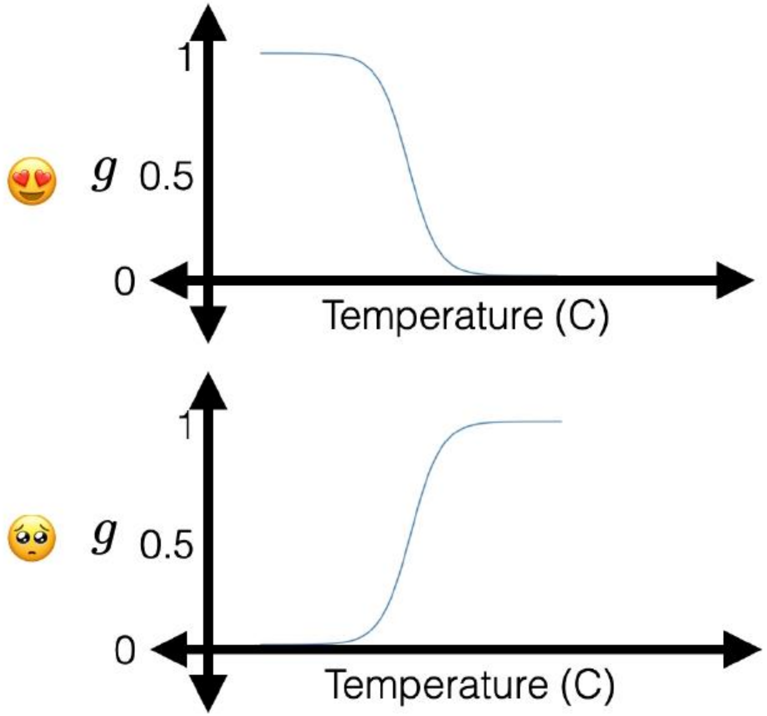
(image credit: Tamara Broderick)

Learning a logistic regression classifier

training data:



$$g(x) = \sigma(\theta x + \theta_0)$$

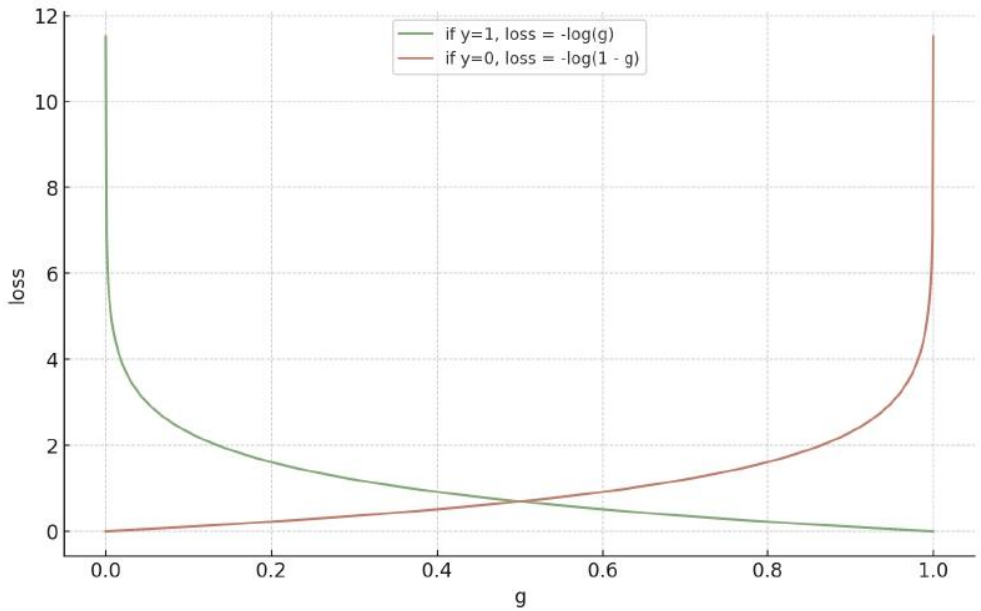


- Let the labels $\boldsymbol{y} \in \{\mathbf{+1, 0}\}$

$$\mathcal{L}_{\text{nll}}(\text{ guess, actual })$$

$$= -[\text{ actual } \cdot \log(\text{ guess }) + (1 - \text{ actual }) \cdot \log(1 - \text{ guess })]$$

$$= -\left[y^{(i)} \log g^{(i)} + \left(1 - y^{(i)}\right) \log\left(1 - g^{(i)}\right)\right]$$

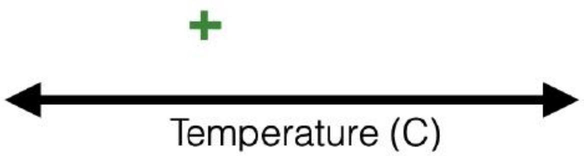


If $y^{(i)} = 1$

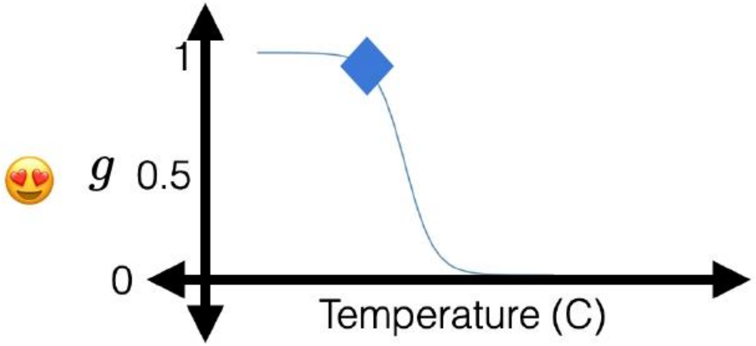
$$\mathcal{L}_{\text{nll}} \text{ (guess, actual)}$$

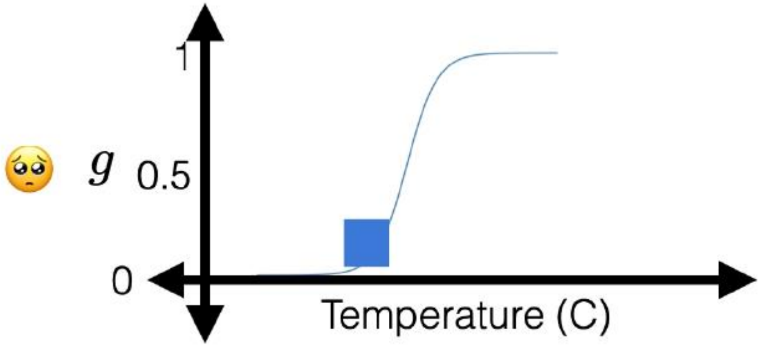
$$= -\left[\cancel{y^{(i)}} \log g^{(i)} + \cancel{\left(1 - y^{(i)}\right) \log \left(1 - g^{(i)}\right)}\right]$$

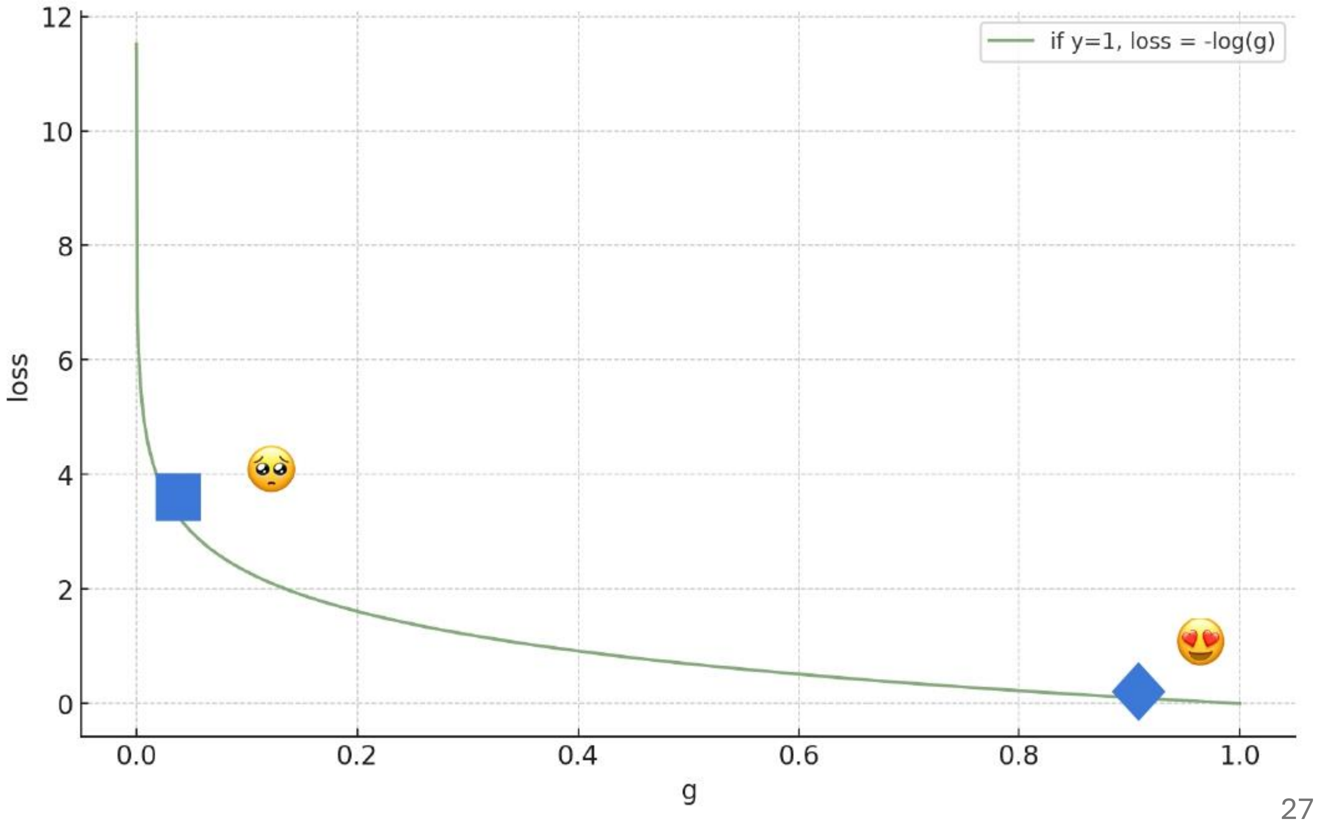
training data:



$$g(x) = \sigma\left(\theta x + \theta_0\right)$$







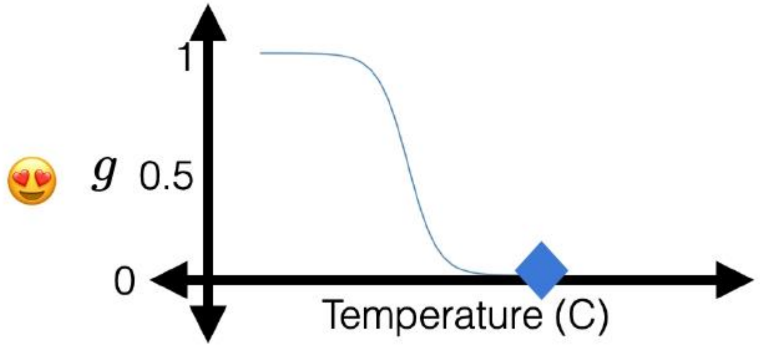
If $y^{(i)} = 0$

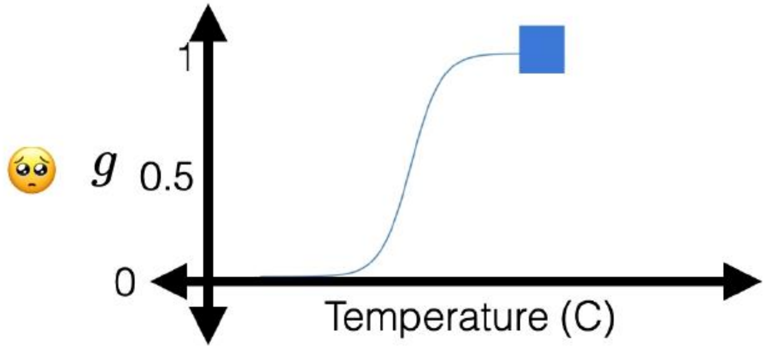
$\mathcal{L}_{\text{nll}}$ (guess, actual)

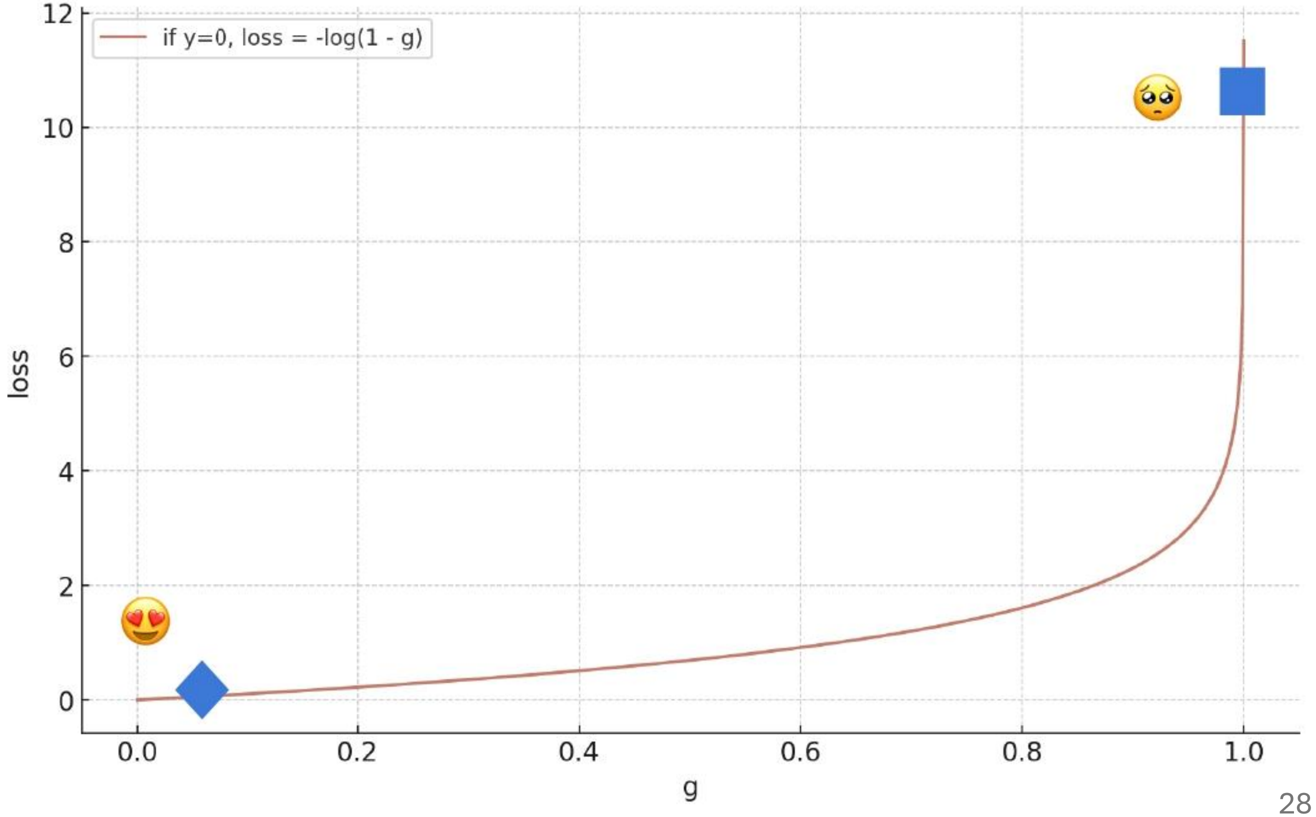
$$= -\left[\cancel{y^{(i)} \log g^{(i)} +} \cancel{\left(1 - y^{(i)}\right)} \log\left(1 - g^{(i)}\right)\right]$$

training data:

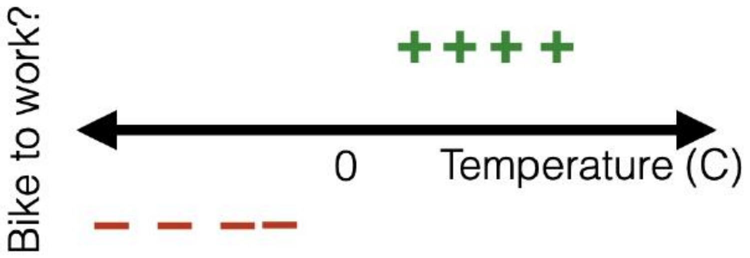
$$g(x) = \sigma\left(\theta x + \theta_0\right)$$







Logistic Regression

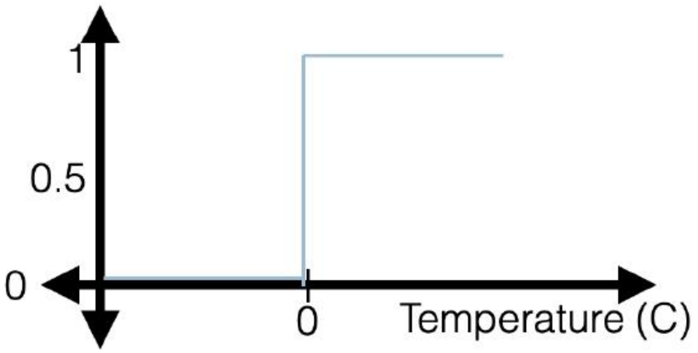


- Minimize using negative-log-likelihood loss:

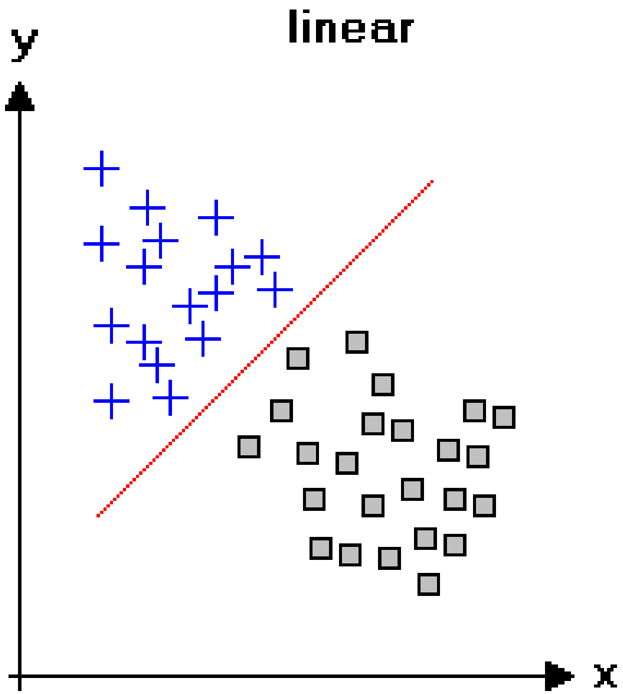
  $J_{lr} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{\mathrm{nll}} \left(\sigma \left(\theta^{\top} x^{(i)} + \theta_0 \right), y^{(i)} \right)$

- Convex, differentiable with **nice** (elegant) gradients
- Doesn't have a closed-form solution
- Can still run gradient descent
- But, a gotcha: when training data is linearly separable

$$g(x) = \sigma \left(\theta^T x + \theta_0 \right)$$



Adopted from : https://introml.mit.edu

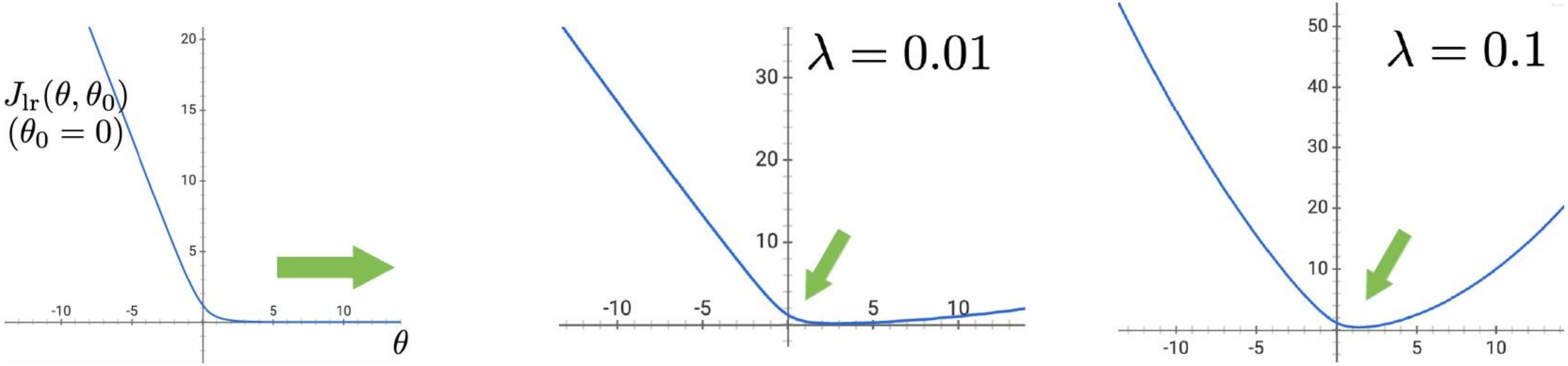


http://www.statistics4u.com/fundstat_eng/cc_data_structure.html

Regularized Logistic Regression

$$\mathrm{J}\left(\theta, \theta_0\right) = \left(\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_{\mathrm{nll}}\left(\sigma\left(\theta^{\top} x^{(i)} + \theta_0\right), y^{(i)}\right)\right) + \lambda\|\theta\|^2$$

- $\lambda \geq 0$
- No regularizing θ_0 (think: why?)
- Penalizes being overly certain
- Objective is still differentiable and convex (gradient descent)



Adopted from : https://introml.mit.edu

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N}\Big(y_i - \sum_{j=0}^{M} x_{ij}W_j\Big)^2 + \lambda\sum_{j=0}^{M}|W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N}\Big(y_i - \sum_{j=0}^{M} x_{ij}W_j\Big)^2}_{\text{Loss function}} + \underbrace{\lambda\sum_{j=0}^{M}W_j^2}_{\text{Regularization Term}}$$

https://medium.com/analytics-vidhya/l1-vs-l2-regularization-which-is-better-d01068e6658c

Outline

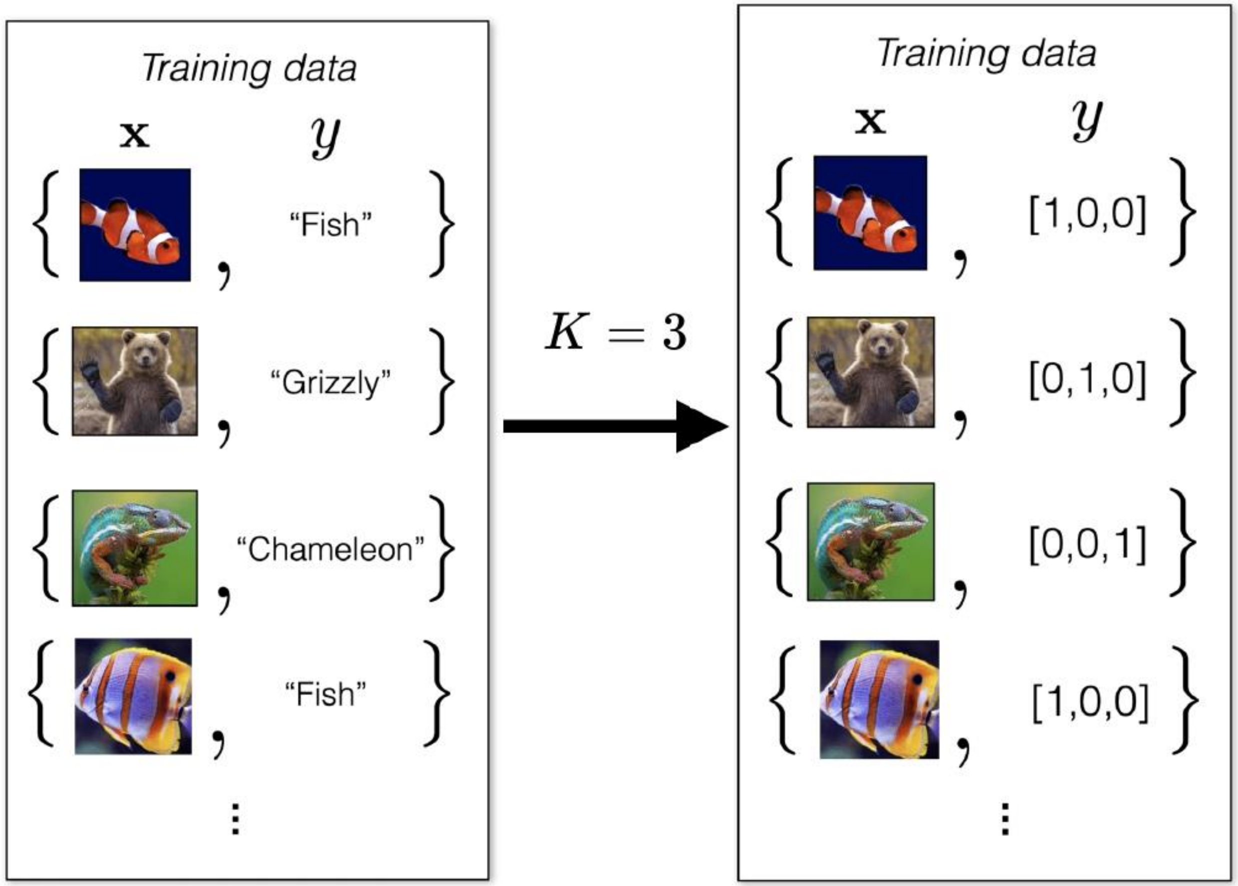
- Recap, classification setup
- Linear classifiers
  - Separator, normal vector, and separability
- Linear logistic classifiers
  - Motivation, sigmoid, and negative log-likelihood loss
- Multi-class classifiers
  - One-hot encoding, softmax, and cross-entropy

One-hot labels

- Generalizes from binary labels
- Suppose K classes

$\in \mathbb{R}^{K}$

One-hot encoding



(image adapted from Phillip Isola)

Softmax

- Generalizes sigmoid
- Applied on z element-wise

Two classes

$$z = \theta^\top x + \theta_0$$

↑

scalar

$$g = \sigma(z) = \frac{1}{1 + \exp(-z)}$$

↑

scalar

K classes

$$z = \theta^\top x + \theta_0$$

↑

K-by-1 Vector

$$g = \mathrm{softmax}(z) = \begin{bmatrix} \exp(z_1) / \sum_i \exp(z_i) \\ \vdots \\ \exp(z_K) / \sum_i \exp(z_i) \end{bmatrix}$$

↑

K-by-1 Vector

Negative log-likelihood multi-class loss

- Generalizes negative log likelihood loss
- Also known as cross-entropy

Two classes

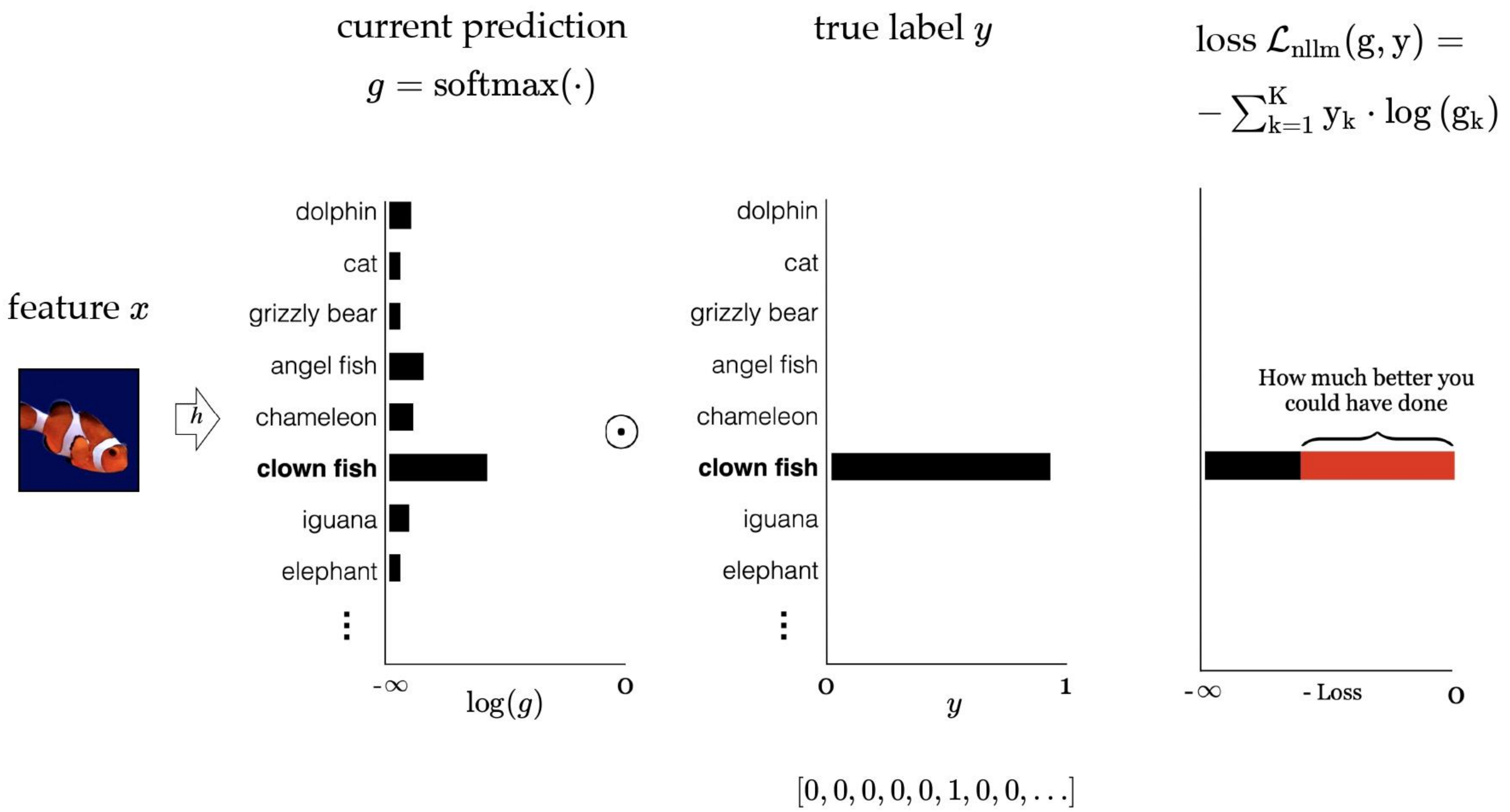
$$\mathcal{L}_{\mathrm{nll}}(\mathrm{g}, \mathrm{y}) = -\left(y \log g + (1-y) \log (1-g)\right)$$

- Appears as sum of two terms
- Only one term "activates" for a single data point

K classes

$$\mathcal{L}_{\mathrm{nllm}}(\mathrm{g}, \mathrm{y}) = -\sum_{\mathrm{k}=1}^{\mathrm{K}} \mathrm{y}_{\mathrm{k}} \cdot \log\left(\mathrm{g}_{\mathrm{k}}\right)$$

- Appears as sum of K terms
- Only one term "activates" for a single data point



(image adapted from Phillip Isola)

Adopted from : https://introml.mit.edu

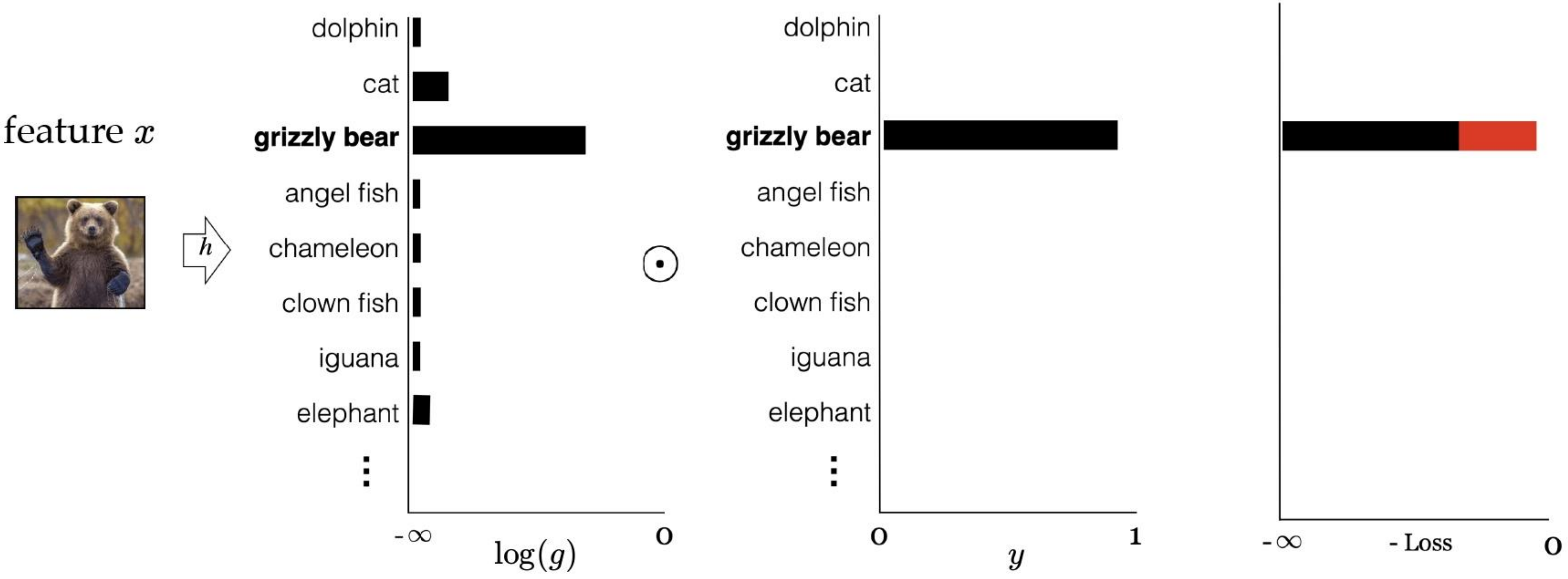
current prediction

$g = \text{softmax}(\cdot)$

true label y

loss $\mathcal{L}_{\text{nllm}}(\text{g}, \text{y}) =$

$-\sum_{\text{k}=1}^{\text{K}} \text{y}_{\text{k}} \cdot \log(\text{g}_{\text{k}})$

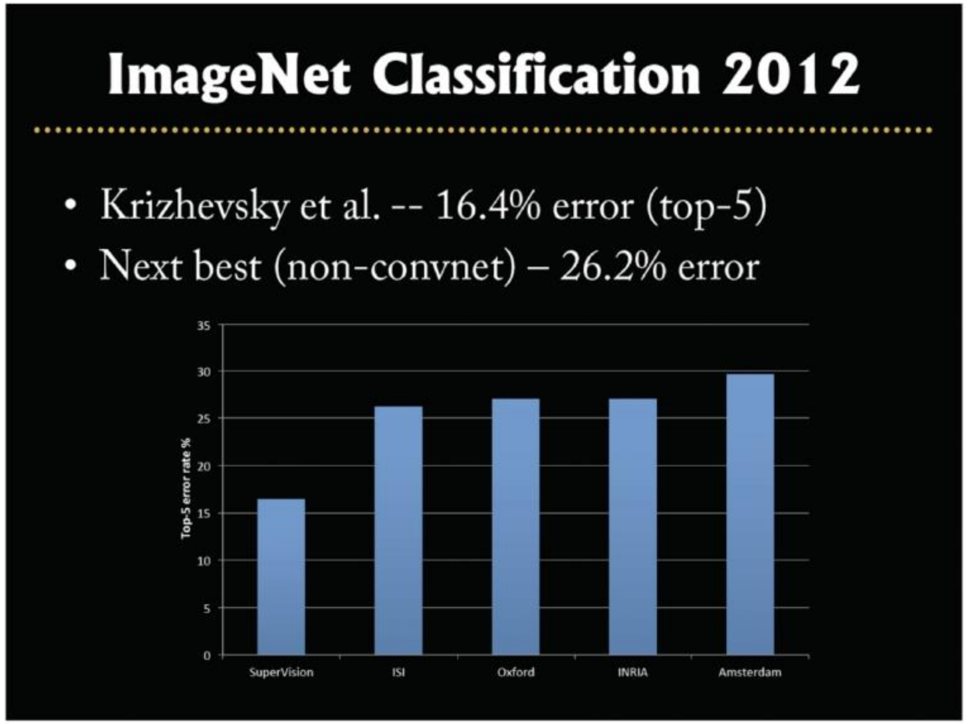


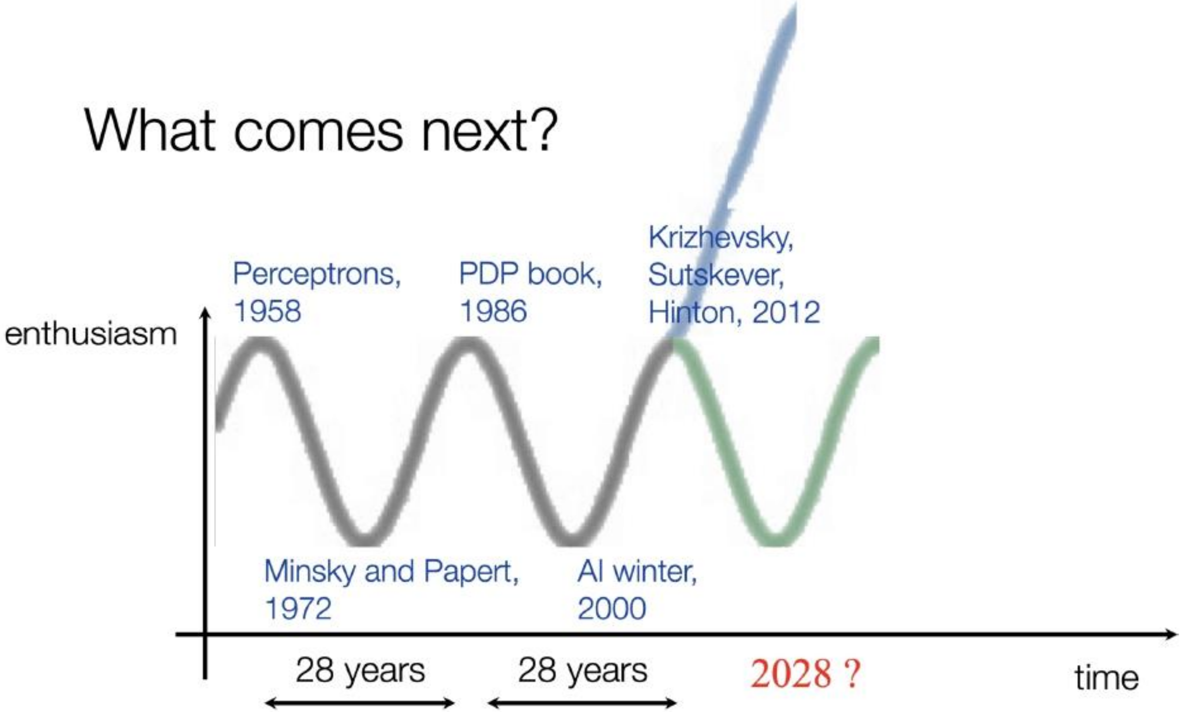
$[0, 0, 1, 0, 0, 0, 0, 0, \ldots]$

(image adapted from Phillip Isola)

Classification

Image classification played a pivotal role in kicking off the current wave of AI enthusiasm





Summary

- Classification: a supervised learning problem, similar to regression, but where the output/label is in a discrete set.
- Binary classification: only two possible label values.
- Linear binary classification: think of θ and θ_0 as defining a d-1 dimensional hyperplane that **cuts** the d-dimensional feature space into two half-spaces.
- 0-1 loss: a natural loss function for classification, BUT, hard to optimize.
- Sigmoid function: motivation and properties.
- Negative-log-likelihood loss: smoother and has nice probabilistic motivations. We can optimize via (S)GD.
- Regularization is still important.
- The generalization to multi-class via (one-hot encoding, and softmax mechanism)
- Other ways to generalize to multi-class (see hw/lab)