UNIVERSITÉ DU LUXEMBOURG

Natural Language Processing

Lecture 6: Text Classification (Part 2)

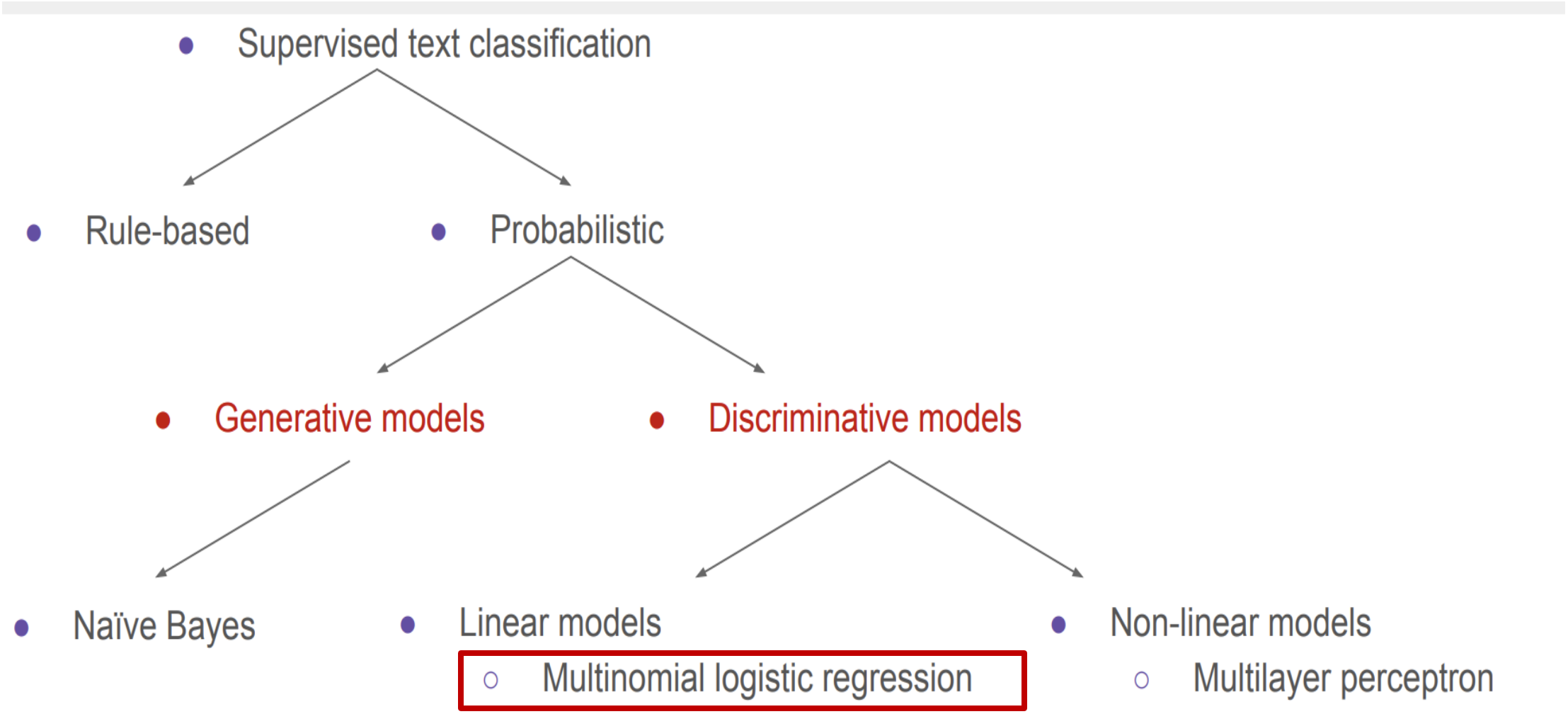
Logistic Regression

Salima Lamsiyah
University Luxembourg, FSTM, DCS, MINE Research Group
Salima.lamsiyah@uni.lu

Lecture Plan

1. Reminder
2. Classification in Logistic Regression
3. Logistic Regression: Sentiment Analysis
4. Learning: Cross-Entropy Loss
5. Optimization: Stochastic Gradient Descent
6. Working through an example
7. Multinomial Logistic Regression
8. Summary

Supervised Text Classification Models



Discriminative Vs **Generative** Machine Learning algorithms

- **Discriminative ML Models** calculate the probability of a latent trait using conditional probability.

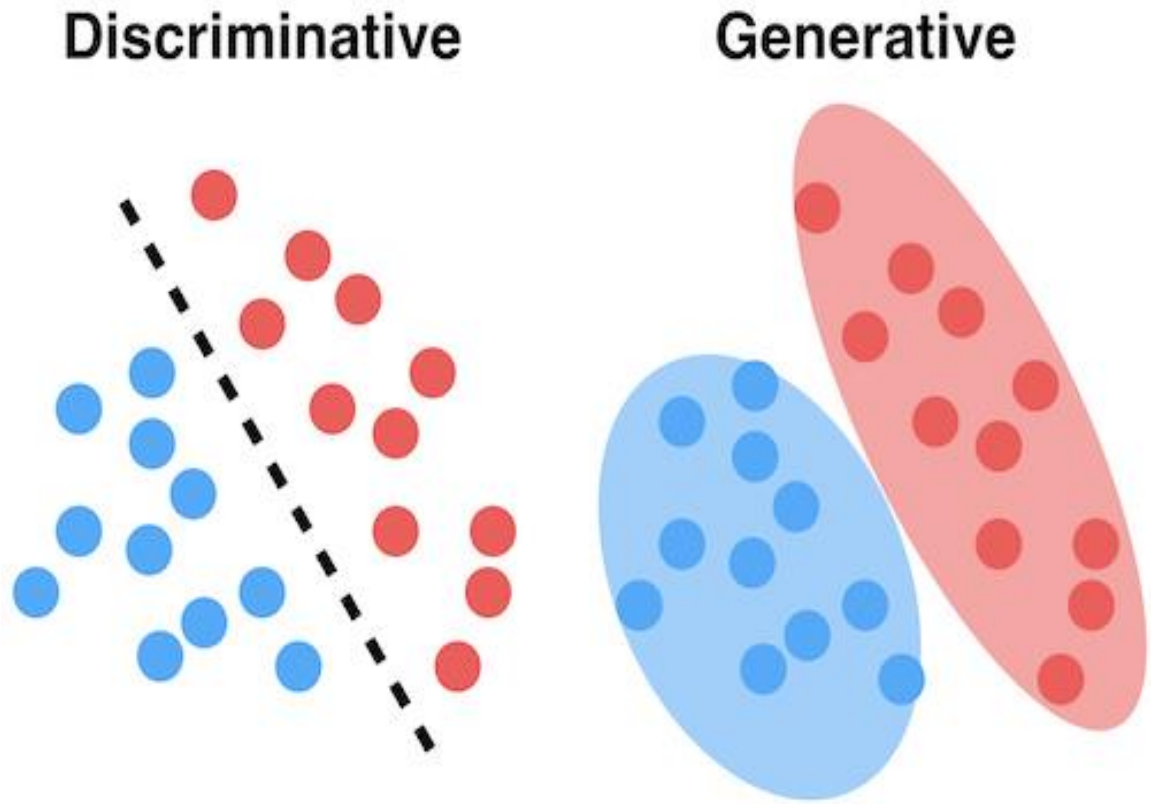
$$P(c|d)$$ **probability of class c given d**

Conditional

- **Generative ML Models** focuses on the distribution of a dataset to return a probability for a given example using Joint Probability.

$$P(d \cap c) = P(d|c).p(c)$$

joint



Source

Examples Generative and Discriminative Models

1. Discriminative Models

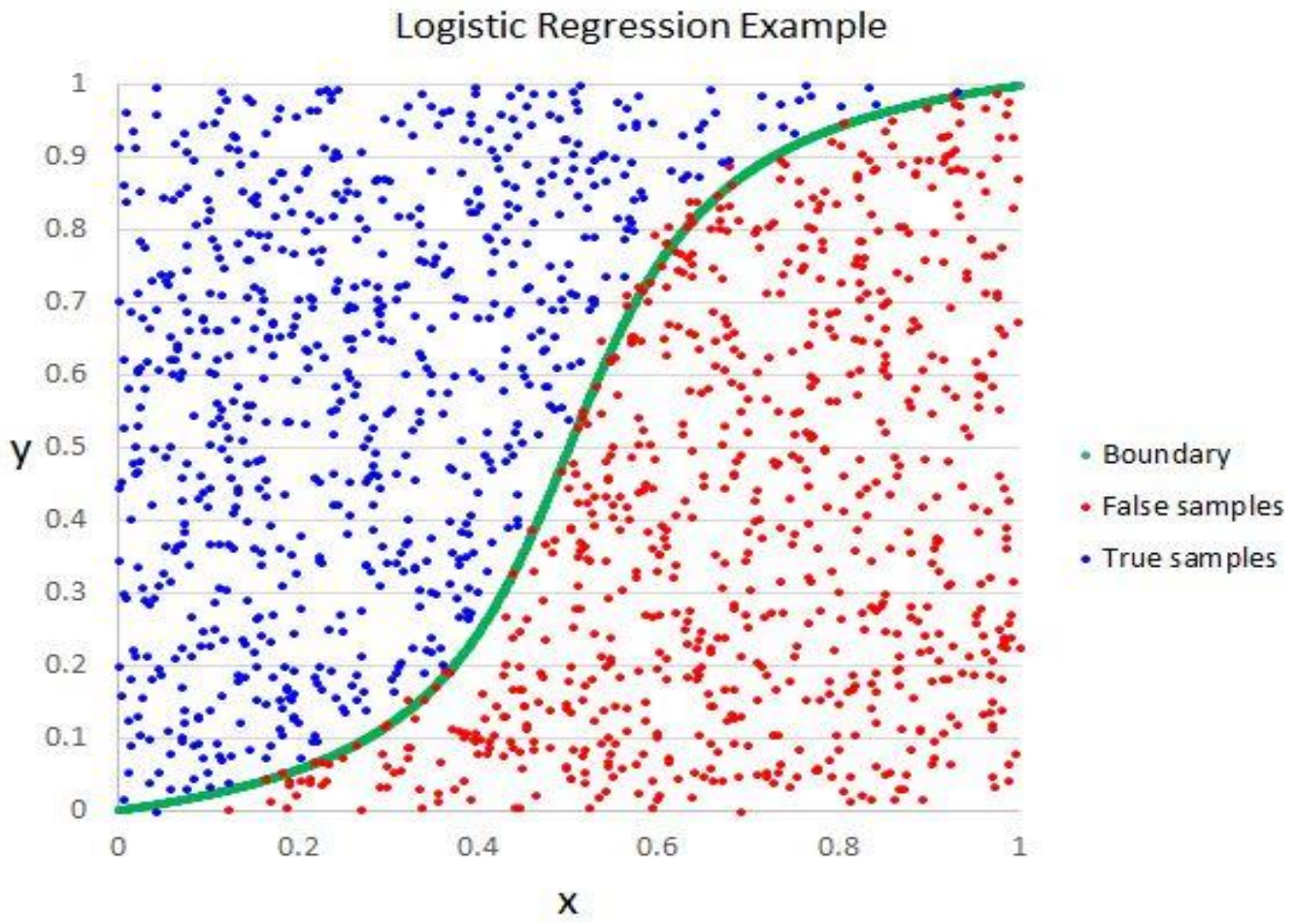
- Support Vector Machine (SVMs)
- K-Nearest neighbors
- Decision Trees and Random Forest
- **Logistic regression**
- Traditional neural networks
- ….

2. Generative Models

- **Naïve Bayes**
- Bayesian networks
- Hidden Markov Models (HMMs)
- Generative Adversarial Networks (GANs)
- Autoregressive Model
- ….

Logistic Regression Classifier

- Powerful supervised machine learning model for classification
- Baseline approach for many NLP tasks.
- It is the foundation of neural networks
- Binary (two classes) or multinomial (>2 classes)



Naïve Bayes Vs Logistic Regression

- Naive Bayes is a **Generative Model**

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \overbrace{P(d|c)}^{\text{likelihood}} \overbrace{P(c)}^{\text{prior}}$$

- Logistic Regression is a **Discriminative Model**

$$\hat{c} = \underset{c \in C}{\operatorname{argmax}} \overbrace{P(c|d)}^{\text{posterior}}$$

Classification using Logistic Regression

Components of a probabilistic machine learning classifier

Given *m* input/output pairs $(x^{(i)}, y^{(i)})$:

1. A **feature representation** of the input.
   - For each input observation $x^{(i)}$, a vector of features $[x_1, x_2, \ldots, x_n]$.
   - Feature *j* for input $x^{(i)}$ is x_j, more completely $x_j^{(i)}$, or sometimes $f_j(x)$.
2. A **classification function** that computes $\hat{y}$, the estimated class, via $p(y|x)$, like the **sigmoid** or **softmax** functions.
3. An objective function for learning, like **cross-entropy loss**.
4. An algorithm for optimizing the objective function: **stochastic gradient descent**.

Features in Logistic Regression

- **Example Sentiment**: *"Compared to the abysmal performances we've seen in the past, today's show was merely mediocre. But the finale was absolutely fantastic!"* **Positive**
- For feature x_i, weight w_i tells is how important is x_i
  - x_1 ="review contains 'fantastic'": w_1 = +10
  - x_2 ="review contains 'abysmal'": w_2 = -2
  - x_3 ="review contains 'mediocre'": w_3 = -10

Logistic Regression for one observation $x^{(i)}$

- **Input observation**: vector $x^{(i)} = [x_1, x_2, \ldots, x_n]$
- **Weights**: one per feature: $W = [w_1, w_2, \ldots, w_n]$
  - Sometimes we call the weights $\theta = [\theta_1, \theta_2, \ldots, \theta_n]$
- **Output**: a predicted class $\hat{y}^{(i)} \in \{0,1\}$
- **Multinomial logistic regression:** $\hat{y}^{(i)} \in \{0, 1, 2, 3, 4, \text{or more}\}$

How to do classification in Logistic Regression

- For each feature x_i, weight w_i tells us importance of x_i
  - (Plus we'll have a bias b)

- We will sum up all the weighted features and the bias

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$z = w \cdot x + b$$

- If this sum is high, we say $y = 1$; if low, then $y = 0$

But we want a probabilistic classifier!

- The problem: z isn't a probability, it's just a number!

$$z = w \cdot x + b$$

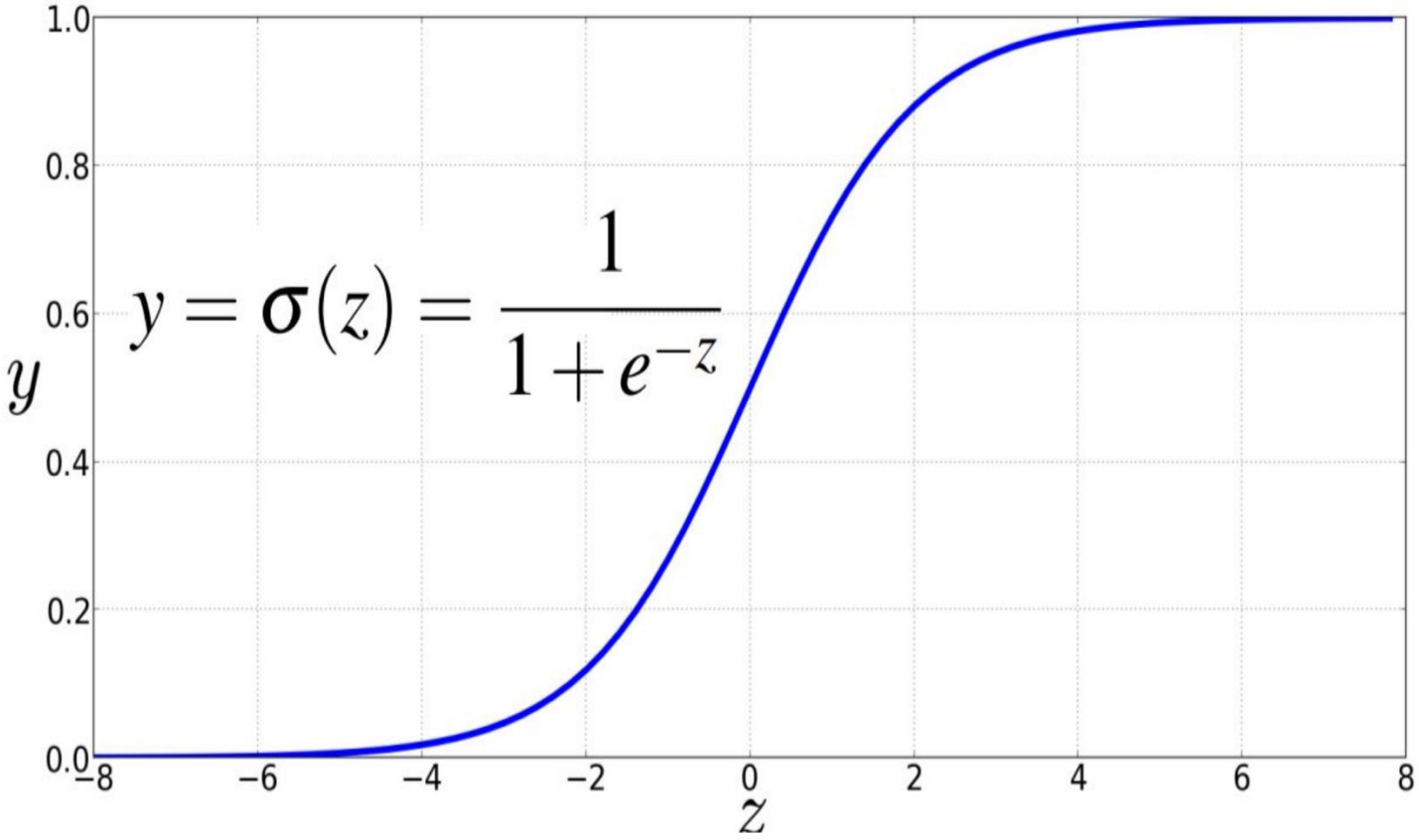
- z ranges from $-\infty$ to $+\infty$, we need to formalize "sum is high".
- We'd like a principled classifier that gives us a probability, just like Naive Bayes did
- We want a model that can tell us:

  $p(y=1|x; \theta)$, $p(y=0|x; \theta)$
- θ refers to the parameters (weights and bias)

But we want a probabilistic classifier!

- Solution: use a function of z that goes from 0 to 1

$$y = s(z) = \frac{1}{1+e^{-z}} = \frac{1}{1+\exp(-z)}$$



"Sigmoid" or "**Logistic**" Function

Idea of logistic regression

- We'll compute $z = w \cdot x + b$
- And then we'll pass it through the sigmoid function:

$$\sigma(z) = \sigma(w \cdot x + b) = \frac{1}{1+\exp(-(w.x+b))}$$

- And we'll just treat it as a probability

Making probabilities with sigmoids!

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

$$P(y=0) = 1 - \sigma(w \cdot x + b)$$
$$= 1 - \frac{1}{1+\exp(-(w \cdot x + b))}$$
$$= \frac{\exp(-(w \cdot x + b))}{1+\exp(-(w \cdot x + b))}$$
$$= \sigma(-(w \cdot x + b))$$

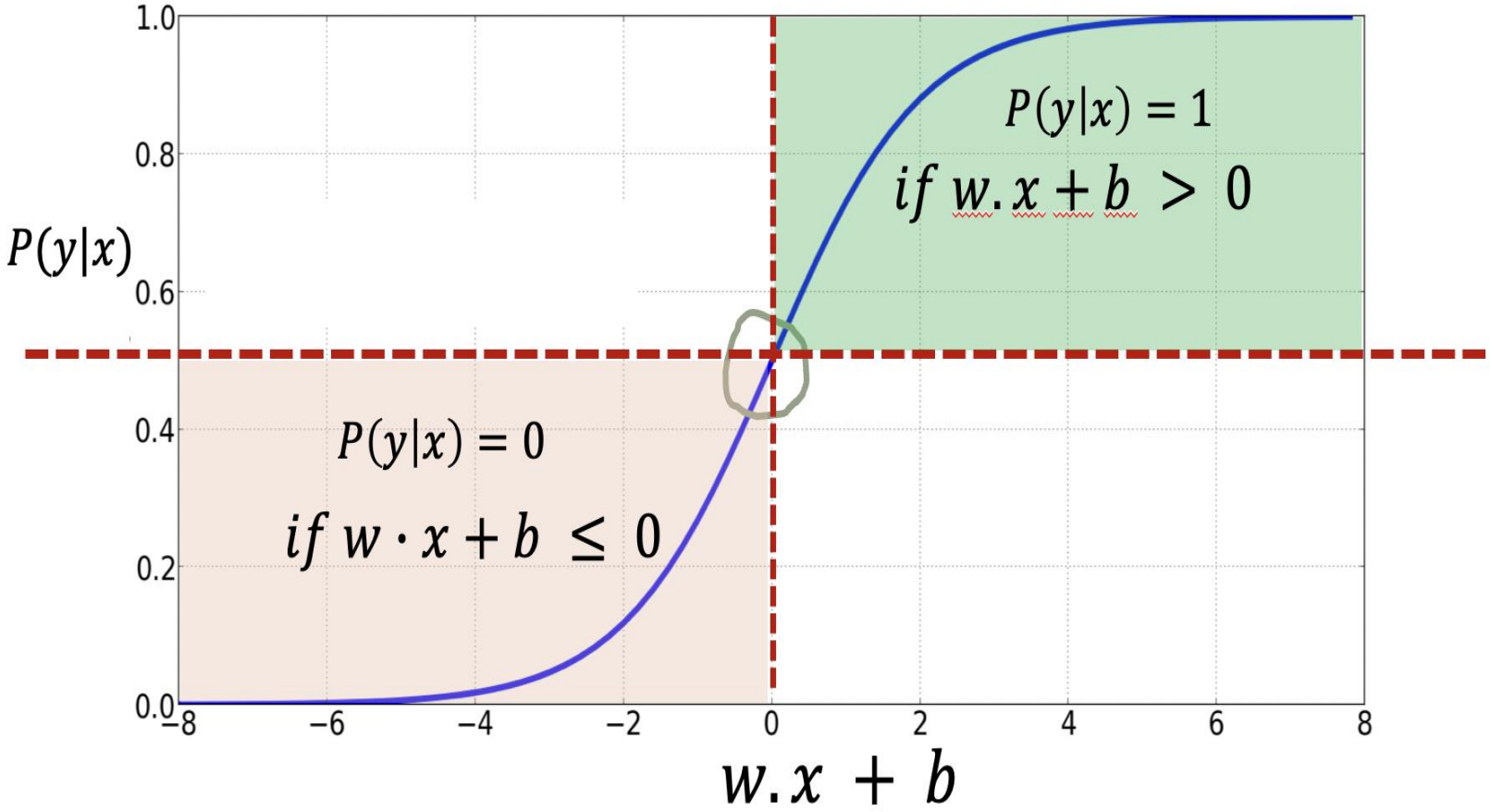
Let's calculate $P(y=0)$

Because $1 - \sigma(x) = \sigma(-x)$

Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases}$$

0.5 here is called the **decision boundary**



Turning a probability into a classifier

$$\hat{y} = \begin{cases} 1 & \text{if } P(y=1|x) > 0.5 \\ 0 & \text{otherwise} \end{cases} \qquad \begin{array}{l} \text{if } w \cdot x + b > 0 \\ \text{if } w \cdot x + b \leq 0 \end{array}$$

Logistic Regression: sentiment classification

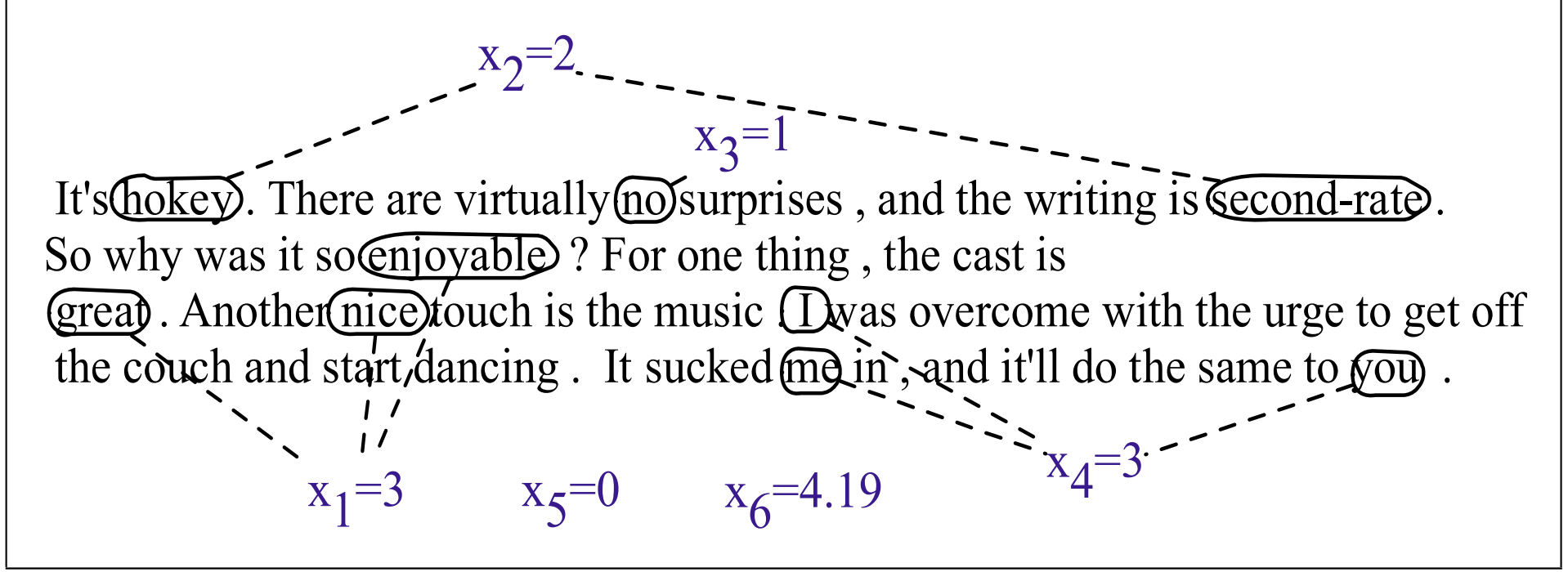
Sentiment example: does y=1 or y=0?

- **Example of movie review:**

- It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing. It sucked me in , and it'll do the same to you .

Chapter 4

Features Extraction



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	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

Chapter 4

Classifying sentiment for input x

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	$\begin{cases} 1 & \text{if } \text{“no”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	log(word count of doc)	$\ln(66) = 4.19$

- **Suppose weights (w)=** $[2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$

  bias (b) = 0.1

- **Let's compute $p(y = 1|x, \theta)$ and $p(y = 0|x, \theta)$**

Classifying sentiment for input x

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= s(w \cdot x + b) \\
&= s([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.19] + 0.1) \\
&= s(.833) \\
&= 0.70 \qquad (5.6)
\end{aligned}
$$

$$
\begin{aligned}
p(-|x) = P(Y=0|x) &= 1 - s(w \cdot x + b) \\
&= 0.30
\end{aligned}
$$

Classification in (**binary)** logistic regression: summary

- Given:
  - a set of classes: (+ sentiment,- sentiment)
  - a vector **x** of features [x_1, x_2, …, x_n]
    - x_1= count("awesome")
    - x_2 = log(number of words in review)
  - A vector **w** of weights [w_1, w_2, …, w_n]
    - w_i for each feature f_i

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

Wait, where did the *Weights* 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 $\boldsymbol{w}$ and $\boldsymbol{b}$ 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.

The two phases of logistic regression

- **Training**: we learn weights *w* and *b* using:
  - A loss function: **cross-entropy loss**
  - An optimization algorithm: **stochastic gradient descent**

- **Test (**also called **inference)** : Given a test example *x* we compute $p(y|x)$ using learned weights *w* and *b*, and return whichever label ($y = 1$ or $y = 0$) has higher probability

Learning: Cross-Entropy Loss

Loss Function: The distance between $\hat{y}$ and y

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

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

- from the true output:

$$y \quad [= either\ 0\ or\ 1]$$

- 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

- We choose the parameters $\boldsymbol{w}, \boldsymbol{b}$ that maximize
  - the log probability
  - of the true y labels in the training data
  - given the observations x

- A case of conditional maximum likelihood estimation

- The resulting loss function is the negative log likelihood loss, generally called **the cross-entropy loss**.

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:

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

- 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**

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

- **Minimize**

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

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

Cross-entropy loss

Let's see if this works for our sentiment example

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

- Let's first suppose the true label of this is y=1 (positive)

It's hokey . There are virtually no surprises , and the writing is second-rate . So why was it so enjoyable ? For one thing , the cast is great . Another nice touch is the music . I was overcome with the urge to get off the couch and start dancing . It sucked me in , and it'll do the same to you .

Let's see if this works for our sentiment example

- True value is y=1. How well is our model doing?

$$\begin{aligned} p(+|x) = P(Y=1|x) &= s(w\cdot x+b) \\ &= s([2.5,-5.0,-1.2,0.5,2.0,0.7]\cdot[3,2,1,3,0,4.19]+0.1) \\ &= s(.833) \\ &= 0.70 \end{aligned} \tag{5.6}$$

- Pretty well! What's the loss?

$$\begin{aligned} L_{\text{CE}}(\hat{y},y) &= -[y\log\sigma(w\cdot x+b)+(1-y)\log(1-\sigma(w\cdot x+b))] \\ &= -[\log\sigma(w\cdot x+b)] \\ &= -\log(.70) \\ &= .36 \end{aligned}$$

Let's see if this works for our sentiment example

- Suppose true value instead was y=0.

$$p(-|x) = P(Y=0|x) = 1 - s(w \cdot x + b)$$
$$= 0.30$$

- What's the loss?

$$\begin{aligned} L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))] \\ &= -[\log (1 - \sigma(w \cdot x + b))] \\ &= -\log(.30) \\ &= 1.2 \end{aligned}$$

Let's see if this works for our sentiment example

- The loss when model was right (if true y=1)

$$
\begin{aligned}
L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))] \\
&= -[\log \sigma(w \cdot x + b)] \\
&= -\log(.70) \\
&= .36
\end{aligned}
$$

- Is lower than the loss when model was wrong (if true y=0):

$$
\begin{aligned}
L_{\text{CE}}(\hat{y}, y) &= -[y \log \sigma(w \cdot x + b) + (1-y) \log (1 - \sigma(w \cdot x + b))] \\
&= -[\log (1 - \sigma(w \cdot x + b))] \\
&= -\log (.30) \\
&= 1.2
\end{aligned}
$$

- Sure enough, loss was bigger when model was wrong!

Log-Likelihood and Cross Entropy

- **Objective Function (Log-likelihood):**

  Maximize $\frac{1}{m}\sum_{=1}^{m}[y_i log\hat{y}_i + (1-y_i) log(1-\hat{y}_i)]$

- **Cross-entropy:**

  Minimize $-\frac{1}{m}\sum_{=1}^{m}[y_i log\hat{y}_i + (1-y_i) log(1-\hat{y}_i)]$

An optimization algorithm: **Stochastic Gradient Descent**
is used to optimize the weights
for logistic regression
also for neural networks

Optimization: Stochastic Gradient Descent

Our Goal: Minimize Loss

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

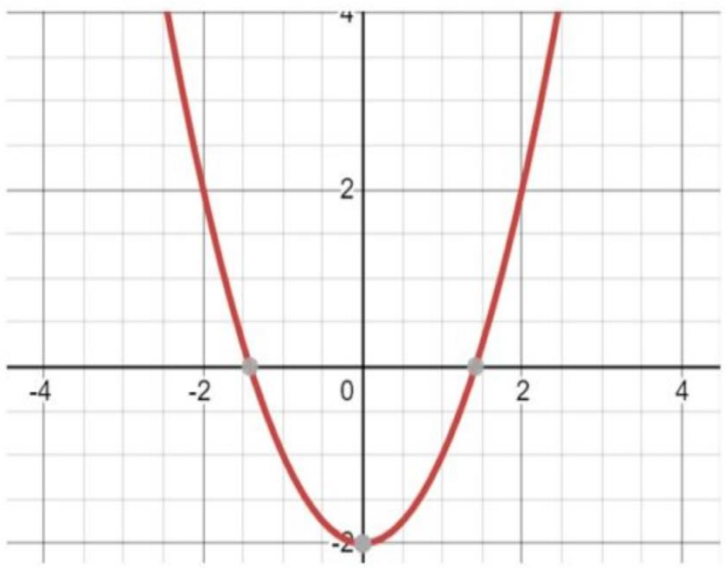
- We want the weights that minimize the loss, averaged over all examples:

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

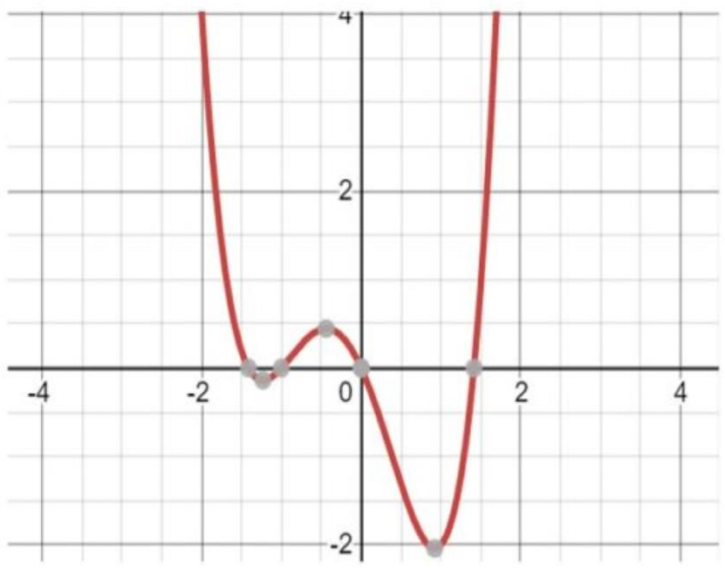
Cost Function

Our Goal: Minimize 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**



Convex Function

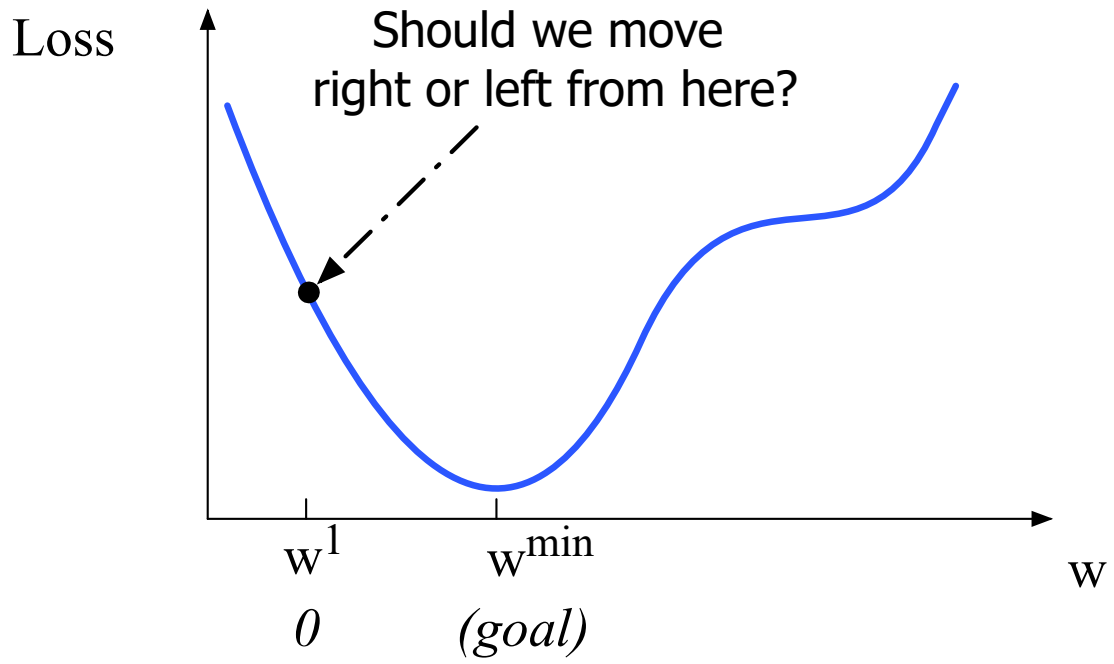


Non-Convex Function

Source

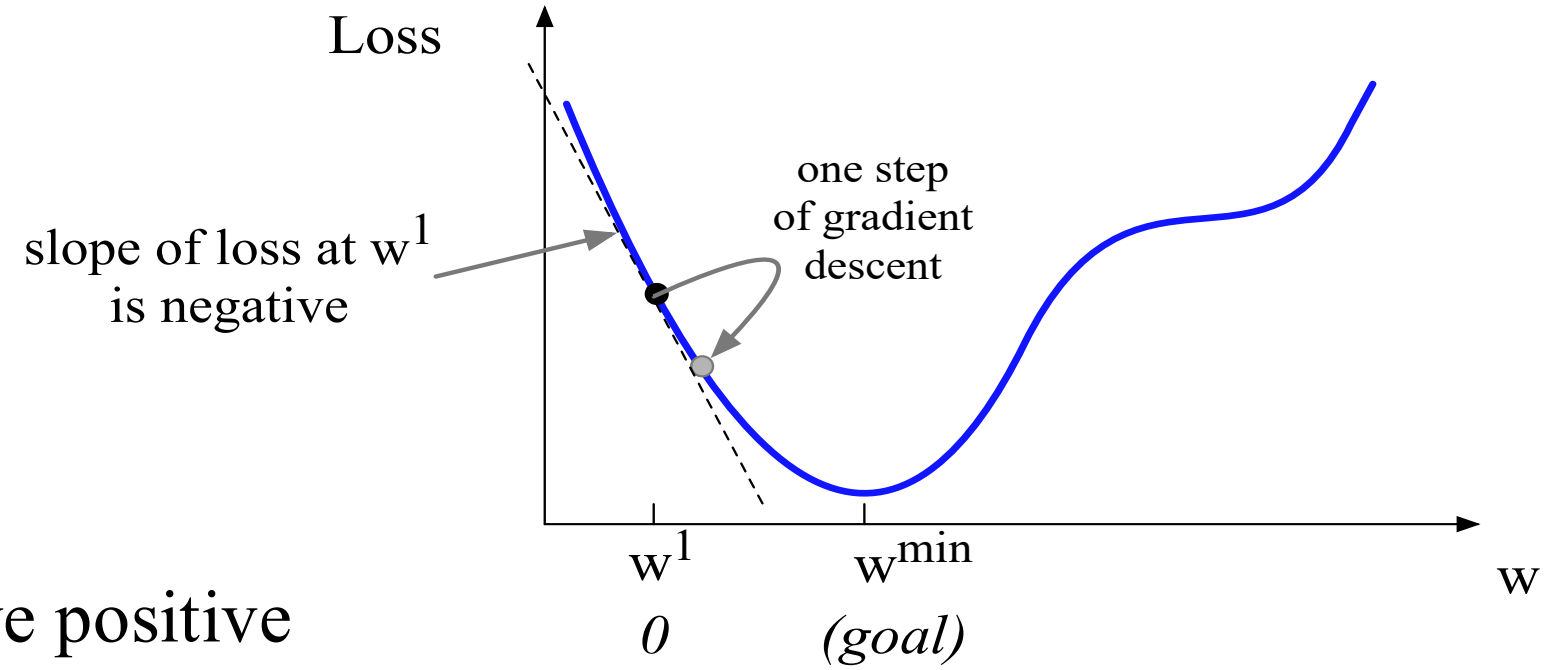
Let's first visualize for a single scalar w

- Q: Given current $\boldsymbol{w}$,should we make it bigger or smaller?
- A: Move $\boldsymbol{w}$ 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

Gradient

- The **gradient** of a function shows the direction in which the function increases the fastest.
- Gradient of $f(x_1, x_2, \ldots, x_n)$ $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$
- **Gradient Descent**: Find the gradient of the loss function at the current point and move in the **opposite** direction.

The Gradient

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

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

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

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

What are these partial derivatives for logistic regression?

- The loss function

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

- The elegant derivative of this function is:

$$\begin{aligned} \frac{\partial L_{\text{CE}}(\hat{y}, y)}{\partial \mathbf{w}_j} &= [\sigma(\mathbf{w} \cdot \mathbf{x} + b) - y]\mathbf{x}_j \\ &= (\hat{y} - y)\mathbf{x}_j \end{aligned}$$

Chapter 4

Stochastic Gradient Descent

- Stochastic gradient descent is an ***online algorithm*** that minimizes the loss function by computing its gradient after each training example, and nudging θ in the right direction (the opposite direction of the gradient).

- An ***"online algorithm"*** is one that processes its input example by example, rather than waiting until it sees the entire input.

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 $x^{(1)}$, $x^{(2)}$,..., $x^{(m)}$
y is the set of training outputs (labels) $y^{(1)}$, $y^{(2)}$,..., $y^{(m)}$

$\theta \leftarrow 0$

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

Figure 5.6 The stochastic gradient descent algorithm. Step 1 (computing the loss) is used mainly to report how well we are doing on the current tuple; we don't need to compute the loss in order to compute the gradient. The algorithm can terminate when it converges (or when the gradient norm $< \epsilon$), or when progress halts (for example when the loss starts going up on a held-out set).

Working through an Example

Example of gradient descent

- Update step for updating θ is: $q_{t+1} = q_t - h\nabla L(f(x;q),y)$

  - Observation $x = [x_1, x_2]$ such as: $x_1 = 3; \quad x_2 = 2$
  - x_1 counts of positive words in a tweet
  - x_2 counts of negative words in a tweet

- Let's assume:
  - *weights and bias are initialized as* $w_1 = w_2 = b = 0;$
  - Learning rate η = 0.1;

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:

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

$\eta = 0.1;$

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

Note that enough negative examples would eventually make w_2 negative

Let's recap: The two phases of logistic regression

- **Training**: we learn weights w and bias b using **stochastic gradient descent** and **cross-entropy loss**.

- **Test**: Given a test example x we compute $p(y|x)$ using learned weights w and b, and return whichever label ($y = 1\ or\ y = 0$) is higher probability.

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.

Mini-batch Training

- **Stochastic gradient descent (SGD)** 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)

Multinomial Logistic Regression

Multinomial Logistic Regression

- Often we need more than 2 classes
  - Positive/negative/neutral
  - Parts of speech (noun, verb, adjective, adverb, preposition, etc.)
  - Classify emergency SMSs into different actionable classes

- If >2 classes we use **multinomial logistic regression**
  = Softmax regression
  = Multinomial logit
  = (defunct names : Maximum entropy modeling or MaxEnt

- So "logistic regression" will just mean binary (2 output classes)

Multinomial Logistic Regression

- The probability of everything must still sum to 1

$$P(positive|doc) + P(negative|doc) + P(neutral|doc) = 1$$

- Need a generalization of the **sigmoid** called the **softmax**
  - Takes a vector $z = [z_1, z_2, ..., z_k]$ of k arbitrary values
  - Outputs a probability distribution
    - each value in the range [0,1]
    - all the values summing to 1

The **softmax** function

- Turns a vector $z = [z_1, z_2, \ldots, z_k]$ of k arbitrary values into probabilities:

$$\text{softmax}(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \quad 1 \leq i \leq k$$

- The denominator $\sum_{j=1}^{k} \exp(z_j)$ is used to normalize all the values into probabilities.

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

The **softmax** function: Example

- Turns a vector $z = [z_1, z_2, \ldots, z_k]$ of *k* arbitrary values into probabilities:

$$z = [0.6, 1.1, -1.5, 1.2, 3.2, -1.1]$$

$$\text{softmax}(z) = \left[\frac{\exp(z_1)}{\sum_{i=1}^{k} \exp(z_i)}, \frac{\exp(z_2)}{\sum_{i=1}^{k} \exp(z_i)}, \ldots, \frac{\exp(z_k)}{\sum_{i=1}^{k} \exp(z_i)} \right]$$

$$= [0.055, 0.090, 0.0067, 0.10, 0.74, 0.010]$$

Softmax in multinomial logistic regression

$$p(y = c|x) = \frac{\exp(w_c \cdot x + b_c)}{\sum_{j=1}^{k} \exp(w_j \cdot x + b_j)}$$

- Input is still the dot product between weight vector $\boldsymbol{w}$ and input vector $\boldsymbol{x}$
- But now we will need separate weight vectors for each of the $\boldsymbol{K}$ classes.

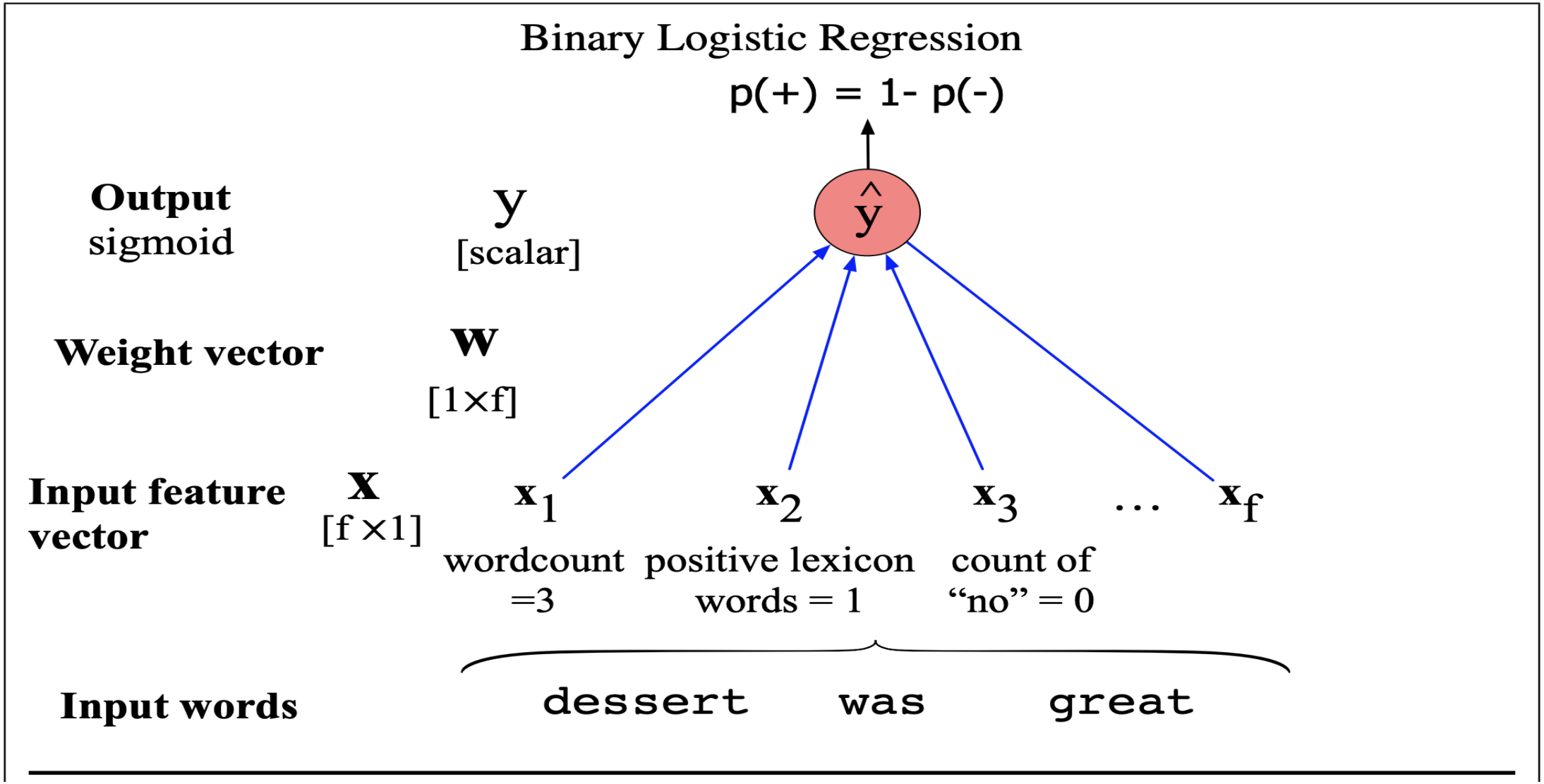
Features in ***binary*** versus ***multinomial*** logistic regression

- ***Binary***: positive weight → y=1 neg weight → y=0

$$x_5 = \begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$$

- ***Multinominal***: separate weights for each class:

Feature	Definition	$w_{5,+}$	$w_{5,-}$	$w_{5,0}$
$f_5(x)$	$\begin{cases} 1 & \text{if } \text{“!”} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	3.5	3.1	−5.3



$$\hat{Y} = sigmoid\,(W.X + b)$$

Chapter 5

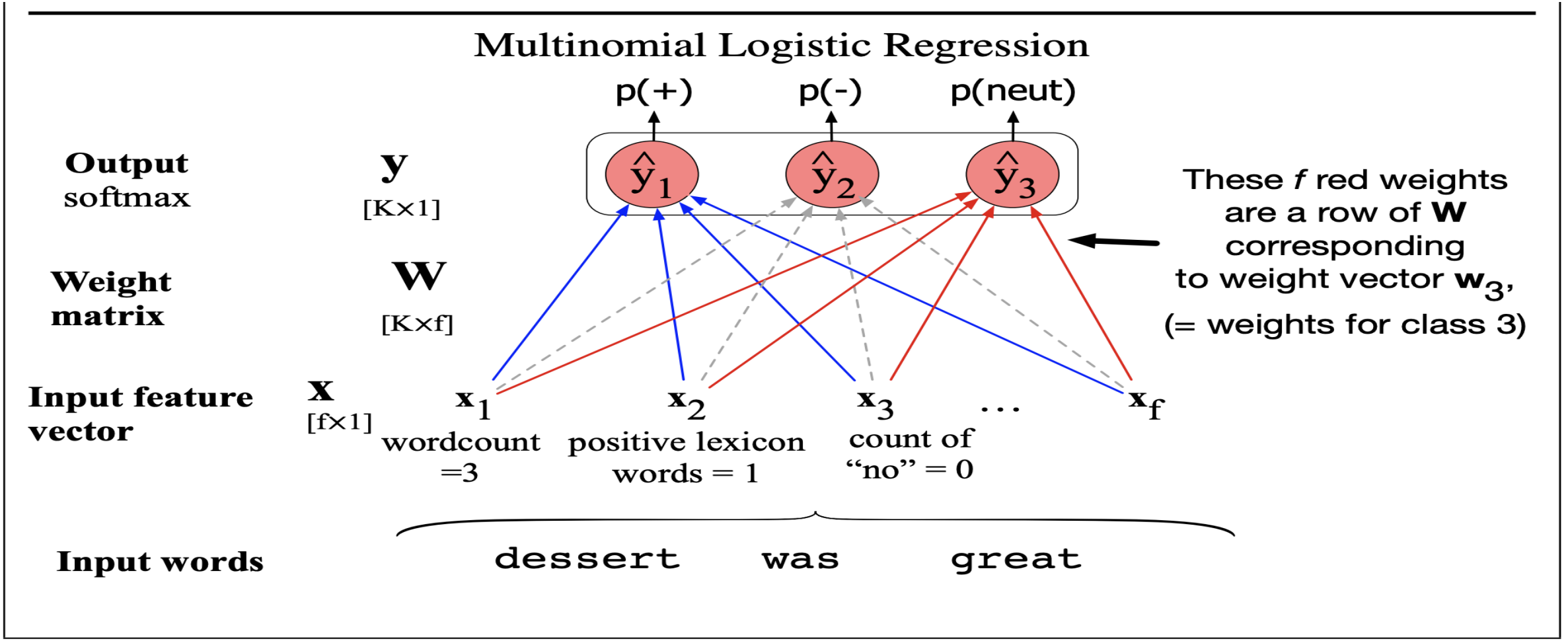


Figure 5.3 Binary versus multinomial logistic regression. Binary logistic regression uses a single weight vector **w**, and has a scalar output $\hat{y}$. In multinomial logistic regression we have K separate weight vectors corresponding to the K classes, all packed into a single weight matrix **W**, and a vector output $\hat{\mathbf{y}}$.

$$\hat{Y} = softmax\,(W.X + b)$$

Chapter 5

Summary

- This chapter introduced the **logistic regression** model of **classification**.

- **Logistic regression** is a supervised machine learning classifier that extracts real-valued features from the input, multiplies each by a weight, sums them, and passes the sum through a **sigmoid** function to generate a probability. A threshold is used to make a decision.

- Logistic regression can be used with two classes (e.g., positive and negative sentiment) or with multiple classes (**multinomial logistic regression**, for example for topic text classification, part-of-speech labeling, etc.)

Summary

- Binary classification uses **sigmoid function** while Multinomial logistic regression uses the **softmax function** to compute probabilities.

- The weights (vector w and bias b) are learned from a labeled training set via a loss function, such as **the cross-entropy loss**, that must be minimized.

- Minimizing this loss function is a **convex optimization problem**, and iterative algorithms like **gradient descent** are used to find the optimal weights.

- **Logistic regression** is also one of the most useful analytic tools, because of its ability to transparently study the importance of individual features.

Homework Exercise

https://drive.google.com/file/d/1hoeD8-aiMcZEj6oKrnm7mXQlCTkxQ21M/view?usp=sharing

Reading

- Please refer to this document : Chapter 4
- https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- https://machinelearningmastery.com/softmax-activation-function-with-python/
- https://www.bmc.com/blogs/bias-variance-machine-learning/
- https://www.baeldung.com/cs/understanding-gradient-descent
- https://www.freecodecamp.org/news/gradient-descent-machine-learning-algorithm-example/

Next Class

- **Lecture 7:** Neural Networks and Word Embeddings