

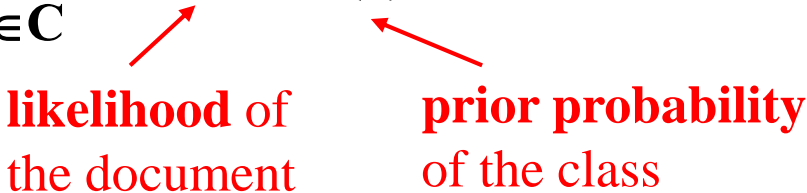
# **Logistic Regression**

## *A Discriminative Classifier*

# Generative and Discriminative Classifiers

- The naive Bayes assigns a class  $c$  to a document  $d$  *not by directly computing  $P(c/d)$*  but by computing a likelihood and a prior.

$$\hat{c} = \operatorname{argmax}_{c \in C} P(d|c) P(c)$$

  
**likelihood of the document**      **prior probability of the class**

- A **generative model** (like naive Bayes) makes use of *likelihood term*, which expresses *how to generate the features of a document if we knew it was of class  $c$* .
- A **discriminative model** (like logistic regression) in the text categorization scenario *attempts model to directly compute  $P(c/d)$* .
  - It will learn to assign high weight to document features that directly improve its ability to discriminate between possible classes, even if it couldn't generate an example of one of the classes.

# Generative and Discriminative Classifiers

- **Discriminative (conditional) models** are widely used in NLP (and ML generally).
  - They give high accuracy performance
  - They make it easy to incorporate lots of linguistically important features
  - They allow automatic building of language independent NLP modules
- In text classification task, we have some data  $\{(d, c)\}$  of paired observations  $d$  and hidden classes  $c$ .
- **Generative (joint) models** place probabilities over both observed data and the hidden stuff (generate the observed data from hidden stuff).
  - *some generative models*: n-gram models, Naive Bayes classifiers, hidden Markov models, probabilistic context-free grammars, ...
- **Discriminative (conditional) models** take the data as given, and put a probability over hidden structure given the data.
  - *some discriminative models*: logistic regression, maximum entropy models, conditional random fields, SVMs, perceptron, ...

# Components of A Probabilistic Machine Learning Classifier

- Machine learning classifiers require a training corpus of observations input/output pairs  $(x,y)$ .
  - In text classification, document/class  $(d,c)$  pairs.
- A machine learning system for classification has four components:
  1. **A feature representation of the input.** For each input observation  $x$ , this will be a vector of features  $[x_1, x_2, \dots, x_n]$ .
  2. **A classification function that computes  $\hat{y}$  (the estimated class)** via  $P(y|x)$ . We will use *sigmoid function* in logistic regression.
  3. **An objective function for learning**, usually involving minimizing error on training examples. We will use *cross-entropy loss function* in logistic regression.
  4. **An algorithm for optimizing the objective function.** We will use *stochastic gradient descent algorithm* in logistic regression.

# Logistic Regression

- The goal of **binary logistic regression** is to train a classifier that can make a binary decision about the class of a new input observation.
- **training:** we train the system to learn a vector of weights for features and a bias term using stochastic gradient descent and the cross-entropy loss.
  - Each weight  $w_i$  is a real number, and is associated with one of the input features  $x_i$ . The weight  $w_i$  represents how important that input feature is to the classification decision, and can be positive (meaning the feature is associated with the class) or negative (meaning the feature is not associated with the class).
    - In a sentiment task the word *awesome* to have a high positive weight, and *abysmal* bias term to have a very negative weight.
  - The bias term is another real number that's added to the weighted inputs.
- **test:** for given a test example  $x$ , we compute  $P(y|x)$  and return the higher probability label  $y=1$  (member of class) or  $y=0$  (not member of class).

# Logistic Regression

## *weighted sum of the evidence for the class*

- To make a decision on a test instance (after we've learned the weights in training) the classifier first multiplies each  $x_i$  by its weight  $w_i$ , sums up the weighted features, and adds the bias term  $b$ .
- The resulting number  $z$  expresses the **weighted sum of the evidence for the class**.

$$z = \left( \sum_{i=1}^n w_i x_i \right) + b$$

- In short, we can write the weighted sum of the evidence for the class as follows:

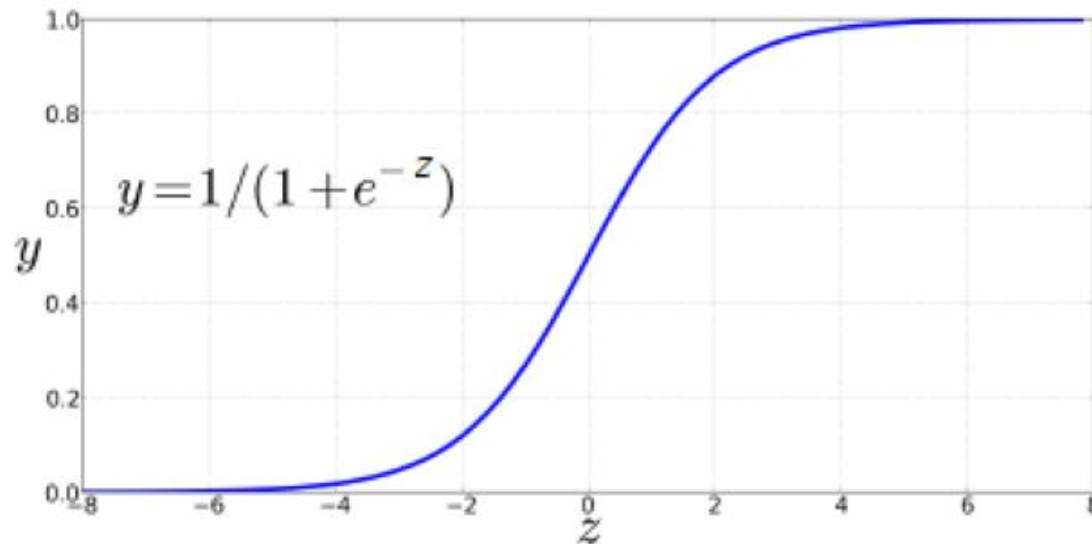
$$z = w \cdot x + b$$

- The weighted sum of the evidence  $z$  is *not a probability value*, it ranges from  $-\infty$  to  $+\infty$ .

# Logistic Regression

## *sigmoid function*

- To create a probability, we'll pass  $z$  through the **sigmoid function**,  $\sigma(z)$ .
- The **sigmoid function** is also called the **logistic function**.
- The sigmoid function  $y = \frac{1}{1+e^{-z}}$  takes a real value and maps it to the range  $[0,1]$ .



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

# Logistic Regression

## *sigmoid function*

- The sigmoid function has a number of advantages:
  - It maps a real-valued number into range [0,1], which is just what we want for a probability.
  - It tends to squash outlier values toward 0 or 1.
  - And it's differentiable, This will be handy for learning.
- If we apply the sigmoid to the sum of the weighted features, we get a number between 0 and 1. To make it a probability, we just need to make sure that the two cases,  $P(y=1)$  and  $P(y=0)$ , sum to 1.

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

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



# Logistic Regression

## *decision boundary for classification*

- We have an algorithm that given an instance  $x$  computes the probability  $P(y=1|x)$ .
- When  $P(y=1|x)$  is more than 0.5 (**decision boundary**), the **estimated class** will be 1 (indicating that the instance  $x$  belongs to the class).

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

# Logistic Regression

## *Example: sentiment classification*

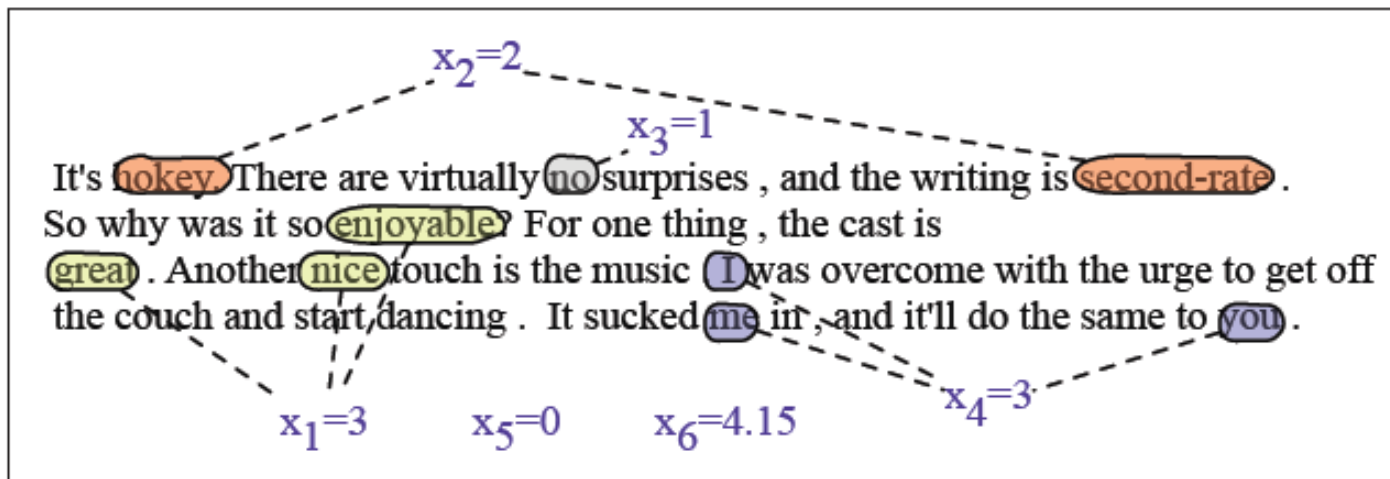
- Suppose we are doing binary sentiment classification on movie review text, and we want to assign the sentiment class + or - to a review document *doc*.
- Each input observation (document) will be represented by 6 features of the input:

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

# Logistic Regression

## *Example: sentiment classification*

- Let's assume for the moment that we've already learned a real-valued weight for each of 6 features.
  - The 6 weights corresponding to the 6 features are  $[2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$ , and the bias term  $b = 0.1$ .
  - For example, the weight  $w_1$  indicates how the number of positive lexicon words (great, nice, enjoyable, etc.) is important to a positive sentiment decision.
- A sample mini test document showing the extracted features in the vector  $x$ .



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

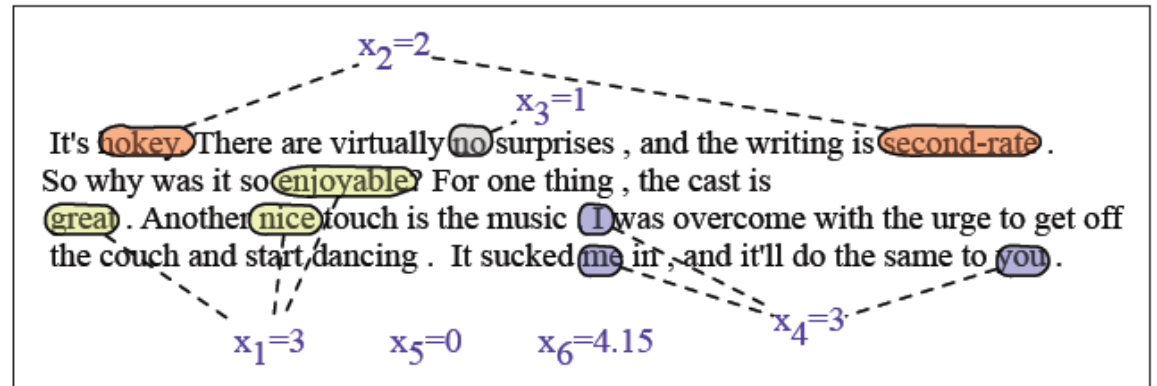
# Logistic Regression

## *Example: sentiment classification*

- Given these 6 features and the input review  $x$ ,  $P(+|x)$  and  $P(-|x)$  can be computed:

$$\begin{aligned} p(+|x) &= P(Y = 1|x) = \sigma(w \cdot x + b) \\ &= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\ &= \sigma(1.805) \\ &= 0.86 \end{aligned}$$

$$\begin{aligned} p(-|x) &= P(Y = 0|x) = 1 - \sigma(w \cdot x + b) \\ &= 0.14 \end{aligned}$$



# Logistic Regression: Learning

- How are the weights  $w$  and bias  $b$  (the parameters of the model) learned?
  - Logistic regression is an instance of supervised classification in which we know the correct label  $y$  (either 0 or 1) for each observation  $x$ .
  - The estimated value  $\hat{y}$  is the system's estimate of the true  $y$ .
  - We want to learn parameters ( $w$  and  $b$ ) that make  $\hat{y}$  for each training observation as close as possible to the true  $y$ .
- In order to learn the parameters of the model, we require two things:
  1. **A metric for how close the current label ( $\hat{y}$ ) is to the true gold label  $y$ .**
    - The distance between the system output and the gold truth is called as the **loss function** or **the cost function**.
    - The loss function that is commonly used for logistic regression is the **cross-entropy loss**.
  2. **An optimization algorithm for iteratively updating the weights so as to minimize this loss function.**
    - The standard algorithm for this is **gradient descent** (or **stochastic gradient descent**)

# Logistic Regression: Learning

## *cross-entropy loss function*

- We need a loss function that expresses, for an observation  $x$ , how close the classifier output ( $\hat{y} = \sigma(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$ ) is to the correct output ( $y$ , which is 0 or 1):

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

- For logistic regression, we will use a loss function known as **conditional maximum likelihood estimation** that prefers the correct class labels of the training example to be more likely.
  - We choose the parameters  $w$ ,  $b$  that **maximize the log probability of the true  $y$  labels in the training data** given the observations  $x$ .
  - The resulting loss function is the negative log likelihood loss, generally called the **cross entropy loss**.
    - We do not prefer to use the *mean squared error* as a loss function because it is harder to optimize for probabilistic classification (it is not convex).

# Logistic Regression: Learning

## *cross-entropy loss function*

- We'd like to learn weights that maximize the probability of the correct label  $p(y|x)$ .
- Since there are only two discrete outcomes (1 or 0), we can express the probability  $p(y|x)$  as follows:

$$p(y|x) = \hat{y}^y (1 - \hat{y})^{1-y} \quad \begin{array}{ll} p(y|x) = \hat{y} & \text{when } y=1 \text{ and} \\ p(y|x) = 1-\hat{y} & \text{when } y=0 \end{array}$$

- Take the log of both sides (whatever values maximize a probability will also maximize the log of the probability):

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

- This describes a log likelihood that should be maximized.

# Logistic Regression: Learning

## *cross-entropy loss function*

- To obtain **cross-entropy loss function for one example**, we change the sign in the equation:

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

- Plug in the definition of  $\hat{y} = \sigma(w \cdot x + b)$ :

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

- A perfect classifier would assign probability 1 to the correct outcome ( $y=1$  or  $y=0$ ) and probability 0 to the incorrect outcome.
- Higher  $\hat{y}$  (close to 1)  $\rightarrow$  better classifier; Lower  $\hat{y}$  is (close to 0)  $\rightarrow$  worse classifier.
  - The negative log of this probability is a convenient loss metric since it goes from 0 to infinity.
  - This loss function also insures that as probability of the correct answer is maximized, the probability of the incorrect answer is minimized;



# Logistic Regression: Learning

## *cross-entropy loss function*

- We will extend our loss function from one example to the whole training set.
- We make the assumption that the training examples are independent, and  $x^{(i)}$  and  $y^{(i)}$  means features and label of  $i^{\text{th}}$  training example:

$$\begin{aligned}\log p(\text{training labels}) &= \log \prod_{i=1}^m p(y^{(i)} | x^{(i)}) \\ &= \sum_{i=1}^m \log p(y^{(i)} | x^{(i)}) \\ &= - \sum_{i=1}^m L_{CE}(\hat{y}^{(i)}, y^{(i)})\end{aligned}$$

# Logistic Regression: Learning

*cross-entropy loss function: **cost function***

- The **cost function for the whole dataset** is the average loss for each example:

$$\begin{aligned} \text{Cost}(w, b) &= \frac{1}{m} \sum_{i=1}^m L_{CE}(\hat{y}^{(i)}, y^{(i)}) \\ &= -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log (1 - \sigma(w \cdot x^{(i)} + b)) \end{aligned}$$

- We want to minimize this cost function.
- We want to find weights  $w$  and bias term  $b$  (optimal weights) which minimize this cost function.

# Logistic Regression: Learning

## *Gradient Descent*

- Our goal with the **gradient descent** is to find the **optimal weights** which minimize the **loss function**.
- The loss function  $L$  is *parameterized by the weights*, which we'll refer to as  $\theta$  (in the logistic regression  $\theta = w, b$ ), and the **optimal weights** are:

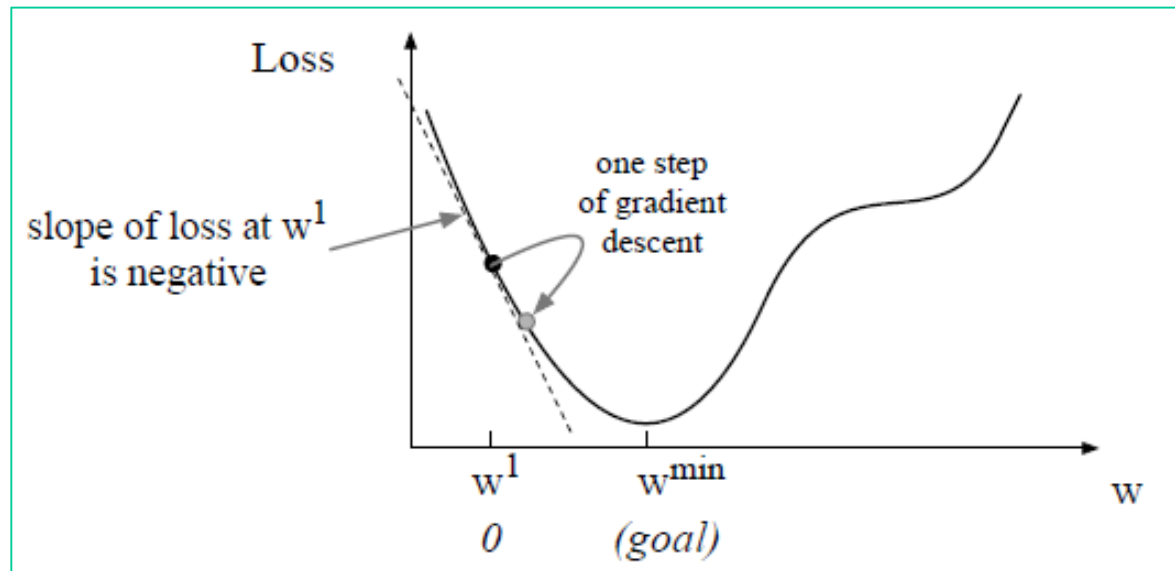
$$\hat{\theta} = \operatorname{argmin}_{\theta} \frac{1}{m} \sum_{i=1}^m L_{CE}(y^{(i)}, x^{(i)}; \theta)$$

- **Gradient descent** is a method that finds a minimum of a function by figuring out in which direction (in the space of the parameters  $\theta$ ) the function's slope is rising the most steeply, and moving in the opposite direction.
- For logistic regression, the *loss function is conveniently convex*.
  - A convex function has just one minimum; there are no local minima to get stuck in, so **gradient descent** starting from any point is guaranteed to find the minimum.

# Logistic Regression: Learning

## *Gradient Descent*

- Although the **gradient descent algorithm** is designed for *direction vectors*, let's first consider the algorithm for a *single value*  $w$ .
- After the random initial value  $w^1$  (normally 0), the **gradient descent algorithm** tell us our direction at the next iteration to reach the minimum.
  - move  $w$  in positive direction (to right) when slope of loss is negative (making  $w^2$  bigger than  $w^1$ ) or
  - move  $w$  in negative direction (to left) when slope of loss is positive (making  $w^2$  smaller than  $w^1$ ).



# Logistic Regression: Learning

## *Gradient Descent*

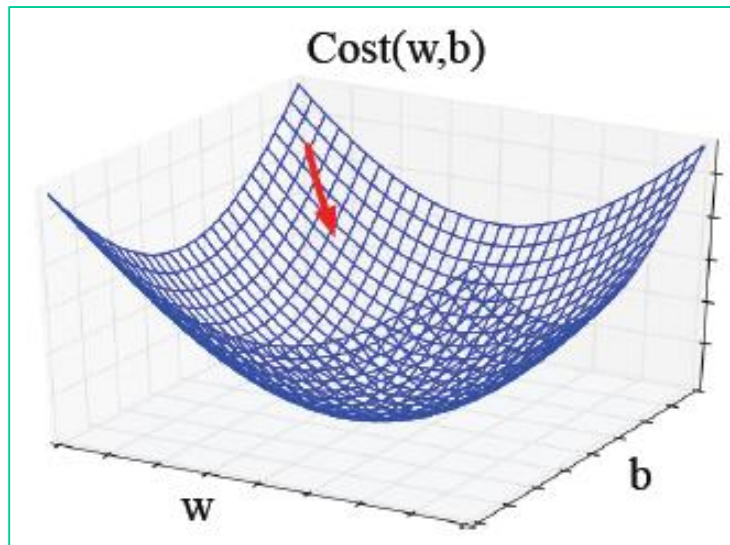
- In order to reach to the minimum, the **gradient descent algorithm finds the gradient of the loss function** at the current point and it **moves in the opposite direction**.
- The **gradient** is the increase direction, and it is equal to the slope  $\frac{d}{dw} f(\mathbf{x}; \mathbf{w})$
- The **magnitude of the amount to move in gradient descent** is the value of the slope  $\frac{d}{dw} f(\mathbf{x}; \mathbf{w})$  weighted by a **learning rate  $\eta$** .
- The change we make in our parameter is the learning rate times the gradient

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

# Logistic Regression: Learning

## *Gradient Descent*

- When we have a weight vector, we want to know where in the N-dimensional space (N parameters in  $\theta$ ) we should move.
- The **gradient** is just such a vector which expresses the *directional components of the sharpest slope along each of those N dimensions*.
- Visualization of the gradient vector in two dimensions w and b.



# Logistic Regression: Learning

## *Gradient Descent*

- For each variable  $w_i$  in  $w$ , the **gradient** will have *a component that tells us the slope with respect to that variable*.
  - “How much would a small change in that variable  $w_i$  influence the total loss function  $L$ ?”
- In each dimension  $w_i$ , we express the slope as a **partial derivative**  $\frac{\partial}{\partial w_i}$  **of the loss function**.
- The **gradient** is then defined as **a vector of these partials**.

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

where  $f(x; \theta)$  is  $\hat{y}$

# Logistic Regression: Learning

## *Gradient Descent*

- The equation for updating  $\theta$  based on the **gradient** is:

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

- In order to update  $\theta$ , we need a definition for the gradient  $\nabla L(f(x; \theta), y)$ .
- The cross-entropy loss function for logistic regression is:

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

- The partial derivative of this function for one observation vector  $x$  is:

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



# Logistic Regression: Learning

## *Gradient Descent*

- The *loss for an entire dataset is the average loss over the  $m$  examples:*

$$Cost(w, b) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log (1 - \sigma(w \cdot x^{(i)} + b))$$

- The *gradient for multiple data points is the sum of the individual gradients:*

$$\frac{\partial Cost(w, b)}{\partial w_j} = \sum_{i=1}^m [\sigma(w \cdot x^{(i)} + b) - y^{(i)}] x_j^{(i)}$$

# Logistic Regression: Learning

## *Stochastic Gradient Descent Algorithm*

- **Stochastic gradient descent** minimizes the loss function by computing its gradient after each training example, and nudging  $\theta$  in the right direction.

```
function STOCHASTIC GRADIENT DESCENT( $L()$ ,  $f()$ ,  $x$ ,  $y$ ) returns  $\theta$ 
  # where:  $L$  is the loss function
  #  $f$  is a function parameterized by  $\theta$ 
  #  $x$  is the set of training inputs  $x^{(1)}, x^{(2)}, \dots, x^{(n)}$ 
  #  $y$  is the set of training outputs (labels)  $y^{(1)}, y^{(2)}, \dots, y^{(n)}$ 

 $\theta \leftarrow 0$ 
repeat  $T$  times
  For each training tuple  $(x^{(i)}, y^{(i)})$  (in random order)
    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  $y^{(i)}$ ?
     $g \leftarrow \nabla_{\theta} L(f(x^{(i)}; \theta), y^{(i)})$  # How should we move  $\theta$  to maximize loss?
     $\theta \leftarrow \theta - \eta g$  # go the other way instead
return  $\theta$ 
```

# Logistic Regression: Learning

## *Gradient Descent: Example – sentiment classification*

- We will give a single step of the gradient descent algorithm for a simplified version of sentiment classification task.
  - It sees a single observation  $x$ , whose correct value is  $y = 1$  (this is a positive review), and
  - There are only two features:
    - $x_1 = 3$  (count of positive lexicon words)
    - $x_2 = 2$  (count of negative lexicon words)
- Initial weights and bias in  $\theta^1$ , and the learning rate  $\eta$  are:

$$w_1 = w_2 = b = 0$$

$$\eta = 0.1$$

# Logistic Regression: Learning

## *Gradient Descent: Example – sentiment classification*

- The single update step:

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

- Since there are three parameters, so the gradient vector has 3 dimensions, for  $w_1$ ,  $w_2$ , and  $b$ . We can compute the gradient as follows:

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

- we compute  $\theta^2$  by moving  $\theta^1$  in the opposite direction from the gradient:

$$\theta^2 = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} -1.5 \\ -1.0 \\ -0.5 \end{bmatrix} = \begin{bmatrix} .15 \\ .1 \\ .05 \end{bmatrix} \quad \mathbf{Q}^{t+1} = \begin{bmatrix} w_1 \\ w_2 \\ b \end{bmatrix} - \eta \begin{bmatrix} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{bmatrix}$$

$\mathbf{Q}^t$                       **gradient**

# Logistic Regression: Learning

## *Gradient Descent: Example – sentiment classification*

$$Q^{t+1} = \begin{matrix} Q^t \\ \left[ \begin{array}{c} w_1 \\ w_2 \\ b \end{array} \right] \end{matrix} - \eta \begin{matrix} \text{gradient} \\ \left[ \begin{array}{c} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{array} \right] \end{matrix}$$

		Initial Weights: $\Theta_1$	Ex1	gradient	Weights: $\Theta_2$
weights	examples				
w1	x1	<b>0.000</b>	3.000	-1.500	<b>0.150</b>
w2	x2	<b>0.000</b>	2.000	-1.000	<b>0.100</b>
b	x0 (=1)	<b>0.000</b>	1.000	-0.500	<b>0.050</b>
	y: class		1.000		
w.x+b			0.000		
sig(w.x+b)			0.500		
<b>eta</b>	<b>0.1</b>				

# Logistic Regression: Learning

## *Gradient Descent: Example – sentiment classification*

$$Q^{t+1} = \begin{matrix} Q^t \\ \left[ \begin{array}{c} w_1 \\ w_2 \\ b \end{array} \right] \end{matrix} - \eta \begin{matrix} \text{gradient} \\ \left[ \begin{array}{c} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{array} \right] \end{matrix}$$

weights	examples	Weights: $\Theta_2$	Ex2	gradient	Weights: $\Theta_3$	Ex3	gradient	Weights: $\Theta_4$
w1	x1	<b>0.150</b>	4.000	-1.372	<b>0.287</b>	0.000	0.000	<b>0.287</b>
w2	x2	<b>0.100</b>	0.000	0.000	<b>0.100</b>	4.000	2.475	<b>-0.148</b>
b	x0 (=1)	<b>0.050</b>	1.000	-0.343	<b>0.084</b>	1.000	0.619	<b>0.022</b>
	y: class		1.000			0.000		
w.x+b			0.650			0.484		
sig(w.x+b)			0.657			0.619		
<b>eta</b>	<b>0.1</b>							

# Logistic Regression: Learning

## *Gradient Descent: Example – sentiment classification*

$$Q^{t+1} = \begin{matrix} Q^t \\ \left[ \begin{array}{c} w_1 \\ w_2 \\ b \end{array} \right] \end{matrix} - \eta \begin{matrix} \text{gradient} \\ \left[ \begin{array}{c} (\sigma(w \cdot x + b) - y)x_1 \\ (\sigma(w \cdot x + b) - y)x_2 \\ \sigma(w \cdot x + b) - y \end{array} \right] \end{matrix}$$

weights	examples	Weights: Θ4	Ex4	gradient	Weights: Θ5	Ex5	gradient	Weights: Θ6
w1	x1	<b>0.287</b>	3.000	-0.971	<b>0.384</b>	0.000	0.000	<b>0.384</b>
w2	x2	<b>-0.148</b>	1.000	-0.324	<b>-0.115</b>	3.000	1.284	<b>-0.243</b>
b	x0 (=1)	<b>0.022</b>	1.000	-0.324	<b>0.055</b>	1.000	0.428	<b>0.012</b>
	y: class		1.000			0.000		
w.x+b			0.737			-0.291		
sig(w.x+b)			0.676			0.428		
<b>eta</b>	<b>0.1</b>							

# Logistic Regression: Learning

## *Overfitting and Regularization*

- The *weights for features will attempt to perfectly fit details of the training set, modeling noisy factors that just accidentally correlate with the class.*
- This problem is called **overfitting**.
  - A good model should be able to **generalize** well from the training data to the unseen test set, but a model that **overfits** will have poor generalization.
- One way to **avoid overfitting**, adding a **regularization term** to the objective function:

$$\hat{w} = \operatorname{argmax}_w \sum_{i=1}^m \log P(y^{(i)} | x^{(i)}) - \alpha R(w)$$

- The new component,  $R(w)$  is called a regularization term, and is used to penalize large weights.



# Multinomial Logistic Regression

- In some classification tasks, we need more than two classes.
- In such cases, we can use **multinomial logistic regression**, also it is also called as **softmax regression** (or **maxent classifier**).
- In **multinomial logistic regression**, the target  $y$  is a variable that ranges over more than two classes.
- We want to know the probability of  $y$  being in each potential class  $c \in C$ ,  $p(y=c|x)$ .
- The **multinomial logistic classifier** uses a generalization of the sigmoid function, called as the **softmax function**, to compute the probability  $p(y=c|x)$ .
  - The **softmax function** takes a vector  $z = [z_1, z_2, \dots, z_k]$  of  $k$  arbitrary values and maps them to a probability distribution, with each value in the range  $(0,1]$ , and all the values summing to 1.

# Logistic Regression: Summary

- **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.
- 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*.
- **Regularization** is used to avoid overfitting.
- **Logistic regression** can be used with two classes or with multiple classes (**multinomial logistic regression**).
- **Multinomial logistic regression** uses the **softmax function** to compute probabilities.