

# Logistic Regression and Perceptron

Instructor: Wei Xu

Some slides adapted from Dan Jurfasky, Brendan O'Connor and Marine Carpuat

# NB & LR

- Both are linear models

$$z = \sum_{i=0}^{|X|} w_i x_i$$

- Training is different:
  - NB: weights are trained independently
  - LR: weights trained jointly

# Linear Models

- Compute Features:

$$f(d_i) = x_i = \begin{pmatrix} \text{count}(\text{"nigerian"}) \\ \text{count}(\text{"prince"}) \\ \text{count}(\text{"nigerian prince"}) \end{pmatrix}$$

- Assume we are given some weights:

$$w = \begin{pmatrix} -1.0 \\ -1.0 \\ 4.0 \end{pmatrix}$$

# Linear Models

- Compute Features
- We are given some weights
- Compute the dot product:

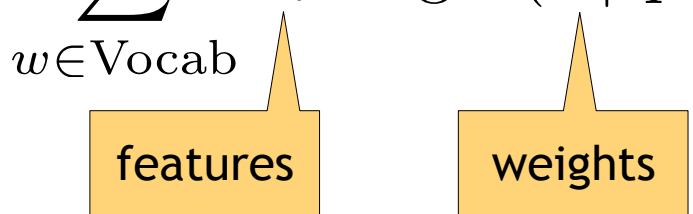
$$z = \sum_{i=0}^{|X|} w_i x_i$$

- Intuition: weighted sum of features
- All Linear models have this form

# Naïve Bayes as a Log-Linear Model

$$P(\text{spam}|D) \propto P(\text{spam}) \prod_{w \in D} P(w|\text{spam})$$

$$P(\text{spam}|D) \propto P(\text{spam}) \prod_{w \in \text{Vocab}} P(w|\text{spam})^{x_i}$$

$$\log P(\text{spam}|D) \propto \log P(\text{spam}) + \sum_{w \in \text{Vocab}} x_i \cdot \log P(w|\text{spam})$$


The diagram illustrates the log-linear model equation. The term  $x_i$  in the summation is pointed to by a yellow callout box labeled "features". The term  $\log P(w|\text{spam})$  is pointed to by a yellow callout box labeled "weights".

# Logistic Regression

- (Log) Linear Model - similar to Naïve Bayes
- Doesn't assume features are independent
- Correlated features don't “double count”

# Logistic Regression

- Compute the dot product:

$$z = \sum_{i=0}^{|X|} w_i x_i$$

linear combination

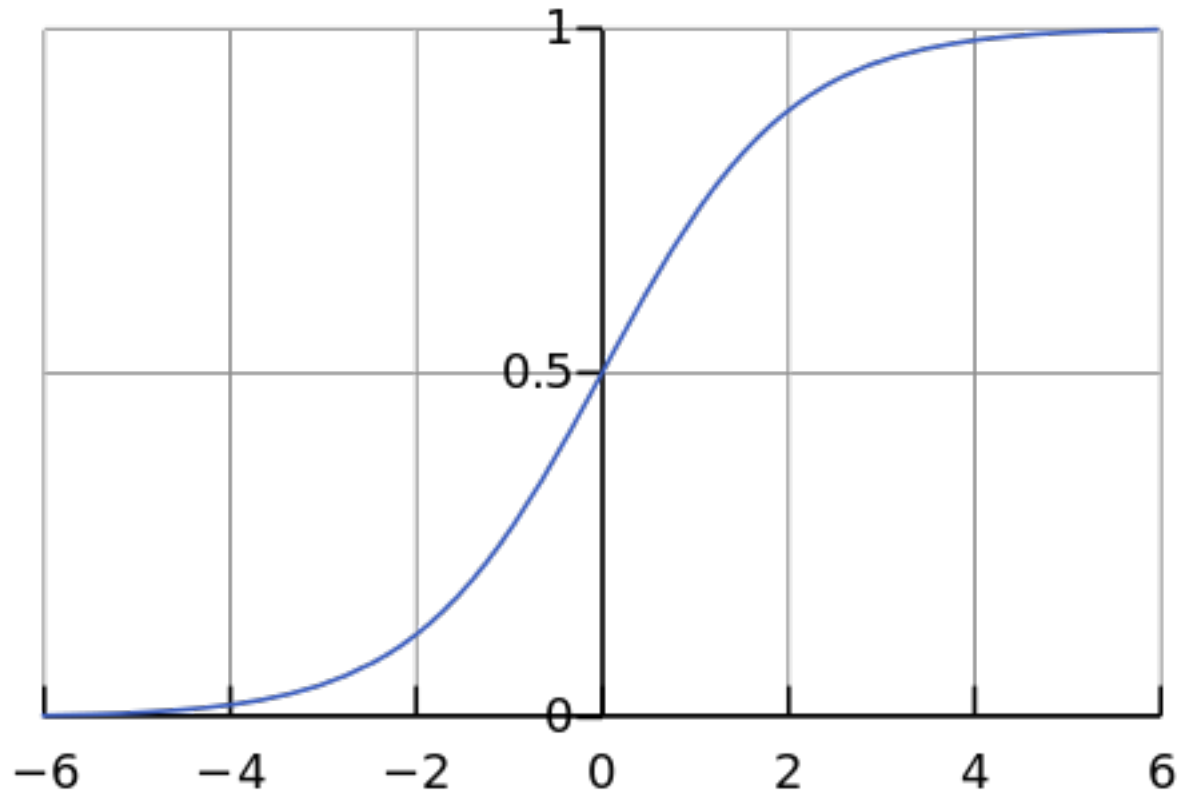
- Compute the logistic function:

$$P(\text{spam}|x) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$

convert into  
probabilities  
between [0, 1]

exponential/log space

# The Logistic function



$$P(\text{spam}|x) = \frac{e^z}{e^z + 1} = \frac{1}{1 + e^{-z}}$$



# NB vs. LR

- Both compute the dot product
- NB: sum of log probabilities
- LR: logistic function

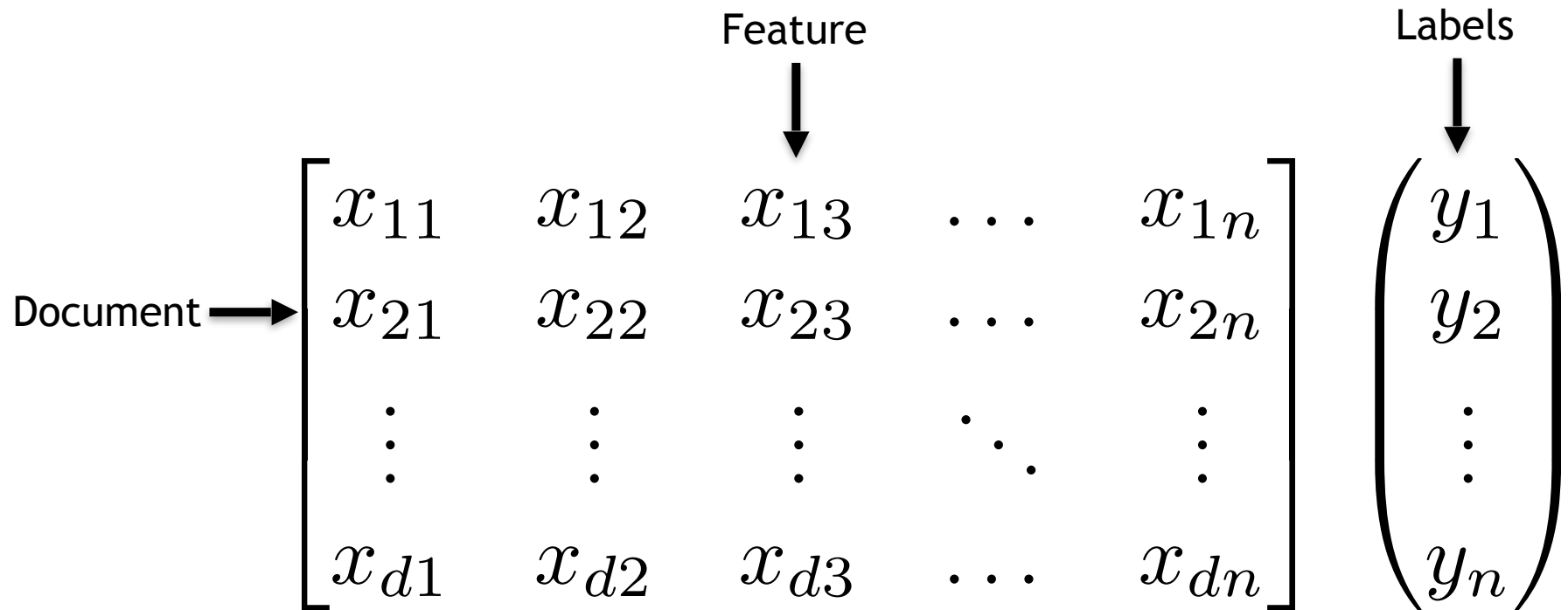
# NB vs. LR: Parameter Learning

- NB: Learn conditional probabilities **independently** by counting
- LR: Learn feature weights **jointly**

# LR: Learning Weights

- Given: a set of feature vectors and labels
- Goal: learn the weights

# LR: Learning Weights



Q: what parameters should we choose?

- What is the right value for the weights?
- Maximum Likelihood Principle:
  - Pick the parameters that maximize the probability of the  $y$  labels in the training data given the observations  $x$ .

# Maximum Likelihood Estimation

$$w_{\text{MLE}} = \operatorname{argmax}_w \log P(y_1, \dots, y_d | x_1, \dots, x_d; w)$$

$$= \operatorname{argmax}_w \sum_i \log P(y_i | x_i; w)$$

$$= \operatorname{argmax}_w \sum_i \log \begin{cases} p_i, & \text{if } y_i = 1 \\ 1 - p_i, & \text{if } y_i = 0 \end{cases}$$

logistic function

$$p_i = \sigma(\sum_j w_j x_j)$$

$$= \operatorname{argmax}_w \sum_i \log p_i^{\mathbb{I}(y_i=1)} (1 - p_i)^{\mathbb{I}(y_i=0)}$$

# Maximum Likelihood Estimation

$$= \operatorname{argmax}_w \sum_i \log p_i^{\mathbb{I}(y_i=1)} (1 - p_i)^{\mathbb{I}(y_i=0)}$$

$$= \operatorname{argmax}_w \sum_i y_i \log p_i + (1 - y_i) \log(1 - p_i)$$

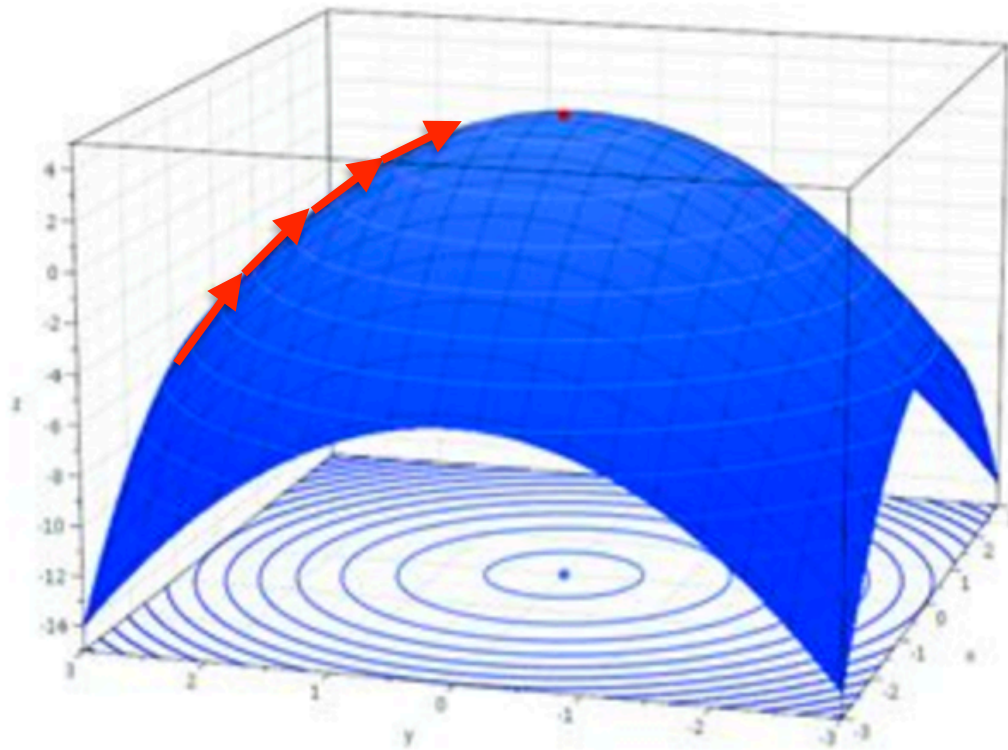
- Unfortunately there is no closed form solution
  - (like there was with naïve Bayes)

# Maximum Likelihood Estimation

- Solution:
  - Iteratively climb the log-likelihood surface through the derivatives for each weight
- Luckily, the derivatives turn out to be nice



# Gradient Ascent



# Gradient Ascent

Loop While not converged:

For all features  $j$ , compute and add derivatives

$$w_j^{\text{new}} = w_j^{\text{old}} + \eta \frac{\partial}{\partial w_j} \mathcal{L}(w)$$

$\mathcal{L}(w)$ : Training set log-likelihood

$\left( \frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \dots, \frac{\partial \mathcal{L}}{\partial w_n} \right)$  : Gradient vector

# LR Gradient

$$w_{\text{MLE}} = \underset{w}{\operatorname{argmax}} \underbrace{\sum_i y_i \log p_i + (1 - y_i) \log(1 - p_i)}_{\mathcal{L}}$$

$$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_i (y_i - p_i) x_j$$

# Exercise

# Derivative of Sigmoid

$$\begin{aligned}\frac{d}{dx}\sigma(x) &= \frac{d}{dx} \left[ \frac{1}{1 + e^{-x}} \right] \\&= \frac{d}{dx} (1 + e^{-x})^{-1} \\&= -(1 + e^{-x})^{-2}(-e^{-x}) \\&= \frac{e^{-x}}{(1 + e^{-x})^2} \\&= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} \\&= \frac{1}{1 + e^{-x}} \cdot \frac{(1 + e^{-x}) - 1}{1 + e^{-x}} \\&= \frac{1}{1 + e^{-x}} \cdot \left( 1 - \frac{1}{1 + e^{-x}} \right) \\&= \sigma(x) \cdot (1 - \sigma(x))\end{aligned}$$

# Logistic Regression: Pros and Cons

- Doesn't assume conditional independence of features
  - Better calibrated probabilities
  - Can handle highly correlated overlapping features
- NB is faster to train, less likely to overfit

# MultiClass Classification

- Q: what if we have more than 2 categories?
  - Sentiment: Positive, Negative, Neutral
  - Document topics: Sports, Politics, Business, Entertainment, ...

Q: How to easily do Multi-label classification?

# Two Types of MultiClass Classification

- Multi-label Classification
  - each instance can be assigned more than one labels
- Multinomial Classification
  - each instance appears in exactly one class (classes are exclusive)



# Multinomial Classification

- Pretty straightforward with Naive Bayes.

$$P(\text{spam}|D) \propto P(\text{spam}) \prod_{w \in D} P(w|\text{spam})$$

# Log-Linear Models

$$P(y|x) \propto e^{w \cdot f(d,y)}$$

$$P(y|x) = \frac{1}{Z(w)} e^{w \cdot f(d,y)}$$

# Multinomial Logistic Regression

$$P(y|x) \propto e^{w \cdot f(x,y)}$$

$$P(y|x) = \frac{1}{Z(w)} e^{w \cdot f(x,y)}$$

$$P(y|x) = \frac{e^{w \cdot f(x,y)}}{\sum_{y' \in Y} e^{w \cdot f(x,y')}}$$

# Multinomial Logistic Regression

- Binary (two classes):
  - We have one feature vector that matches the size of the vocabulary
- Multi-class in practice:
  - one weight vector for each category

$w_{\text{pos}}$

$w_{\text{neg}}$

$w_{\text{neut}}$

Can represent this in practice with one giant weight vector and repeated features for each category.

# Maximum Likelihood Estimation

$$w_{\text{MLE}} = \operatorname{argmax}_w \log P(y_1, \dots, y_n | x_1, \dots, x_n; w)$$

$$= \operatorname{argmax}_w \sum_i \log P(y_i | x_i; w)$$

$$= \operatorname{argmax}_w \sum_i \log \frac{e^{w \cdot f(x_i, y_i)}}{\sum_{y' \in Y} e^{w \cdot f(x_i, y')}}$$

# (a.k.a) Softmax Regression



WIKIPEDIA  
The Free Encyclopedia

[Main page](#)  
[Contents](#)  
[Featured content](#)  
[Current events](#)  
[Random article](#)  
[Donate to Wikipedia](#)  
[Wikipedia store](#)

Not logged in [Talk](#) [Contributions](#) [Create account](#) [Log in](#)

Article

[Talk](#)

Read

[Edit](#)

[View history](#)



## Softmax function

From Wikipedia, the free encyclopedia

In [mathematics](#), the **softmax function**, or **normalized exponential function**,<sup>[1]:198</sup> is a generalization of the [logistic function](#) that "squashes" a  $K$ -dimensional vector  $\mathbf{z}$  of arbitrary real values to a  $K$ -dimensional vector  $\sigma(\mathbf{z})$  of real values in the range (0, 1) that add up to 1. The function is given by

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

# (a.k.a) Maximum Entropy Classifier

- or MaxEnt
- Math proof of “LR=MaxEnt”:
  - [Klein and Manning 2003]
  - [Mount 2011]

<http://www.win-vector.com/dfiles/LogisticRegressionMaxEnt.pdf>

# Multiclass LR Gradient

$$\frac{\partial \mathcal{L}}{\partial w_j} = \sum_{i=1}^D f_j(y_i, d_i) - \sum_{i=1}^D \sum_{y \in Y} f_j(y, d_i) P(y|d_i)$$



# Perceptron Algorithm

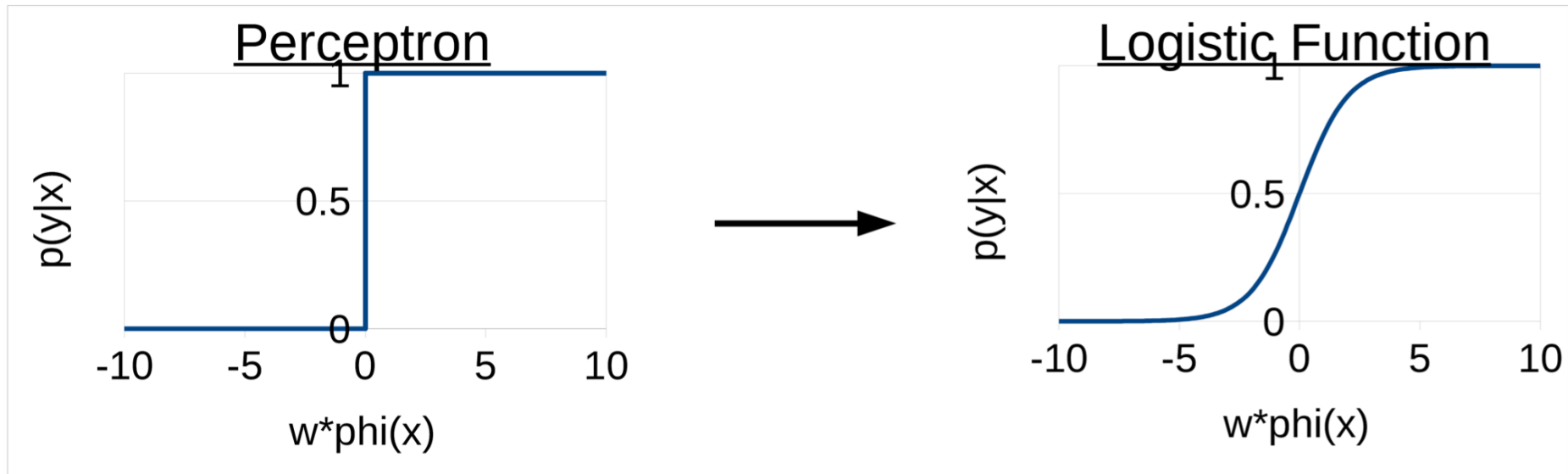
- Very similar to logistic regression
- Not exactly computing gradient



[Rosenblatt 1957]

# Perceptron Algorithm

- Very similar to logistic regression
- Not exactly computing gradient



$$\begin{aligned} P(y=1|x) &= 1 \text{ if } \mathbf{w} \cdot \boldsymbol{\phi}(x) \geq 0 \\ P(y=1|x) &= 0 \text{ if } \mathbf{w} \cdot \boldsymbol{\phi}(x) < 0 \end{aligned}$$

$$P(y=1|x) = \frac{e^{\mathbf{w} \cdot \boldsymbol{\phi}(x)}}{1 + e^{\mathbf{w} \cdot \boldsymbol{\phi}(x)}}$$

# Online Learning

- Update parameters for each training example (when predication is wrong)

```
for / iterations
  for each labeled pair  $x$ ,  $y$  in the data
     $\phi = \text{CREATE\_FEATURES}(x)$ 
     $y' = \text{PREDICT\_ONE}(w, \phi)$ 
    if  $y' \neq y$ 
       $\text{UPDATE\_WEIGHTS}(w, \phi, y)$ 
```

# Online Learning

- The Perceptron is an online learning algorithm.
- Logistic Regression is not:

$$w_{\text{MLE}} = \operatorname{argmax}_w \log P(y_1, \dots, y_d | x_1, \dots, x_d; w)$$

# Perceptron Algorithm

- Very similar to logistic regression
- Not exactly computing gradient

Initialize weight vector  $w = 0$

Loop for  $K$  iterations

  Loop For all training examples  $x_i$

    if  $\text{sign}(w * x_i) \neq y_i$

$w += (y_i - \text{sign}(w * x_i)) * x_i$

# Perceptron Notes

- Guaranteed to converge if the data is linearly separable
- Only hyperparameter is maximum number of iterations
- Parameter averaging will greatly improve performance

# Differences between LR and Perceptron

- Online learning vs. Batch
- Perceptron doesn't always make updates