School of Computing
UNIVERSITY OF GEORGIA

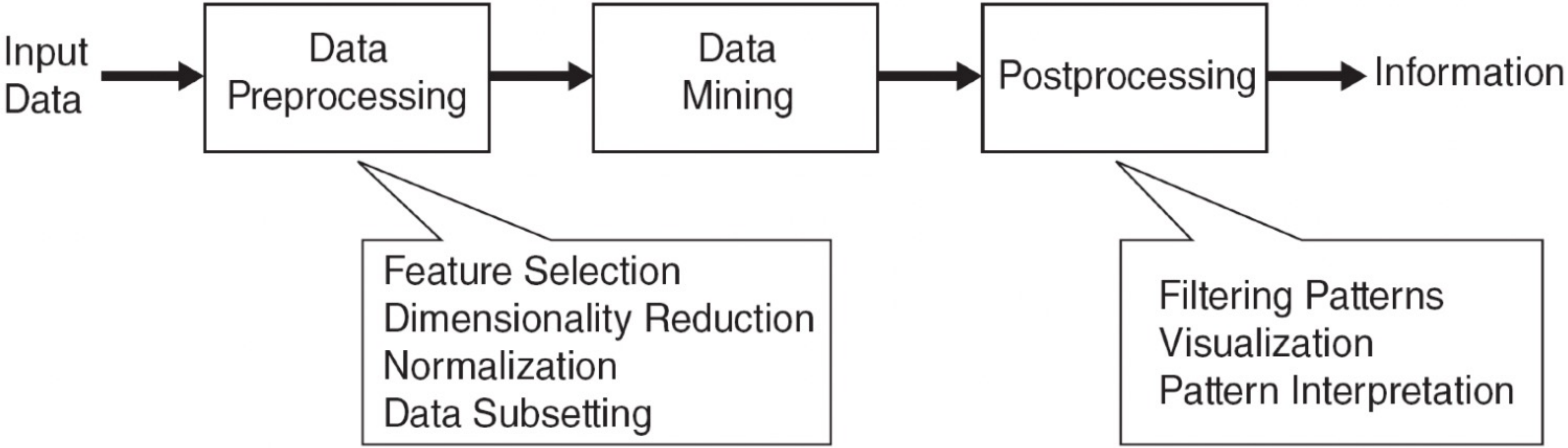
CSCI 4380/6380 DATA MINING

Fei Dou

Assistant Professor
School of Computing
University of Georgia

October 11,12,17,18 2023

Recap: Data Mining Process



Naïve Bayes

Recap: Using Bayes Theorem for Classification

- Approach:
  - compute posterior probability $P(Y \mid X_1, X_2, \ldots, X_d)$ using the Bayes theorem

$$\boxed{P(Y \mid X_1 X_2 \ldots X_n) = \frac{P(X_1 X_2 \ldots X_d \mid Y) P(Y)}{P(X_1 X_2 \ldots X_d)}}$$

  - *Maximum a-posteriori*: Choose Y that maximizes
    $P(Y \mid X_1, X_2, \ldots, X_d)$
  - Equivalent to choosing value of Y that maximizes
    $P(X_1, X_2, \ldots, X_d \mid Y)\, P(Y)$

- How to estimate $P(X_1, X_2, \ldots, X_d \mid Y)$?

Recap: Naïve Bayes Classifier

- Assume independence among attributes X_i when class is given:
  - $P(X_1, X_2, \ldots, X_d | Y_j) = P(X_1| Y_j)\, P(X_2| Y_j) \ldots P(X_d| Y_j)$
  - Now we can estimate $P(X_i| Y_j)$ for all X_i and Y_j combinations from the training data
  - New point is classified to Y_j if $P(Y_j) \prod P(X_i| Y_j)$ is maximal.

Recap: Estimate Probabilities from Data

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- P(y) = fraction of instances of class y
  - e.g., P(No) = 7/10,
    P(Yes) = 3/10

- For categorical attributes:

  $P(X_i = c | y) = n_c / n$

  - where $|X_i = c|$ is number of instances having attribute value $X_i = c$ and belonging to class y
  - Examples:

    P(Status=Married|No) = 4/7
    P(Refund=Yes|Yes)=0

Recap: Estimate Probabilities from Data

- For continuous attributes:
  - Discretization: Partition the range into bins:
    - Replace continuous value with bin value
    - Attribute changed from continuous to ordinal
  - Probability density estimation:
    - Assume attribute follows a normal distribution
    - Use data to estimate parameters of distribution (e.g., mean and standard deviation)
    - Once probability distribution is known, use it to estimate the conditional probability $P(X_i|Y)$

Recap: Estimate Probabilities from Data

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

- Normal distribution:

$$P(X_i \mid Y_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(X_i-\mu_{ij})^2}{2\sigma_{ij}^2}}$$

  - One for each (X_i,Y_i) pair
- For (Income, Class=No):
  - If Class=No
    - sample mean = 110
    - sample variance = 2975

$$P(Income = 120 \mid No) = \frac{1}{\sqrt{2\pi}(54.54)} e^{-\frac{(120-110)^2}{2(2975)}} = 0.0072$$

Recap: Example of Naïve Bayes Classifier

Given a Test Record: $X = (\text{Refund} = \text{No}, \text{Divorced}, \text{Income} = 120\text{K})$

Naïve Bayes Classifier:

P(Refund = Yes | No) = 3/7
P(Refund = No | No) = 4/7
P(Refund = Yes | Yes) = 0
P(Refund = No | Yes) = 1
P(Marital Status = Single | No) = 2/7
P(Marital Status = Divorced | No) = 1/7
P(Marital Status = Married | No) = 4/7
P(Marital Status = Single | Yes) = 2/3
P(Marital Status = Divorced | Yes) = 1/3
P(Marital Status = Married | Yes) = 0

For Taxable Income:
If class = No: sample mean = 110
sample variance = 2975
If class = Yes: sample mean = 90
sample variance = 25

- P(X | No) = P(Refund=No | No)
  $\times$ P(Divorced | No)
  $\times$ P(Income=120K | No)
  $= 4/7 \times 1/7 \times 0.0072 = 0.0006$
- P(X | Yes) = P(Refund=No | Yes)
  $\times$ P(Divorced | Yes)
  $\times$ P(Income=120K | Yes)
  $= 1 \times 1/3 \times 1.2 \times 10^{-9} = 4 \times 10^{-10}$

Since P(X|No)P(No) > P(X|Yes)P(Yes)

Therefore P(No|X) > P(Yes|X)

=> Class = No

Issues with Naïve Bayes Classifier

Given a Test Record: **X = (Married)**

Naïve Bayes Classifier:

P(Refund = Yes | No) = 3/7
P(Refund = No | No) = 4/7
P(Refund = Yes | Yes) = 0
P(Refund = No | Yes) = 1
P(Marital Status = Single | No) = 2/7
P(Marital Status = Divorced | No) = 1/7
P(Marital Status = Married | No) = 4/7
P(Marital Status = Single | Yes) = 2/3
P(Marital Status = Divorced | Yes) = 1/3
P(Marital Status = Married | Yes) = 0

For Taxable Income:
If class = No: sample mean = 110
sample variance = 2975
If class = Yes: sample mean = 90
sample variance = 25

P(Yes) = 3/10
P(No) = 7/10

P(Yes | Married) = 0 x 3/10 / P(Married)
P(No | Married) = 4/7 x 7/10 / P(Married)

Issues with Naïve Bayes Classifier

Consider the table with Tid = 7 deleted

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Naïve Bayes Classifier:

P(Refund = Yes | No) = 2/6
P(Refund = No | No) = 4/6
P(Refund = Yes | Yes) = 0
P(Refund = No | Yes) = 1
P(Marital Status = Single | No) = 2/6
P(Marital Status = Divorced | No) = 0
P(Marital Status = Married | No) = 4/6
P(Marital Status = Single | Yes) = 2/3
P(Marital Status = Divorced | Yes) = 1/3
P(Marital Status = Married | Yes) = 0/3
For Taxable Income:
If class = No: sample mean = 91
sample variance = 685
If class = No: sample mean = 90
sample variance = 25

Given X = (Refund = Yes, Divorced, 120K)

P(X | No) = 2/6 X 0 X 0.0083 = 0

P(X | Yes) = 0 X 1/3 X 1.2 X 10^{-9} = 0

Naïve Bayes will not be able to classify X as Yes or No!

Issues with Naïve Bayes Classifier

- If one of the conditional probabilities is zero, then the entire expression becomes zero
- Need to use other estimates of conditional probabilities than simple fractions
- Probability estimation:

original: $P(X_i = c|y) = \frac{n_c}{n}$

Laplace Estimate: $P(X_i = c|y) = \frac{n_c + 1}{n + v}$

$\mathrm{m-estimate}$: $P(X_i = c|y) = \frac{n_c + mp}{n + m}$

n: number of training instances belonging to class y

n_c: number of instances with $X_i = c$ and $Y = y$

v: total number of attribute values that X_i can take

p: initial estimate of
(P($X_i = c|y$) known apriori

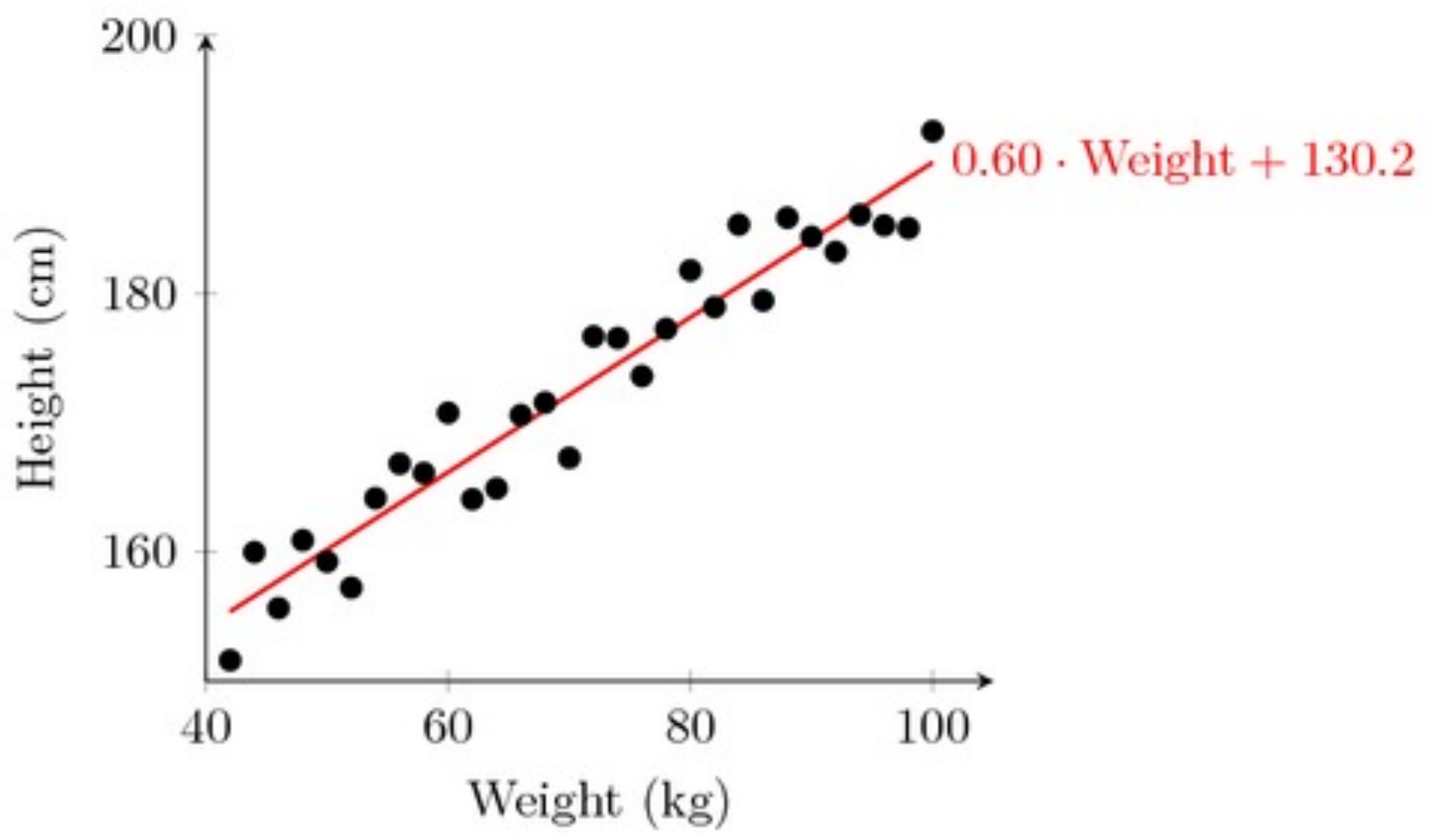
m: hyper-parameter for our confidence in p

Naïve Bayes (Summary)

- Robust to isolated noise points
- Handle missing values by ignoring the instance during probability estimate calculations
- Robust to irrelevant attributes
- Redundant and correlated attributes will violate class conditional assumption
  - Use other techniques such as Bayesian Belief Networks (BBN)

Logistic Regression

Linear Regression



Linear Regression

- Given data $\mathbf{x}_i$ and y_i for an example i
- $\mathbf{x}_i$ is data for independent variables
- y_i is the value for a response variable
- y_i takes continuous number
- Find a linear function f that best predicts y_i based on $\mathbf{x}_i$

$$f(\mathbf{x}_i) = \mathbf{x}_i^T \mathbf{w} \rightarrow y_i$$

- Find best $\mathbf{w}$ with least squares

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \left(y_i - \mathbf{x}_i^T \mathbf{w}\right)^2$$

Closed-form solution:

Normal Equation:

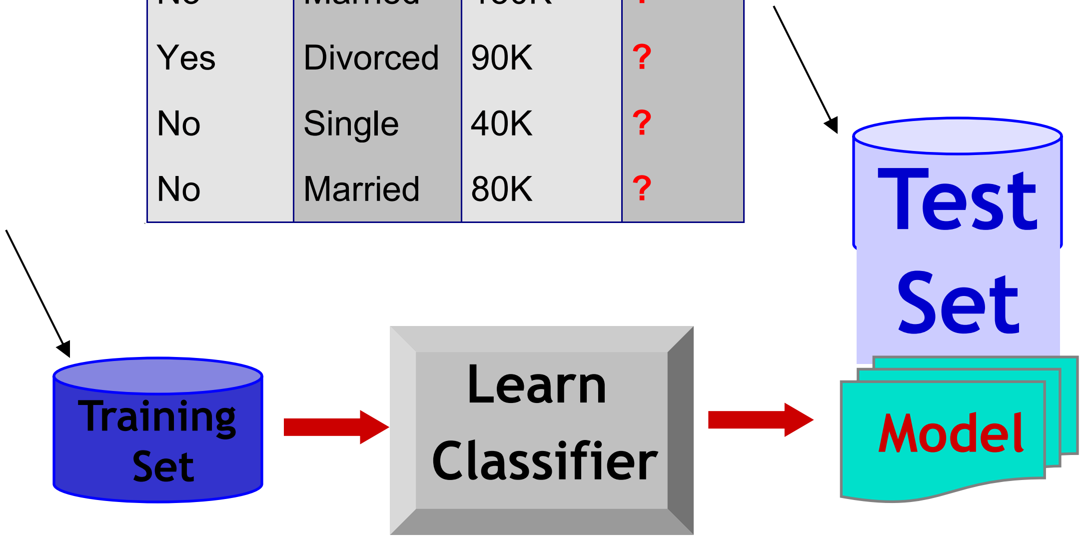
$$\boldsymbol{w} = (X^T X)^{-1} X^T y$$

Classification (example)

categorical categorical continuous class

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Refund	Marital Status	Taxable Income	Cheat
No	Single	75K	?
Yes	Married	50K	?
No	Married	150K	?
Yes	Divorced	90K	?
No	Single	40K	?
No	Married	80K	?



Why Not Linear Regression?

- Code “Yes” with 1 and “No” with -1
- Fit a linear regression model

$$y_i \leftarrow \mathbf{w}^{\mathrm{T}}\mathbf{x}_i$$

- Set up a threshold t (e.g., 0)
- If $y_i \geq t$, classify $\mathbf{x}_i$ as “Yes”, or otherwise “No”

Problem

Pull all predicted values close to either 1 or -1

Restricted searching space

Suboptimal model!

Change the target

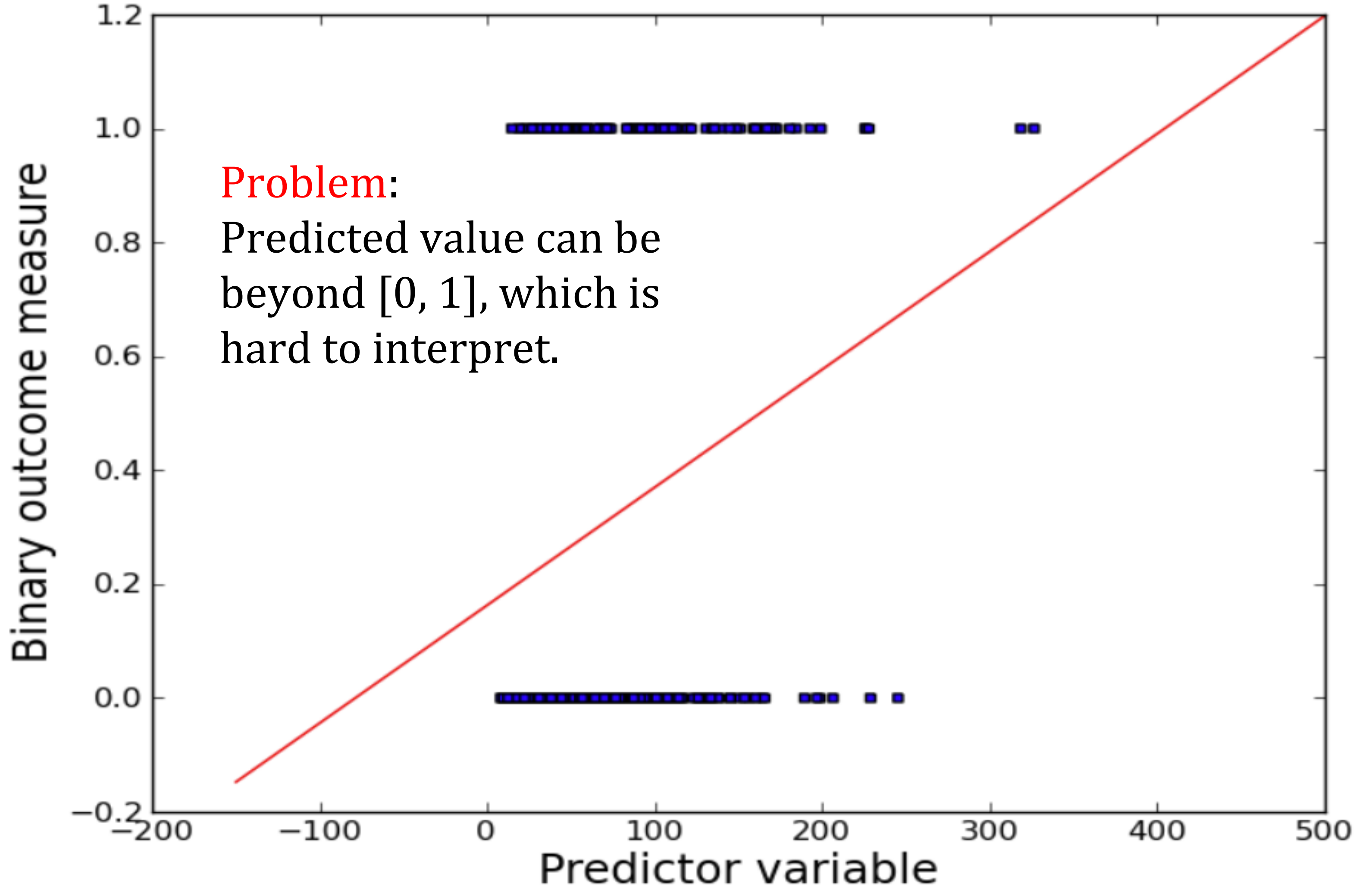
- Predict

$$p(y_i = \text{“yes”}\ or\ \text{“no”} | \mathbf{x}_i)$$

- We do want the predicted value to be close to either 0 or 1

Why not linear regression, again?

Why Not Linear Regression, Again?



Logistic Regression

- Find a function f that best predicts
$$p(y_i = \text{“yes”}\ or\ \text{“no”}|\mathbf{x}_i)$$
- Still find a linear function of $\mathbf{x}_i$ parameterized with $\mathbf{w}$, i.e., $\mathbf{x}_i^T\mathbf{w}$
- But $\mathbf{x}_i^T\mathbf{w}$ is used to compute the probability (p)
- $P \in [0,1]$, however, $\mathbf{x}_i^T\mathbf{w} \in (-\infty, +\infty)$

So, how to link $\mathbf{x}_i^T\mathbf{w}$ to p?

Probability to Odds Ratio

$$\text{Odds}(P) = \frac{P(y_i=1|\mathbf{x}_i)}{P(y_i=0|\mathbf{x}_i)} = \frac{P(y_i=1|\mathbf{x}_i)}{1-P(y_i=1|\mathbf{x}_i)}$$

Probability $P(y_i = 1|\mathbf{x}_i)$	Odds
1.0	$+\infty$
0.99	99
0.75	3
0.5	1
0.25	0.3333
0.01	0.0101
0	0

$p \in [0,1]$

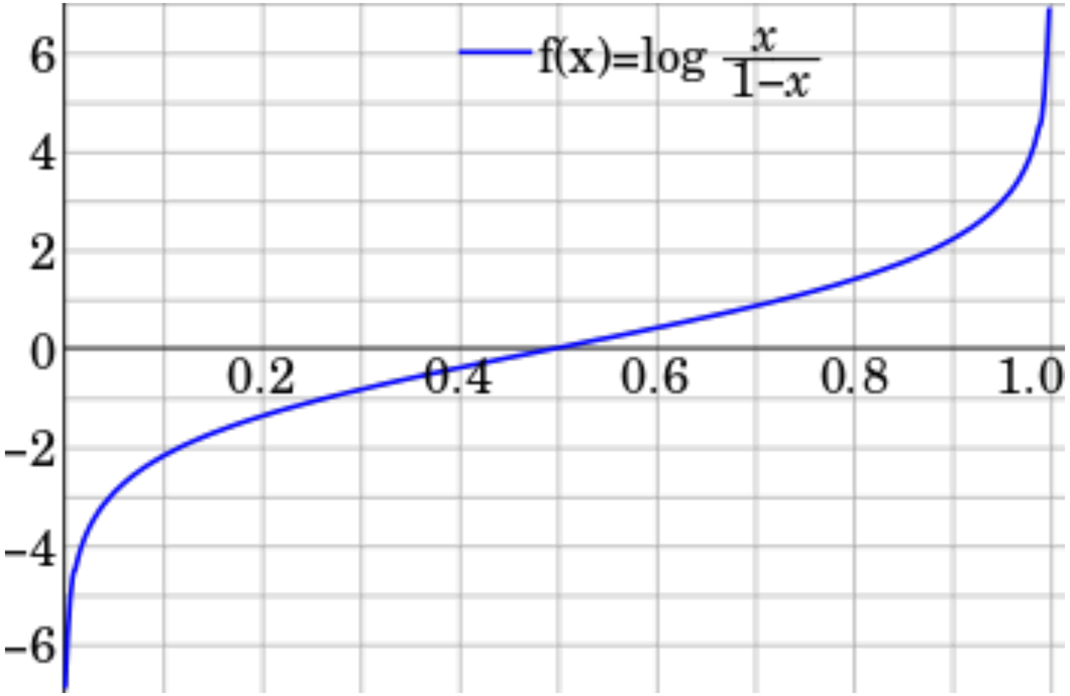
$\text{Odds}(p) \in (0, +\infty)$

$\mathbf{x}_i^T\mathbf{w} \in (-\infty, +\infty)$

Logit Function

- Take the log of odds

$$\log(Odds) = \log(\frac{P}{1-P})$$



We call this function the **logit function** of P, i.e.,

$$logit(P) = \log(\frac{P}{1-P})$$

$$logit(p) \in (-\infty, +\infty)$$

$$\mathbf{x}_i^T \mathbf{w} \in (-\infty, +\infty)$$

Logistic Regression

- Assume log odds is a linear function of $\mathbf{x}$

$$\log\left(\frac{P(y_i = 1|\mathbf{x}_i)}{1 - P(y_i = 1|\mathbf{x}_i)}\right) = \mathbf{x}_i^T\mathbf{w}$$

$$\Downarrow$$

$$P(y_i = 1|\mathbf{x}_i) = \frac{\exp(\mathbf{x}_i^T\mathbf{w})}{1 + \exp(\mathbf{x}_i^T\mathbf{w})}$$

Sigmoid function

$$\sigma(\mathbf{x}_i^T\mathbf{w}) = \frac{\exp(\mathbf{x}_i^T\mathbf{w})}{1+\exp(\mathbf{x}_i^T\mathbf{w})} = \frac{1}{1+\exp(-\mathbf{x}_i^T\mathbf{w})}$$

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

Maximum Likelihood Estimation (MLE)

- When $y_i = 1$, find $\mathbf{w}$ that maximizes

$$P(y_i = 1|\mathbf{x}_i)$$

- When $y_i = 0$, find $\mathbf{w}$ that maximizes

$$P(y_i = 0|\mathbf{x}_i) = 1 - P(y_i = 1|\mathbf{x}_i)$$

- Overall

$$\max_{\mathbf{w}} \left(\prod_{i:y_i=1} P(y_i = 1|\mathbf{x}_i) \prod_{i:y_i=0} \big(1 - P(y_i = 1|\mathbf{x}_i)\big) \right)$$

Maximum Likelihood Estimation (MLE)

$$\max_{\mathbf{w}} \left(\prod_{i:y_i=1} P(y_i = 1|\mathbf{x}_i) \prod_{i:y_i=0} \left(1 - P(y_i = 1|\mathbf{x}_i)\right) \right)$$

Let $P_i = P(y_i = 1|\mathbf{x}_i)$

$$\max_{\mathbf{w}} \left(\prod_{i} p_i^{y_i} (1 - p_i)^{(1-y_i)} \right)$$

Maximum Likelihood Estimation (MLE)

$$\max_{\mathbf{w}} \left(\prod_i p_i^{y_i} (1 - p_i)^{(1-y_i)} \right)$$

$$\Big\downarrow \text{ Take log}$$

$$\max_{\mathbf{w}} \left(\sum_i y_i \log p_i + (1 - y_i) \log(1 - p_i) \right)$$

$$\Big\downarrow \; p_i = \frac{\exp(\mathbf{x}_i^T \mathbf{w})}{1 + \exp(\mathbf{x}_i^T \mathbf{w})}$$

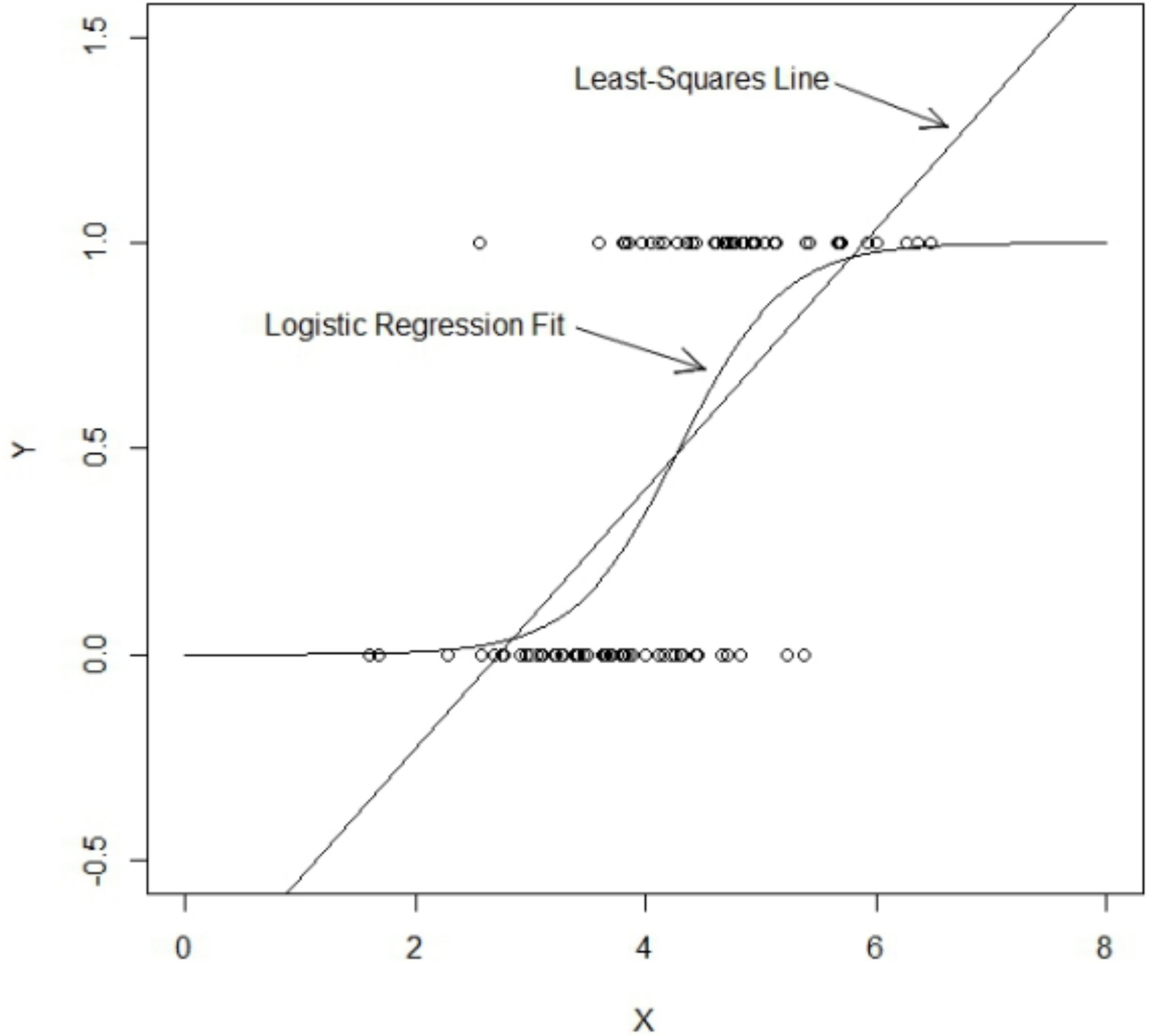
$$\max_{\mathbf{w}} \left(\sum_i y_i \mathbf{x}_i^T \mathbf{w} - \log\left(1 + \exp(\mathbf{x}_i^T \mathbf{w})\right) \right)$$

$$\Big\downarrow$$

$$\min_{\mathbf{w}} \left(\sum_i \log\left(1 + \exp(\mathbf{x}_i^T \mathbf{w})\right) - y_i \mathbf{x}_i^T \mathbf{w} \right)$$

Solved with gradient descent

Linear Regression vs Logistic Regression



Logistic Regression – Cross Entropy Loss

Wait, where did the w's come from?

- Supervised classification:
  - We know the correct label y (either 0 or 1) for each x.
  - But what the system produces is an estimate, $\hat{y}$

- We want to set w and b to minimize the **distance** between our estimate $\hat{y}^{(i)}$ and the true $y^{(i)}$.
  - We need a distance estimator: a **loss function** or a **cost function**
  - We need an optimization algorithm to update w and b to minimize the loss.

Learning components

- A loss function:
  - **cross-entropy loss**
- An optimization algorithm:
  - **stochastic gradient descent**

The distance between $\hat{y}$ and y

- We want to know how far is the classifier output:

$$\hat{y} = \sigma(w \cdot x + b)$$

$$\sigma(\mathbf{x}_i^T \mathbf{w}) \longrightarrow \sigma(\mathbf{wx} + \mathrm{b})$$

- from the true output:

y [= either 0 or 1]

- We'll call this difference:

$L(\hat{y}, y)$ = how much $\hat{y}$ differs from the true y

Intuition of negative log likelihood loss = cross-entropy loss

- A case of conditional maximum likelihood estimation
- We choose the parameters *w*,*b* that maximize
  - the log probability
  - of the true *y* labels in the training data
  - given the observations *x*

Deriving cross-entropy loss for a single observation x

- **Goal**: maximize probability of the correct label $p(y|x)$
  - Since there are only 2 discrete outcomes (0 or 1) we can express the probability $p(y|x)$ from our classifier (the thing we want to maximize) as

$$p(y|x) = \hat{y}^{y} (1 - \hat{y})^{1-y}$$

  - noting:
    - if y=1, this simplifies to $\hat{y}$
    - if y=0, this simplifies to 1- $\hat{y}$

Deriving cross-entropy loss for a single observation x

- **Goal**: maximize probability of the correct label $p(y|x)$
  - Maximize: $$p(y|x) = \hat{y}^y (1-\hat{y})^{1-y}$$
- Now take the log of both sides (mathematically handy)
  - Maximize: $$\begin{aligned} \log p(y|x) &= \log \left[\hat{y}^y (1-\hat{y})^{1-y}\right] \\ &= y \log \hat{y} + (1-y)\log(1-\hat{y}) \end{aligned}$$
- Whatever values maximize log p(y|x) will also maximize p(y|x)

Deriving cross-entropy loss for a single observation x

Goal: maximize probability of the correct label $p(y|x)$

Maximize

$$\begin{aligned} \log p(y|x) &= \log \left[\hat{y}^{y}\,(1-\hat{y})^{1-y}\right] \\ &= y\log\hat{y} + (1-y)\log(1-\hat{y}) \end{aligned}$$

Now flip sign to turn this into a loss: something to minimize

Cross-entropy loss (because is formula for cross-entropy(y, $\hat{y}$))

Minimize

$$L_{\mathrm{CE}}(\hat{y},y) = -\log p(y|x) \;=\; -\left[y\log\hat{y} + (1-y)\log(1-\hat{y})\right]$$

Or, plugging in definition of $\hat{y}$:

$$L_{\mathrm{CE}}(\hat{y},y) \;=\; -\left[y\log\sigma(w\cdot x+b) + (1-y)\log\left(1-\sigma(w\cdot x+b)\right)\right]$$

Deriving cross-entropy loss for a single observation x

- We want loss to be:
  - smaller if the model estimate is close to correct
  - bigger if model is confused

$$L_{\mathrm{CE}}(\hat{y},y) = -\log p(y|x) \;=\; -[y\log\hat{y} + (1-y)\log(1-\hat{y})]$$

Logistic Regression – Stochastic Gradient Descent

Our goal: minimize the loss

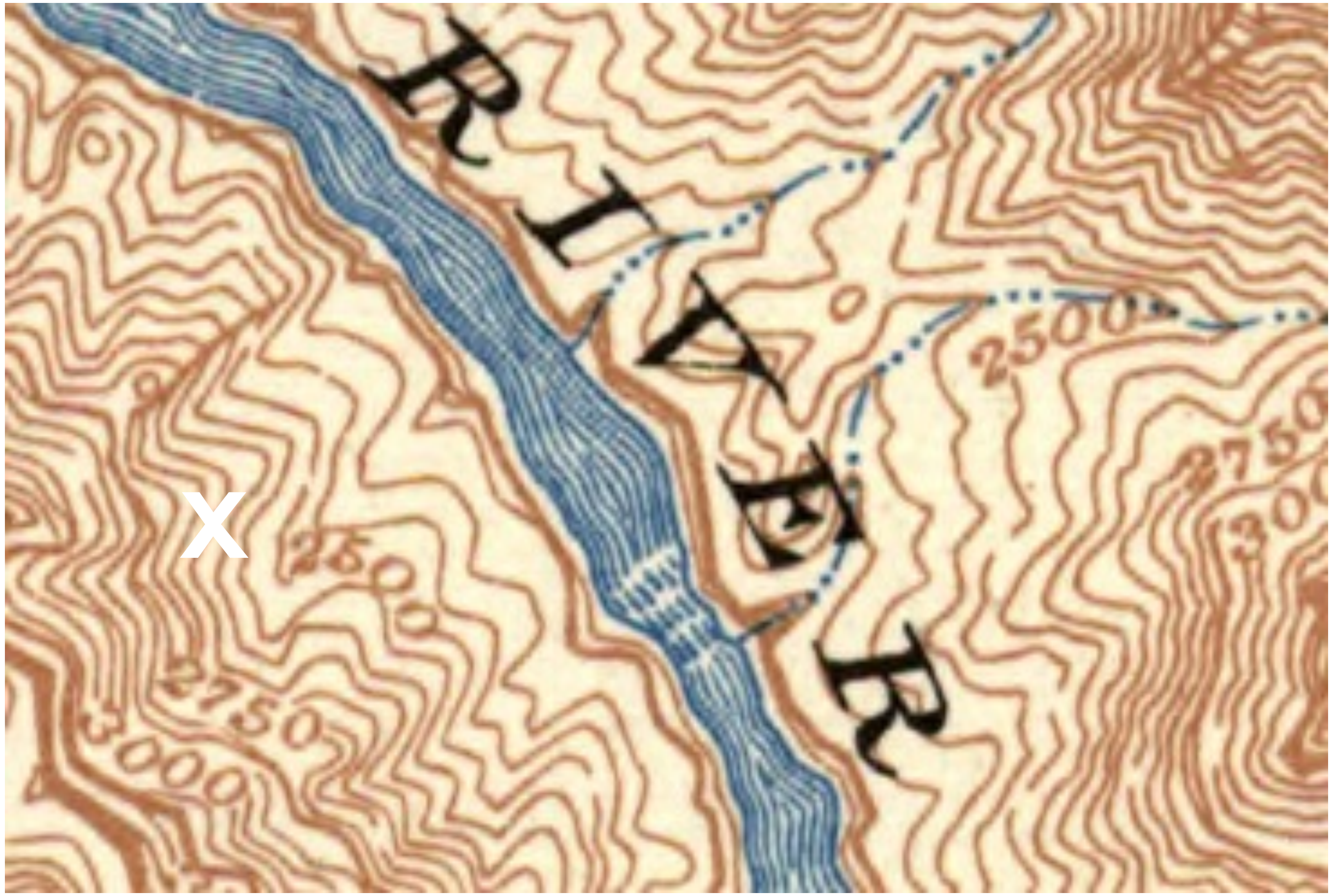
- Let's make explicit that the loss function is parameterized by weights θ=(w,b)
  - And we'll represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious

- We want the weights that minimize the loss, averaged over all examples:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} L_{\mathrm{CE}}(f(x^{(i)};\theta), y^{(i)})$$

Intuition of gradient descent

- How do I get to the bottom of this river canyon?



Look around me 360°
Find the direction of steepest slope down

Go that way

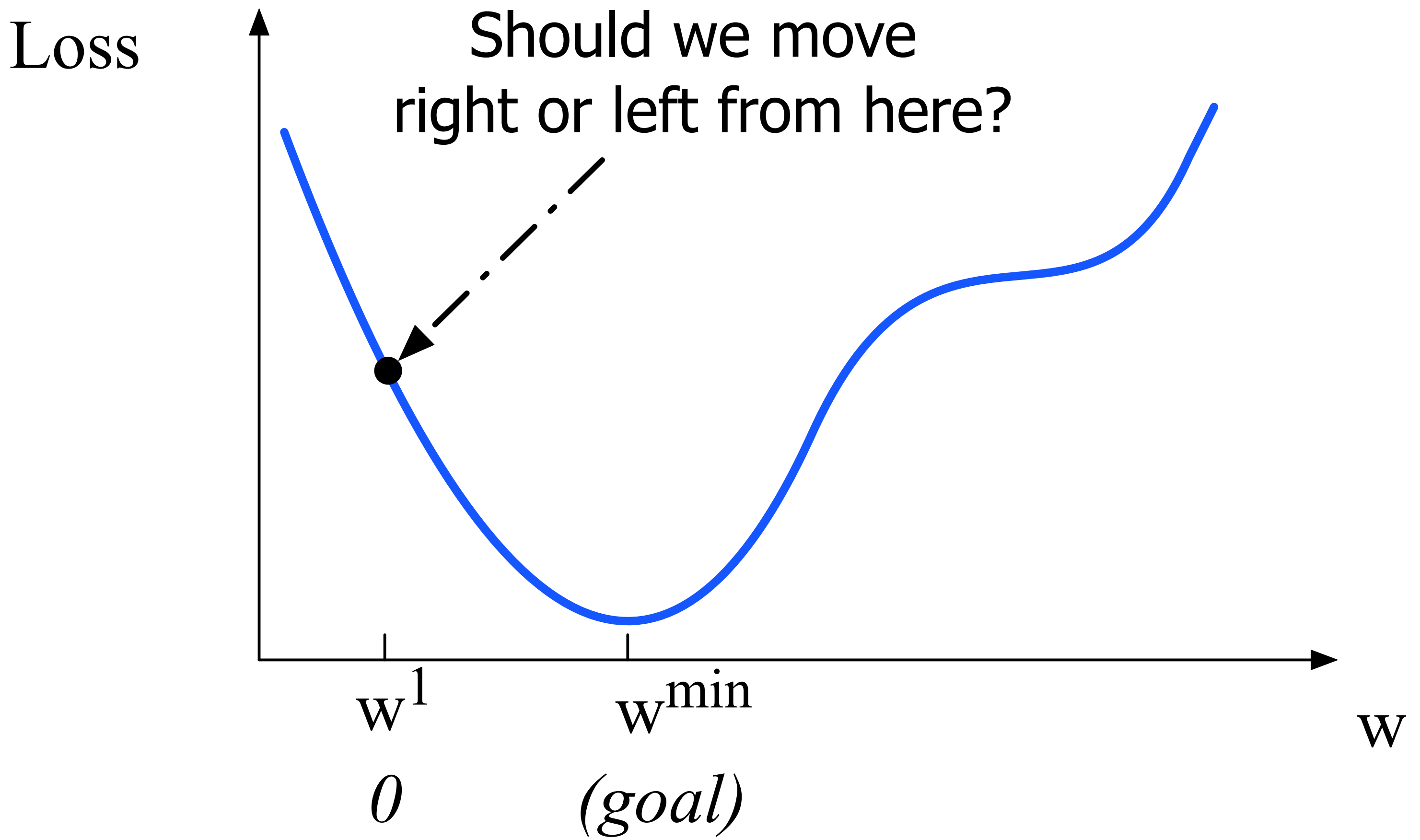
Our goal: minimize the loss

- For logistic regression, loss function is **convex**
  - A convex function has just one minimum
  - Gradient descent starting from any point is guaranteed to find the minimum
    - (Loss for neural networks is non-convex)

Let's first visualize for a single scalar w

Q: Given current w, should we make it bigger or smaller?

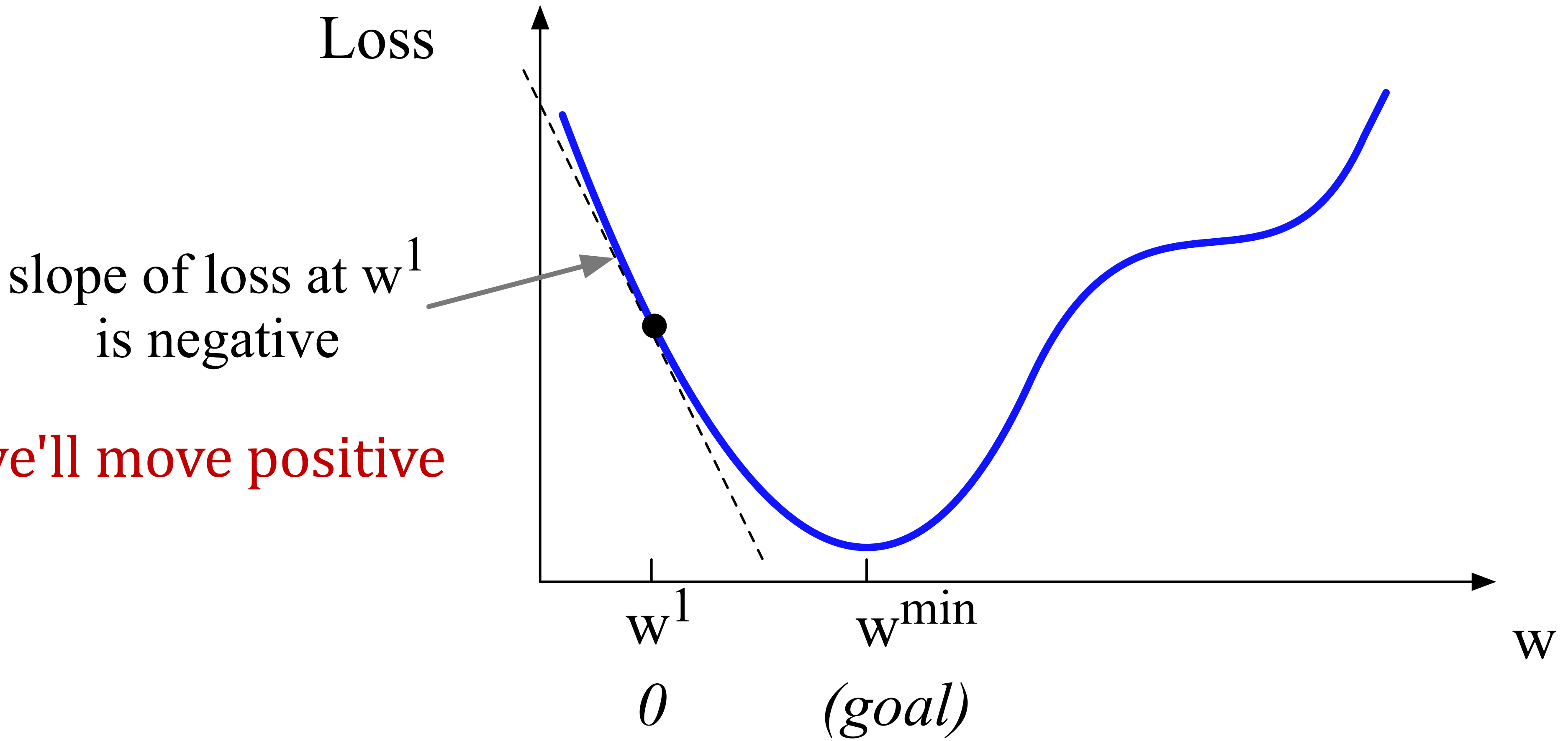
A: Move *w* in the reverse direction from the slope of the function



Let's first visualize for a single scalar w

Q: Given current w, should we make it bigger or smaller?

A: Move *w* in the reverse direction from the slope of the function

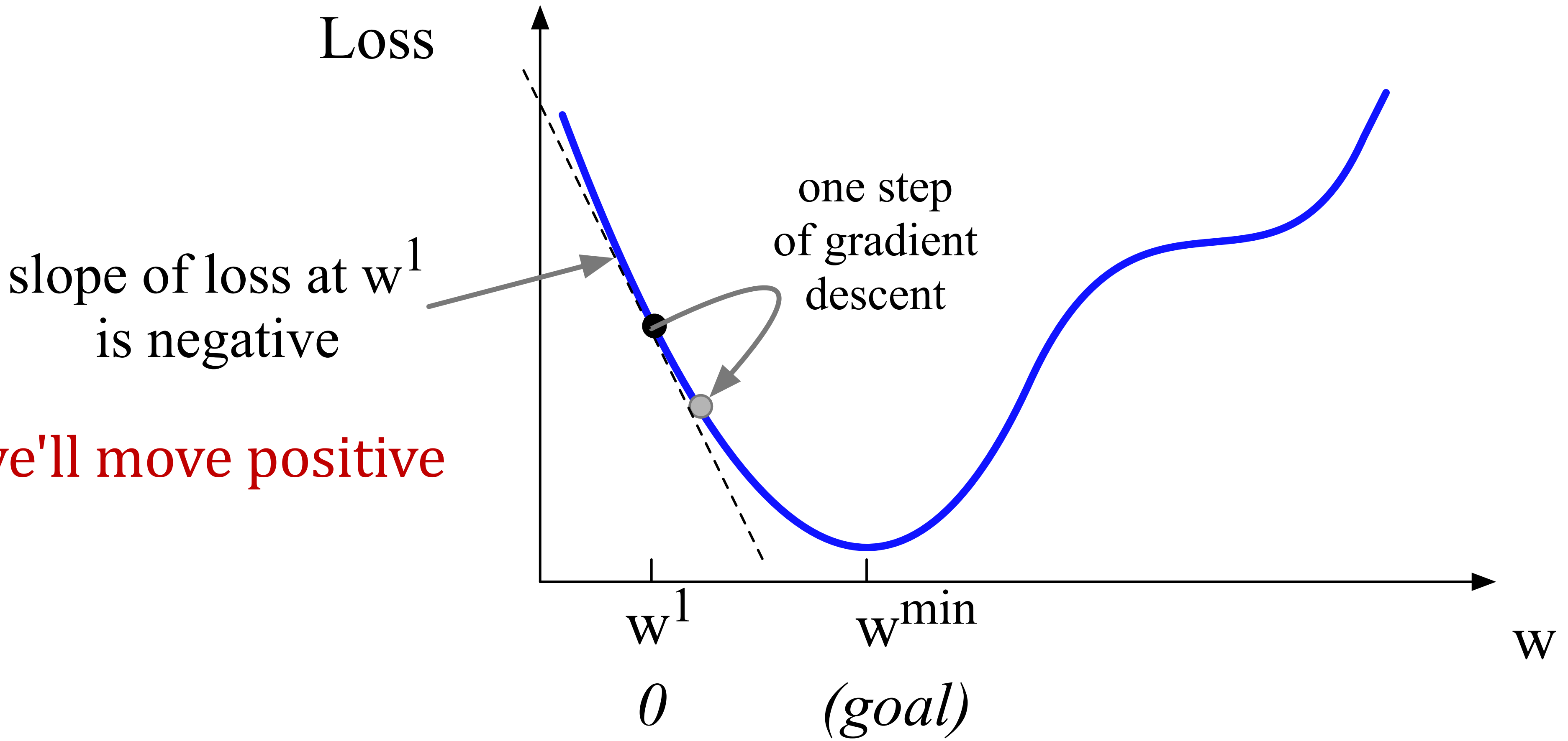


So we'll move positive

Let's first visualize for a single scalar w

Q: Given current w, should we make it bigger or smaller?

A: Move *w* in the reverse direction from the slope of the function



So we'll move positive

Gradients

- The **gradient** of a function of many variables is a vector pointing in the direction of the greatest increase in a function.

- **Gradient Descent**: Find the gradient of the loss function at the current point and move in the **opposite** direction.

How much do we move in that direction ?

- The value of the gradient (slope in our example) $\frac{d}{dw}L(f(x;w),y)$ weighted by a **learning rate** η

- Higher learning rate means move w faster

$$w^{t+1} = w^t - \eta \frac{d}{dw} L(f(x;w),y)$$

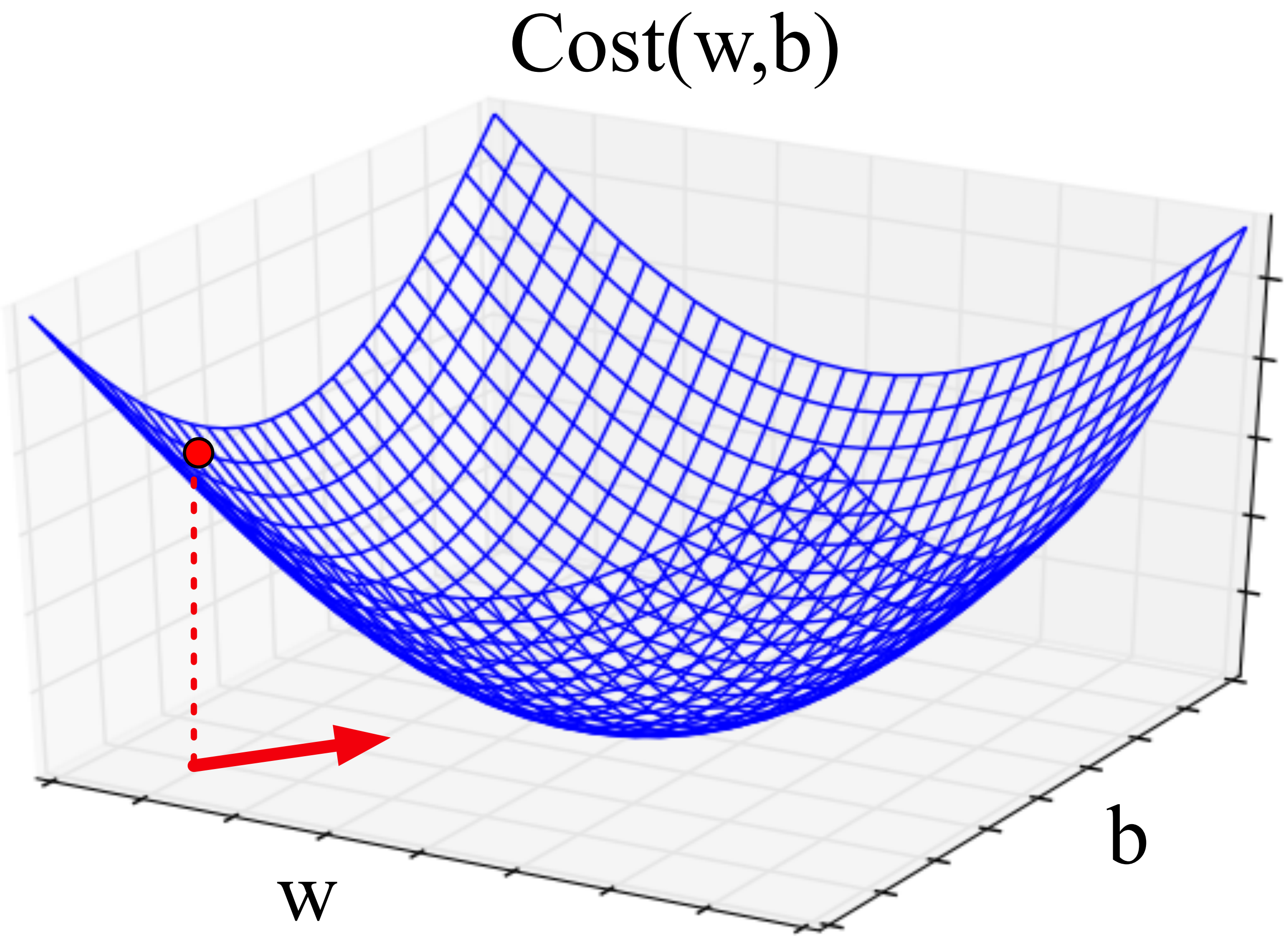
Now let's consider N dimensions

- We want to know where in the N-dimensional space (of the N parameters that make up θ) we should move.

- The gradient is just such a vector; it expresses the directional components of the sharpest slope along each of the N dimensions.

Imagine 2 dimensions, w and b

- Visualizing the gradient vector at the red point
- It has two dimensions shown in the x-y plane



Real gradients

- Are much longer; lots and lots of weights

- For each dimension w_i the gradient component i tells us the slope with respect to that variable.
  - “How much would a small change in w_i influence the total loss function L?”
  - We express the slope as a partial derivative ∂ of the loss ∂w_i

- The gradient is then defined as a vector of these partials.

The gradient

We'll represent $\hat{y}$ as $f(x; \theta)$ to make the dependence on θ more obvious:

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

The final equation for updating θ based on the gradient is thus

$$\theta_{t+1} \quad = \quad \theta_t - \eta \nabla L(f(x;\theta),y)$$

What are these partial derivatives for logistic regression?

The loss function

$$L_{\text{CE}}(\hat{y},y) \;=\; -\left[y\log\sigma(w\cdot x+b)+(1-y)\log\left(1-\sigma(w\cdot x+b)\right)\right]$$

The elegant derivative of this function

$$\frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_j} \;=\; [\sigma(w\cdot x+b)-y]x_j$$

Stochastic Gradient Descent

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 $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$

    #   y is the set of training outputs (labels) $y^{(1)}, y^{(2)}, \ldots, y^{(m)}$

$\theta \leftarrow 0$

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 $\hat{y}$?

      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

Hyperparameters

- The learning rate η is a **hyperparameter**
  - too high: the learner will take big steps and overshoot
  - too low: the learner will take too long

- Hyperparameters:
  - Briefly, a special kind of parameter for an ML model
  - Instead of being learned by algorithm from supervision (like regular parameters), they are chosen by algorithm designer.

Stochastic Gradient Descent: example & more details

Working through an example

One step of gradient descent

- A mini-sentiment example, where the true y=1 (positive)
- Two features:
  - $x_1 = 3$ (count of positive lexicon words)
  - $x_2 = 2$ (count of negative lexicon words)
- Assume 3 parameters (2 weights and 1 bias) in Θ^0 are zero:
  - $w_1 = w_2 = b = 0$
  - $\eta = 0.1$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

- Update step for update θ is:

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

where
$$\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix}$$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

- Update step for update θ is:

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

where

$$\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

- Update step for update θ is:

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

where

$$\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y]x_j$$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix}$$

Example of gradient descent

- Update step for update θ is:

$w_1 = w_2 = b = 0$;
$x_1 = 3$; $x_2 = 2$

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

where

$$\frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y) x_1 \\ (\sigma(w \cdot x + b) - y) x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1) x_1 \\ (\sigma(0) - 1) x_2 \\ \sigma(0) - 1 \end{bmatrix} =$$

Example of gradient descent

$w_1 = w_2 = b = 0;$
$x_1 = 3; \quad x_2 = 2$

- Update step for update θ is:

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

where

$$\frac{\partial L_{\mathrm{CE}}(\hat{y}, y)}{\partial w_j} = [\sigma(w \cdot x + b) - y] x_j$$

- Gradient vector has 3 dimensions:

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w\cdot x+b)-y)x_1 \\ (\sigma(w\cdot x+b)-y)x_2 \\ \sigma(w\cdot x+b)-y \end{bmatrix} = \begin{bmatrix} (\sigma(0)-1)x_1 \\ (\sigma(0)-1)x_2 \\ \sigma(0)-1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

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

$\eta = 0.1$;

$$\theta^1 =$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

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

$\eta = 0.1$;

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\text{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

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

$\eta = 0.1;$

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Example of gradient descent

$$\nabla_{w,b} = \begin{bmatrix} \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_1} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial w_2} \\ \frac{\partial L_{\mathrm{CE}}(\hat{y},y)}{\partial b} \end{bmatrix} = \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix} = \begin{bmatrix} (\sigma(0) - 1)x_1 \\ (\sigma(0) - 1)x_2 \\ \sigma(0) - 1 \end{bmatrix} = \begin{bmatrix} -0.5x_1 \\ -0.5x_2 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix}$$

Now that we have a gradient, we compute the new parameter vector θ^1 by moving θ^0 in the opposite direction from the gradient:

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

$\eta = 0.1$;

$$\theta^1 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix}$$

Note that enough negative examples would eventually make w_2 negative

Mini-batch training

- Stochastic gradient descent chooses a single random example at a time.
- That can result in choppy movements
- More common to compute gradient over batches of training instances.
  - **Batch training**: entire dataset
  - **Mini-batch training**: m examples (512, or 1024)