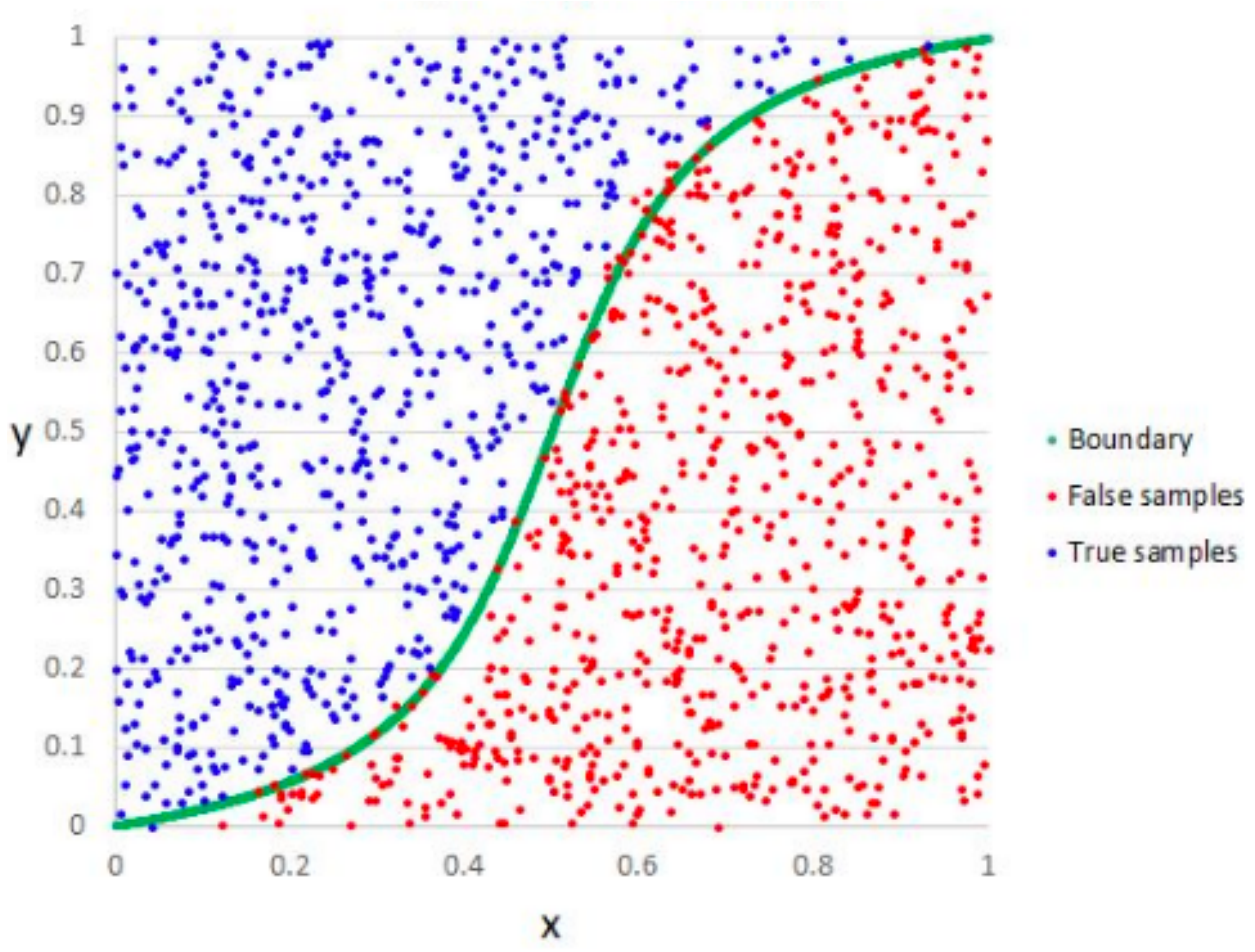
Logistic Regression

Logistic Regression



- Powerful supervised model
- Baseline approach for many NLP tasks
- Connections with neural networks
- Binary (two classes) or multinomial (>2 classes)

Discriminative Model

- Logistic Regression is a *discriminative* model
- Naive Bayes: *generative* model

Discriminative Model

- Logistic Regression: $\hat{c} = \underset{c}{\text{argmax}}\ P(c|d)$
- Naive Bayes: $\hat{c} = \underset{c}{\text{argmax}}\ P(c)\,P(d|c)$

Cat: a domesticated carnivorous mammal with soft fur, a $\boxed{\text{short snout}}$, and $\boxed{\text{retractable}}$ claws.

Dog: a domesticated carnivorous mammal with a $\boxed{\text{long snout}}$, $\boxed{\text{nonretractable}}$ claws, and a barking, howling, or whining voice.

Overview

- Inputs:
  1. Classification instance in a **feature representation**
  2. **Classification function** to compute $\hat{y}$ using $P(\hat{y} \mid x)$
  3. **Loss function** (for learning)
  4. Optimization **algorithm**
- Train phase: Learn the **parameters** of the model to minimize **loss function**
- Test phase: Apply **parameters** to predict class given a new input x

1. Feature representation

- Input observation: $x^{(i)}$
- Feature vector: $[x_1, x_2, \ldots, x_d] = \mathbf{x}$
- Feature j of ith input : $x_j^{(i)}$

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

$x^{(i)}$

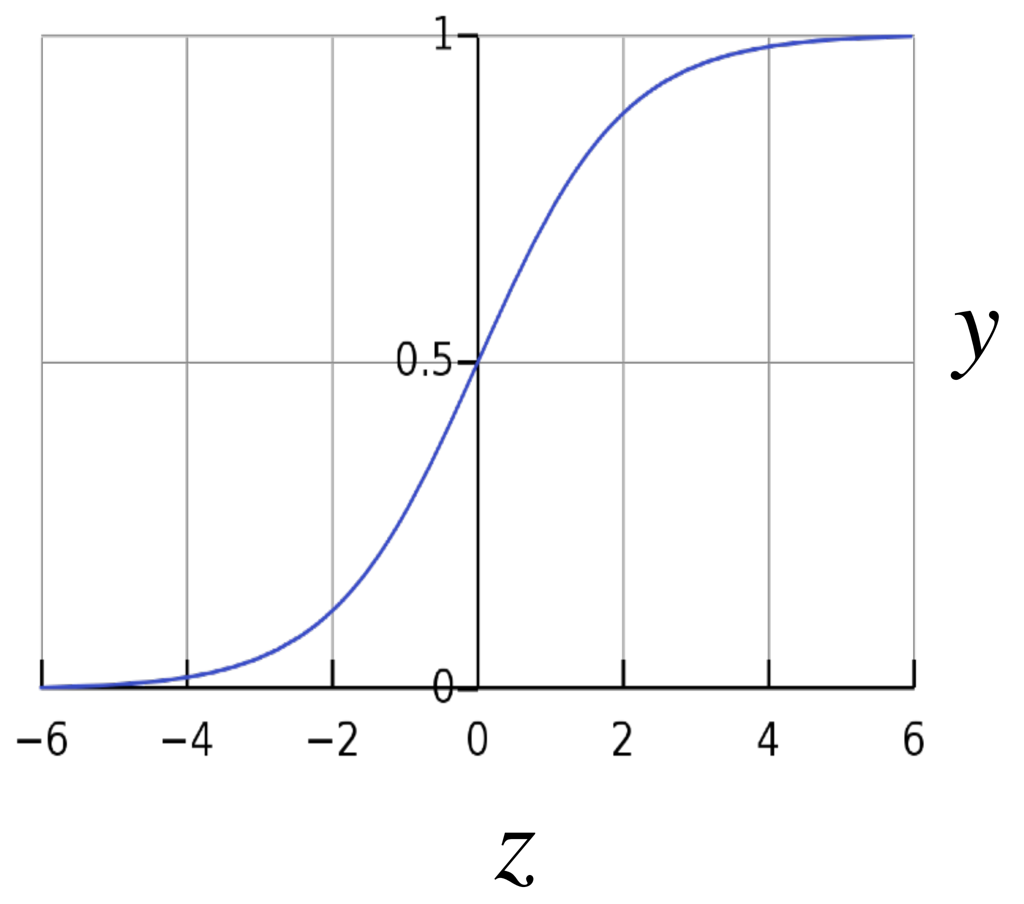
it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1

$[x_1, x_2, \ldots, x_d]$

Bag of words

2. Classification function

- *Given*: Input feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_d]$
- *Output*: $P(y = 1 \,|\, \mathbf{x})$ and $P(y = 0 \,|\, \mathbf{x})$
  (binary classification)
- Require a *function*, $F : \mathbb{R}^{\mathrm{d}} \to [0,1]$
- Sigmoid:



$$f(z) = \frac{1}{1 + e^{-z}} = \frac{e^{z}}{1 + e^{z}}$$

Weights and Biases

- *Which features are important* and *how much*?
- Learn a vector of **weights** and a **bias**
- Weights: Vector of real numbers, $\mathbf{w} = [w_1, w_2, \ldots, w_d]$
- Bias: Scalar intercept, b
- Given input features $\mathbf{x}$, : $z = \mathbf{w} \cdot \mathbf{x} + b$
- Therefore, $f(\mathbf{w} \cdot \mathbf{x} + b) = \dfrac{e^{\mathbf{w} \cdot \mathbf{x} + b}}{1 + e^{\mathbf{w} \cdot \mathbf{x} + b}}$

Putting it together

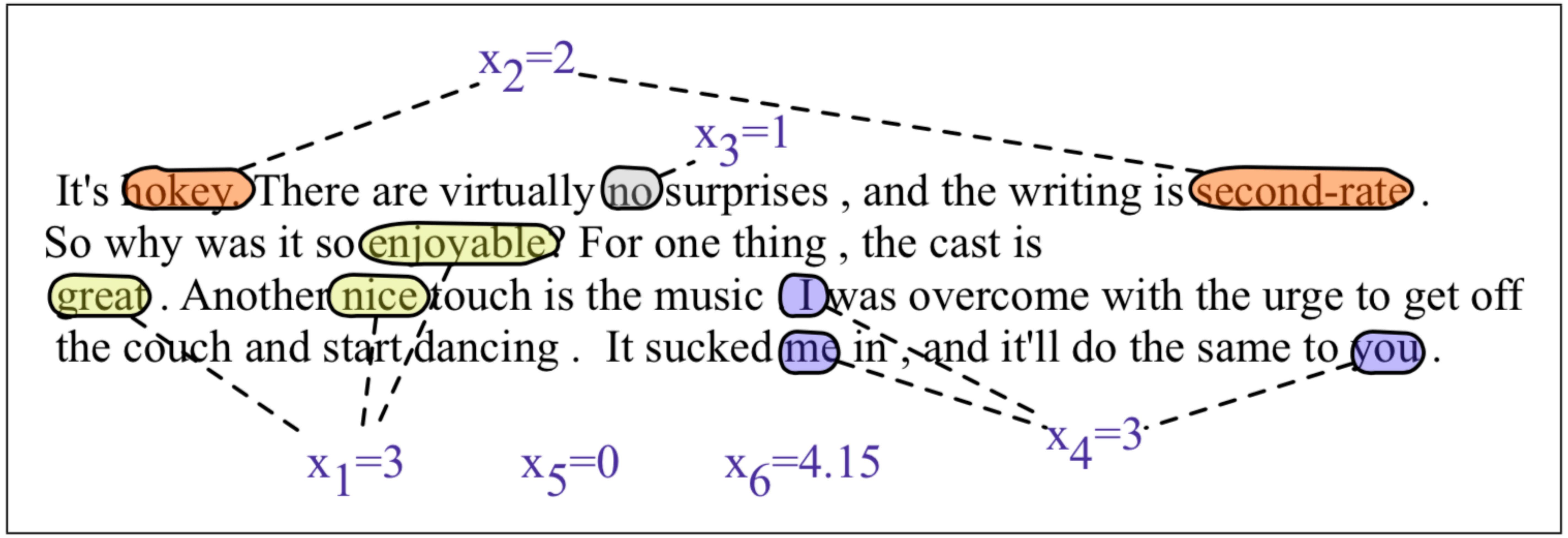
- Compute probabilities: $P(y = 1 \mid \mathbf{x}) = \dfrac{1}{1 + e^{-z}}$

$$P(y = 1) = \sigma(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\begin{aligned} P(y = 0) &= 1 - \sigma(\mathbf{w} \cdot \mathbf{x} + b) \\ &= 1 - \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} = \frac{e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}} \end{aligned}$$

- Decision boundary: $\hat{y} = \begin{cases} 1 & \text{if } P(y = 1 \mid \mathbf{x}) > 0.5 \\ 0 & \text{otherwise} \end{cases}$

Example: Sentiment classification



Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

Remember that the values make up the feature vector!

Example: Sentiment classification

Var	Definition	Value
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

- Assume weights $\mathbf{w} = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$ and bias $b = 0.1$

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(.805) \\
&= 0.69 \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Designing features

- Most important rule: Data is *key*!
- Linguistic intuition (e.g. part of speech tags, parse trees)
- Complex combinations

$$x_1 = \begin{cases} 1 & \text{if “}\mathit{Case}(w_i) = \text{Lower”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_2 = \begin{cases} 1 & \text{if “}w_i \in \text{AcronymDict”} \\ 0 & \text{otherwise} \end{cases}$$

$$x_3 = \begin{cases} 1 & \text{if “}w_i = \text{St. \& }\mathit{Case}(w_{i-1}) = \text{Cap”} \\ 0 & \text{otherwise} \end{cases}$$

- Feature templates
  - Sparse representations, hash only seen features into index
  - Ex. Trigram(*logistic regression classifier*) = Feature #78
- Advanced: Representation learning (we will see this later!)

Logistic Regression: what's good and what's not

- More freedom in designing features
  - No strong independence assumptions like Naive Bayes
  - More robust to correlated features ("San Francisco" vs "Boston")
    —LR is likely to work better than NB
  - Can even have the same feature twice! (*why?*)
- May not work well on small datasets (compared to Naive Bayes)
- Interpreting learned weights can be challenging

3. Learning

- We have our **classification function -** how to assign weights and bias?
- Goal: prediction $\hat{y}$ as close as possible to actual label y
  - **Distance metric/Loss function** between predicted $\hat{y}$ and true y : $L(\hat{y}, y)$
  - **Optimization algorithm** for updating weights

Loss function

- Assume $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + b)$
- $L(\hat{y}, y) =$ Measure of difference between $\hat{y}$ and y. But what form?
- Maximum likelihood estimation (conditional):
  - Choose w and b such that $\log P(y \mid x)$ is maximized for true labels y paired with input x
  - Similar to language models!
    - where we chose parameters to maximize $\log P(w_t \mid w_{t-n}, \ldots, w_{t-1})$ given a corpus

Cross Entropy loss for a single instance

- Assume a single data point (x, y) and two possible classes to choose from
- **Classifier probability:** $P(y \mid x) = \hat{y}^{\,y}(1-\hat{y})^{1-y}$ (compact notation)
- **Log probability:** $\log P(y \mid x) = \log[\hat{y}^{y}(1-\hat{y})^{1-y}]$
  $= y \log \hat{y} + (1-y)\log(1-\hat{y})$ (maximize this)
- **Loss:** $-\log P(y \mid x) = -\,[y \log \hat{y} + (1-y)\log(1-\hat{y})]$ (minimize this)
  - $y = 1 \implies -\log \hat{y}$, and $y = 0 \implies -\log(1-\hat{y})$

Cross Entropy loss

- For n data points $(x^{(i)}, y^{(i)})$,
- **Classifier probability:** $\Pi_{i=1}^{n} P(y \mid x) = \Pi_{i=1}^{n} \hat{y}^{y}(1 - \hat{y})^{1-y}$
- **Loss:** $-\log \prod_{i=1}^{n} P(y \mid x) = -\sum_{i=1}^{n} \log P(y \mid x)$

$$L_{CE} = -\sum_{i=1}^{n} [y \log \hat{y} + (1 - y)\log(1 - \hat{y})]$$

Example: Computing CE Loss

Var	Definition	Value
x_1	count(positive lexicon) $\in$ doc)	3
x_2	count(negative lexicon) $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if “no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if “!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(64) = 4.15$

- Assume weights $w = [2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$ and bias $b = 0.1$
- If y = 1 (positive sentiment), $L_{CE} = -\log(0.69) = 0.37$
- If y = 0 (negative sentiment), $L_{CE} = -\log(0.31) = 1.17$

Properties of CE Loss

- $L_{CE} = -\sum_{i=1}^{n} [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$
- What values can this loss take?

A) 0 to ∞ B) $-\infty$ to ∞ C) $-\infty$ to 0 D) 1 to ∞

Properties of CE Loss

- $L_{CE} = -\sum_{i=1}^{n} [y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})]$
- Ranges from 0 (perfect predictions) to ∞
- Lower the value, better the classifier
- *Cross-entropy* between the true distribution $P(y \mid x)$ in the data and predicted distribution $P(\hat{y} \mid x)$

4. Optimization

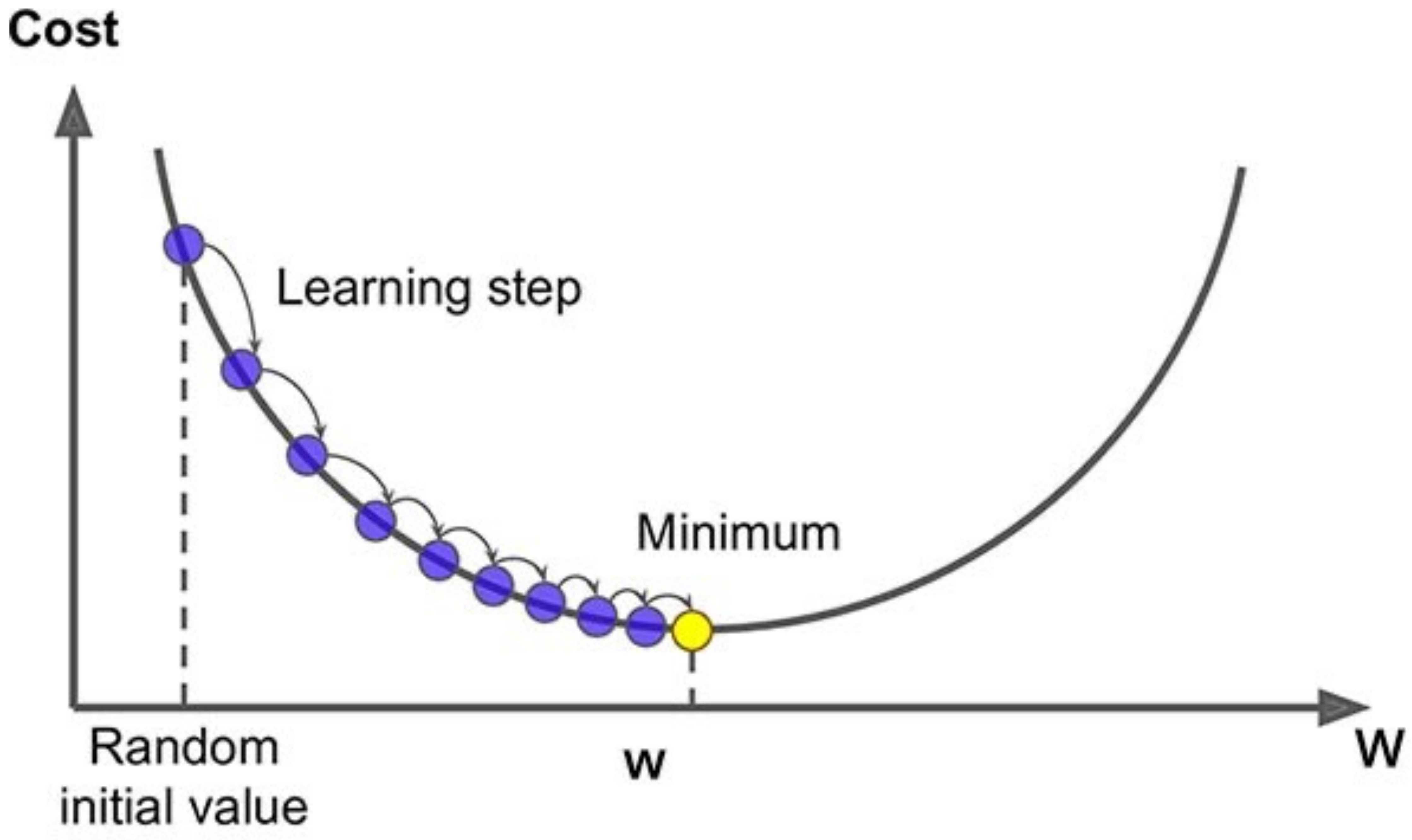
- We have our **classification function** and **loss function** - how do we find the best w and b?

$$\theta = [w; b]$$

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} L_{CE}(y^{(i)}, x^{(i)}; \theta)$$

- Gradient descent:
  - Find direction of steepest slope
  - Move in the opposite direction

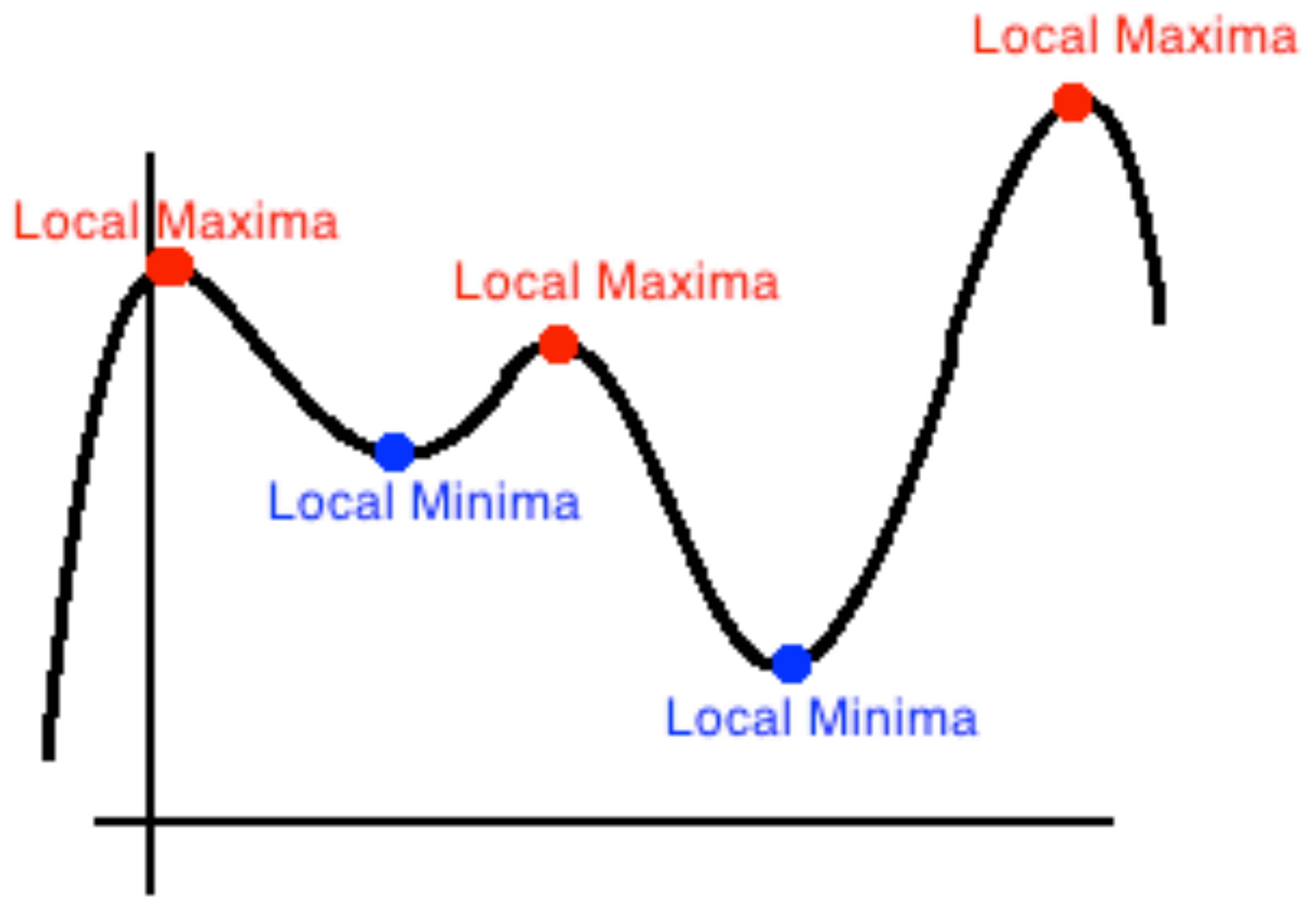
Gradient descent (1-D)



$$\theta^{t+1} = \theta^t - \eta \frac{d}{d\theta} f(x;\theta)$$

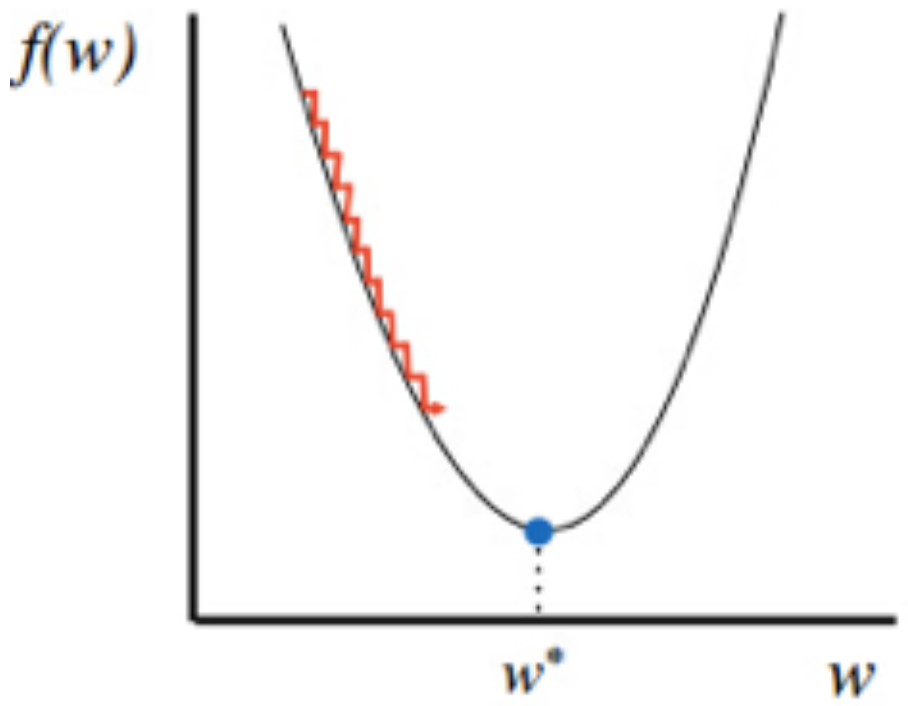
Gradient descent for LR

- Cross entropy loss for logistic regression is **convex** (i.e. has only one global minimum)
  - No local minima to get stuck in
- Deep neural networks are not so easy
  - Non-convex

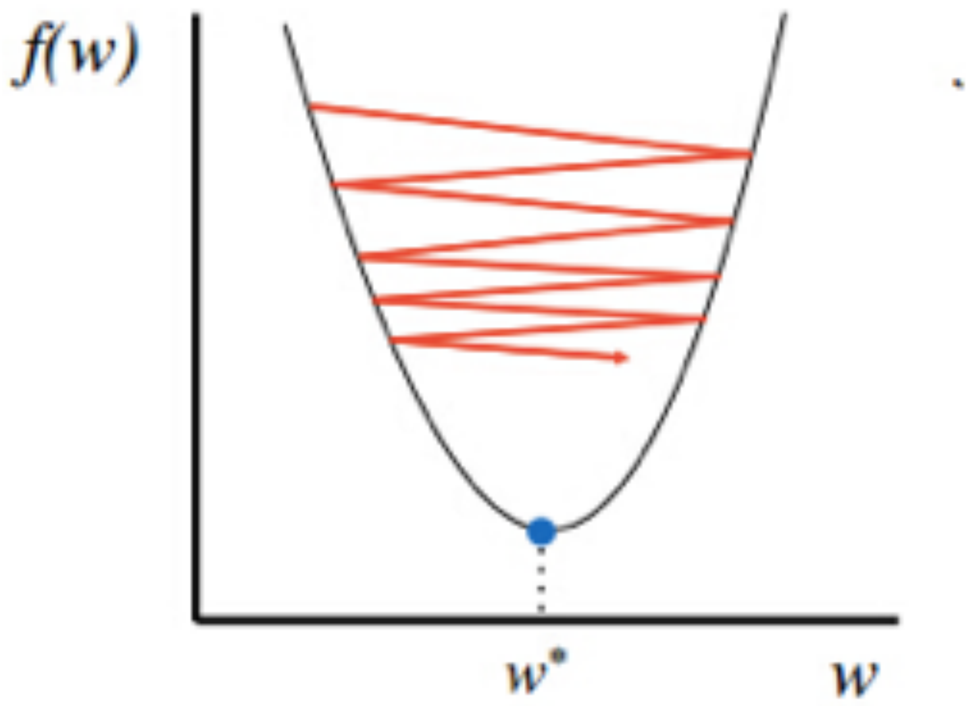


Learning Rate

- Updates: $\theta^{t+1} = \theta^{t} - \eta \frac{d}{d\theta} f(x;\theta)$
- Magnitude of movement along gradient
- Higher/faster learning rate = larger updates to parameters



Too small: converge very slowly



Too big: overshoot and even diverge

Recap: Logistic regression

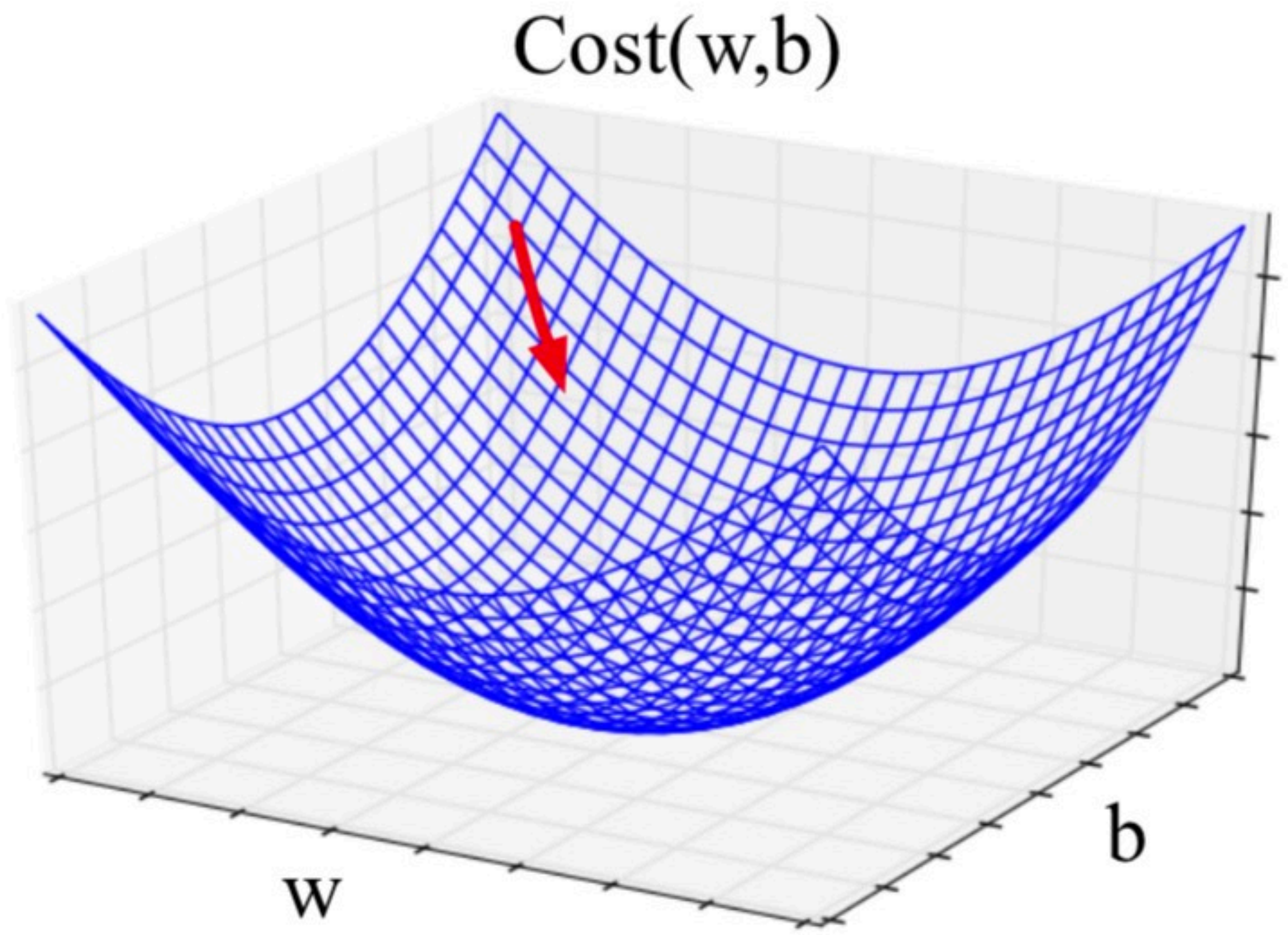
- Inputs:
    1. Classification instance in a **feature representation**
    2. **Classification function** to compute $\hat{y}$ using $P(\hat{y} \mid x)$
    3. **Loss function** (for learning)
    4. Optimization **algorithm**
- Train phase: Learn the **parameters** of the model to minimize **loss function**
- Test phase: Apply **parameters** to predict class given a new input x

Gradient descent with vector weights

- In LR: weight w is a vector

- Express slope as a partial derivative of loss w.r.t each weight:

$$\nabla_{\theta} L(f(x;\theta),y)) = \begin{bmatrix} \frac{\partial}{\partial w_1} L(f(x;\theta),y) \\ \frac{\partial}{\partial w_2} L(f(x;\theta),y) \\ \vdots \\ \frac{\partial}{\partial w_n} L(f(x;\theta),y) \end{bmatrix}$$



- Updates: $\theta^{(t+1)} = \theta^t - \eta \nabla L(f(x;\theta), y)$

Gradient for logistic regression

- $L_{CE} = -\sum_{i=1}^{n}[y^{(i)}\log\sigma(\mathbf{w}\cdot\mathbf{x}^{(i)}+b)+(1-y^{(i)})\log(1-\sigma(\mathbf{w}\cdot\mathbf{x}^{(i)}+b))]$
- Gradient, $\dfrac{dL_{CE}(\mathbf{w},b)}{dw_j} = \sum_{i=1}^{n}\underbrace{[\sigma(\mathbf{w}\cdot\mathbf{x}^{(i)}+b)-y^{(i)}]}_{\text{Diff between true y and prediction}}x_j^{(i)}$

  input feature value

- $\dfrac{dL_{CE}(\mathbf{w},b)}{db} = \sum_{i=1}^{n}[\sigma(\mathbf{w}\cdot\mathbf{x}^{(i)}+b)-y^{(i)}]$

Stochastic Gradient Descent

- Online optimization
- Compute loss and minimize after **each** training example

function STOCHASTIC GRADIENT DESCENT($L()$, $f()$, x, y) **returns** θ

    # where: L is the loss function
    #   f is a function parameterized by θ
    #   x is the set of training inputs $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$
    #   y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(n)}$

$\theta \leftarrow 0$

repeat til done   # see caption

  For each training tuple $(x^{(i)}, y^{(i)})$ (in random order)

    1. Optional (for reporting):   # How are we doing on this tuple?

      Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ # What is our estimated output $\hat{y}$?

Per Instance Loss

      Compute the loss $L(\hat{y}^{(i)}, y^{(i)})$ # How far off is $\hat{y}^{(i)}$) from the true output $y^{(i)}$?

    2. $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$ # How should we move θ to maximize loss?

    3. $\theta \leftarrow \theta - \eta\, g$ # Go the other way instead

return θ

Stochastic Gradient Descent

- Online optimization
- Compute loss and minimize after **each** training example