

Logistic Regression for Text Classification

CS 485, Fall 2026

Applications of Natural Language Processing

Brendan O'Connor

College of Information and Computer Sciences

University of Massachusetts Amherst

[With slides from Ari Kobren and SLP3]

- First midterm: either 10/19 or 10/21

Linear classification models

- The foundational model for machine learning-based NLP!
- Examples
 - The humble "keyword count" classifier (no ML)
 - Naive Bayes ("generative" ML)
- Today: **Logistic Regression**
 - a linear classification model, trained to be good at *prediction*
 - allows for *features* (today: Bag-of-Words and manually defined features)
 - later: used within more complex models (neural networks), with machine-learned features (word embeddings)

Motivation: feature engineering

- For Naive Bayes, we used counts of each word in the vocabulary (BOW representation). But why not also use....
 - Number of words from "CS485 Crowdsourcing Positive Lexicon"
 - ...from "CS485 Crowdsourcing Negative Lexicon" ... or another...
 - Phrases (e.g. bigrams):
 - "The movie is awesome" => {"the movie", "movie is", "is awesome"}
 - Words/phrases with negation markers?
 - Number of "!" occurrences?
 - or...?
- NB tends to work poorly when there are many potentially



Var	Definition	Value in Fig. 5.2
x_1	count(positive lexicon words $\in$ doc)	3
x_2	count(negative lexicon words $\in$ doc)	2
x_3	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	1
x_4	count(1st and 2nd pronouns $\in$ doc)	3
x_5	$\begin{cases} 1 & \text{if "!"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	0
x_6	$\ln(\text{word count of doc})$	$\ln(66) = 4.19$

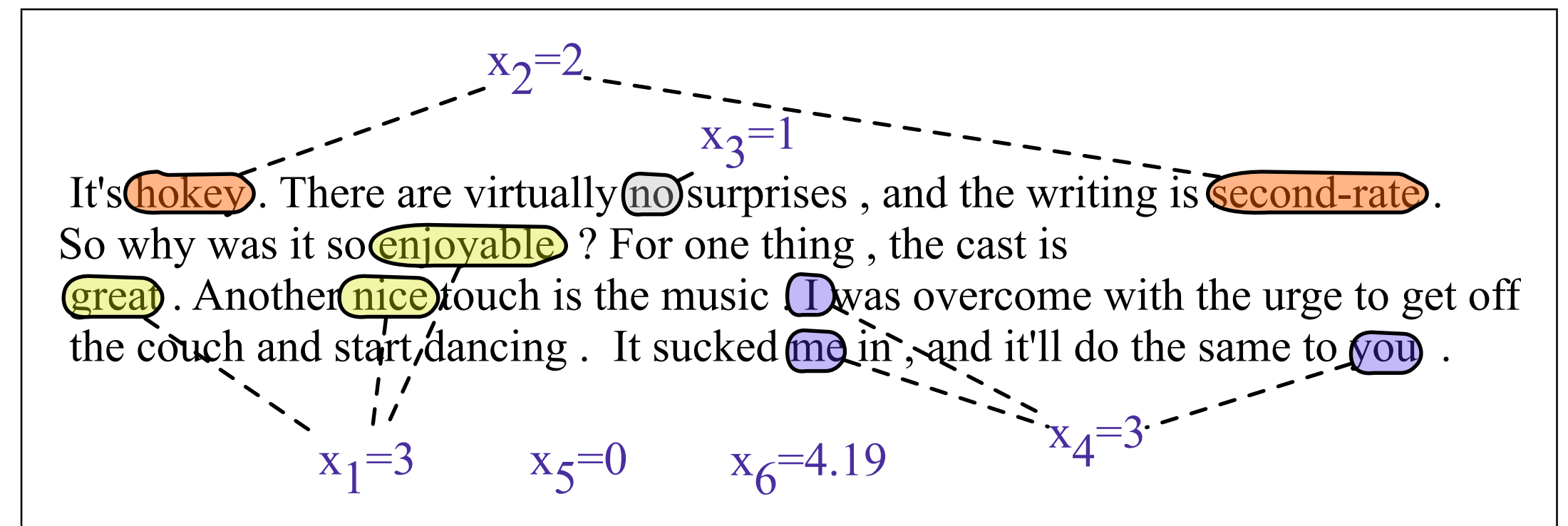


Figure 5.2 A sample mini test document showing the extracted features in the vector x .

x_7 : count of "aardvark"
 x_8 : count of "apple"
 ...

← Bag of words features (usually the most important ones, so far)

- Logistic regression can accommodate **any arbitrary features**
- Feature engineering: when you try and test new features. What improves accuracy? Can apply knowledge about the problem

Negation

Das, Sanjiv and Mike Chen. 2001. Yahoo! for Amazon: Extracting market sentiment from stock message boards. In Proceedings of the Asia Pacific Finance Association Annual Conference (APFA).
Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. 2002. Thumbs up? Sentiment Classification using Machine Learning Techniques. EMNLP-2002, 79—86.

Add NOT_ to every word between negation and following punctuation:

didn't like this movie , but I



didn't NOT_like NOT_this NOT_movie but I

Classification: LogReg (I)

First, we'll discuss **how LogReg works.**

Then, **why** it's set up the way that it is.

Application: **spam filtering**

Classification: LogReg (I)

- compute **features** (xs)

$$\mathbf{x}_i = (\text{count "nigerian", count "prince", count "nigerian prince"})$$

- given **weights** (betas)

$$\boldsymbol{\beta} = (-1.0, \quad -1.0, \quad 4.0)$$

Classification: LogReg (I)

- compute **features** (x's)
- given **weights** (betas)
- compute the **dot product**

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

Classification: LogReg (II)

- compute the **dot product**

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

- compute the **logistic function**

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

LogReg Exercise

features: (count “nigerian”, count “prince”, count “nigerian prince”)

$$\mathbf{x} = (1, 1, 1)$$

$$\boldsymbol{\beta} = (-1.0, -1.0, 4.0)$$

$$P(y=1 \mid \mathbf{x}) =$$

LogReg Exercise

$$\mathbf{x} = (1, 1, 1)$$

$$\boldsymbol{\beta} = (-1.0, -1.0, 4.0)$$

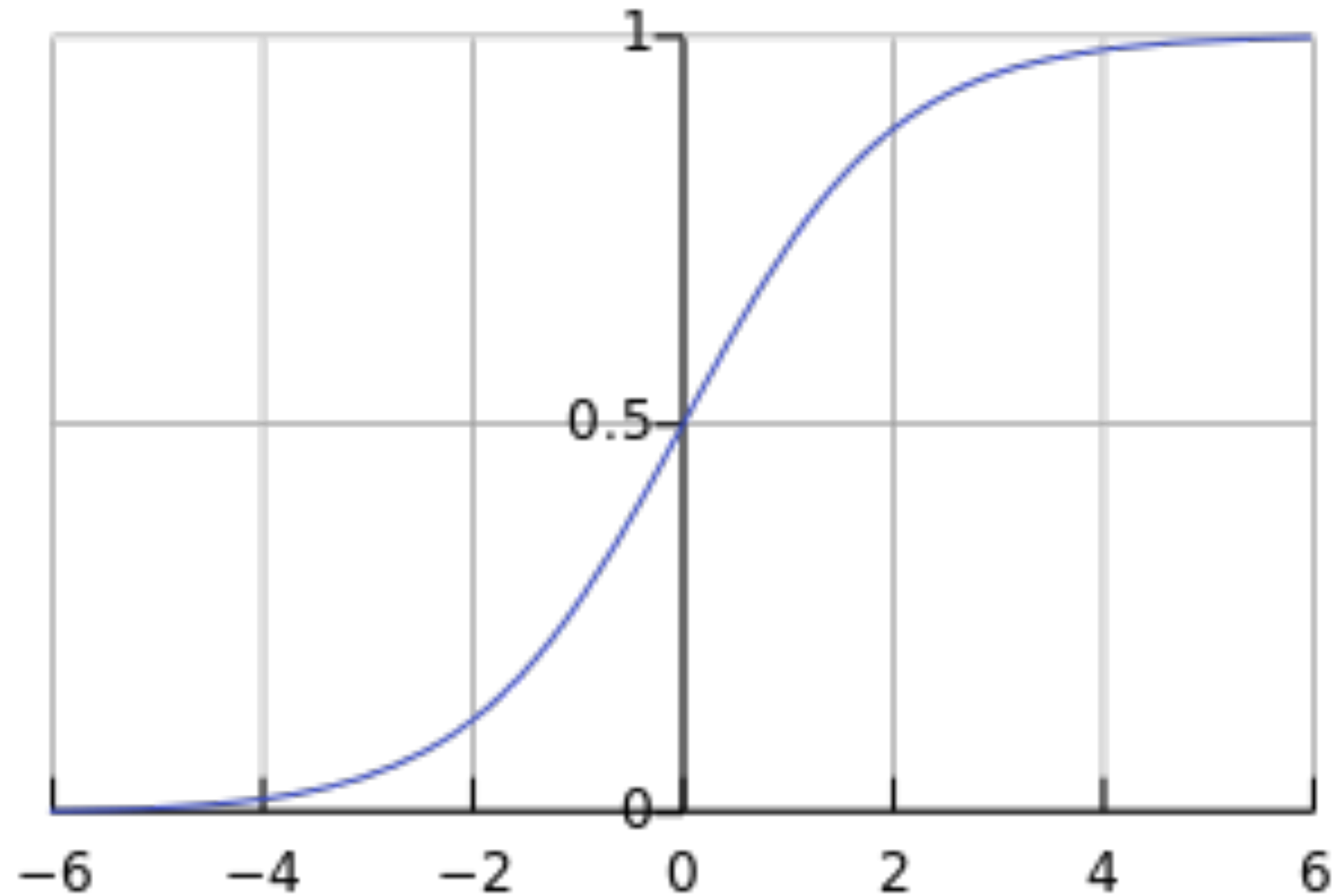
$$z = \sum_{i=0}^{|\mathbf{X}|} \beta_i x_i$$

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

Why the **logistic function**?

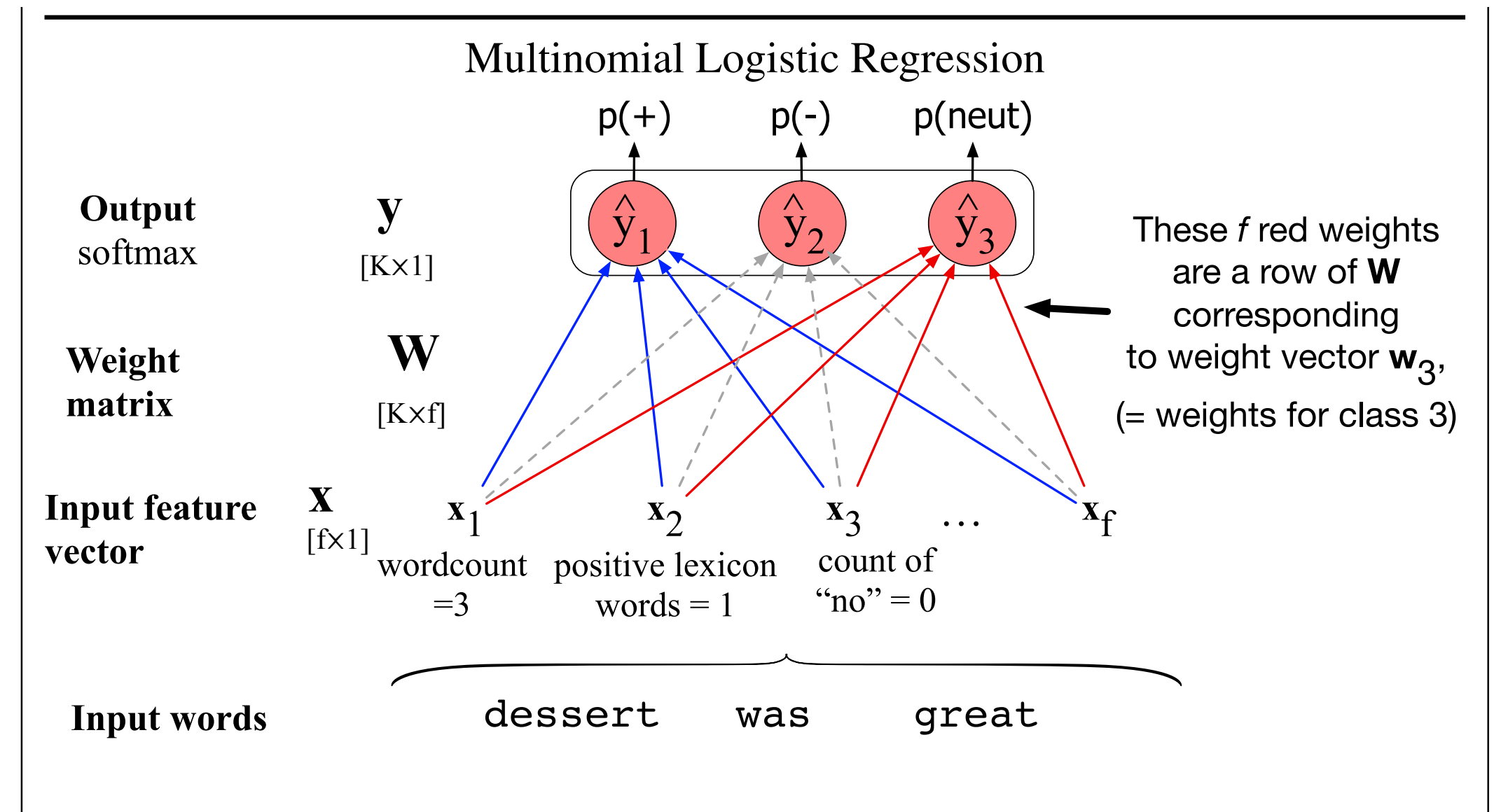
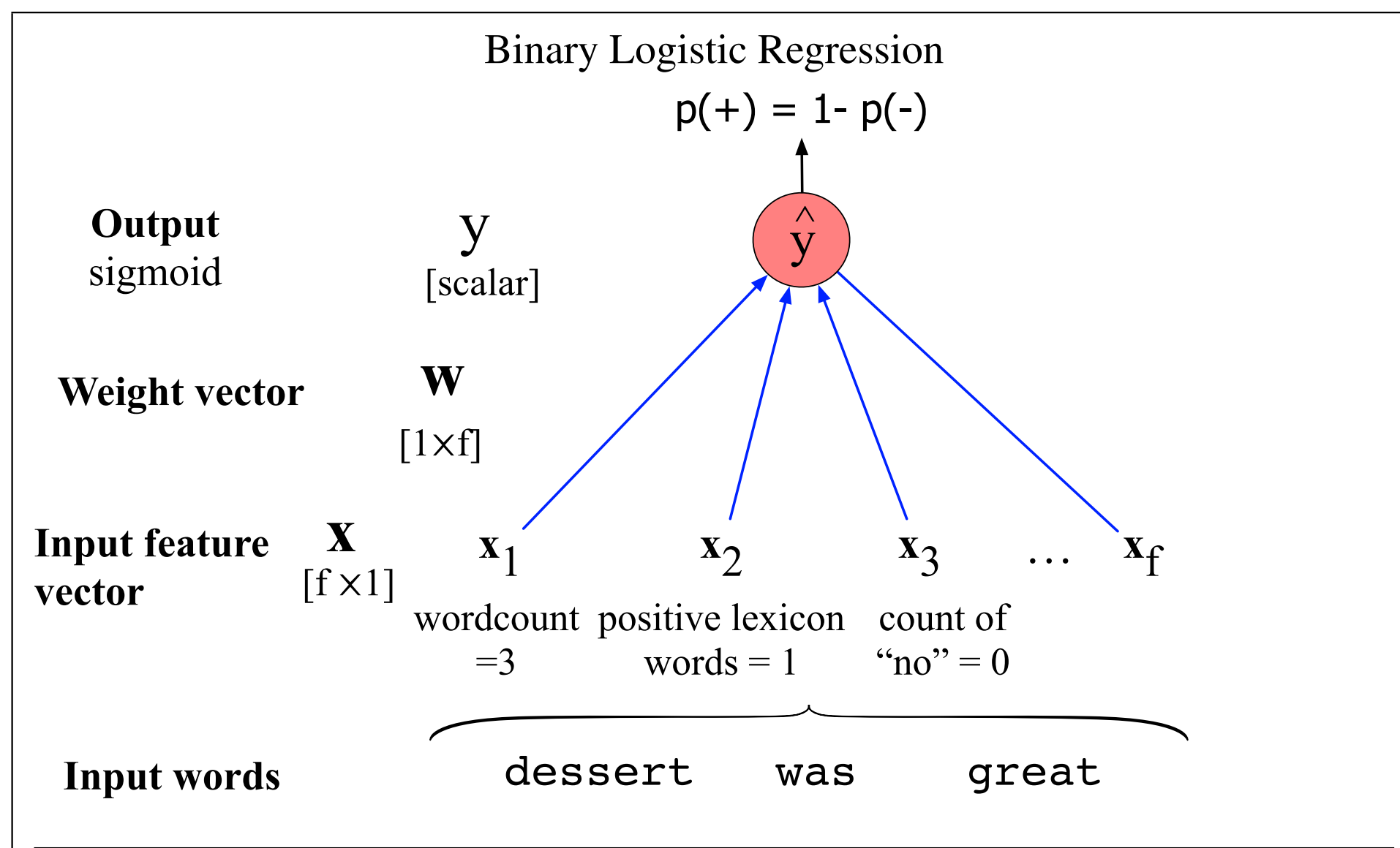
Logistic Function

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



Multiclass Logistic Regression

- Generalize to $K > 2$ classes
- Each class has its own weight vector (across all features; e.g. BOW counts)



Multiclass Logistic Regression

- Weight vector for each class
- Prediction: dot product for each class
- Predicted probabilities: apply the *softmax function* to normalize

Why the softmax function?

Learning Weights

- **NB**: learn conditional probabilities separately via **counting**
- **LogReg**: learn weights **jointly**

Learning Weights

- given: a set of **feature vectors** and **labels**
- goal: learn the weights.

Learning Weights

x_{00}	x_{01}	$\dots$	x_{0m}	y_0
x_{10}	x_{11}	$\dots$	x_{1m}	y_1
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
x_{n0}	x_{n1}	$\dots$	x_{nm}	y_n

n examples; xs - features; ys - class

Learning Weights

We know:

$$g(z) = \frac{1}{1 + e^{-z}} \quad P(y = 1 \mid x) = g \left(\sum_{j=1}^{\text{Nfeat}} \beta_j x_{ij} \right)$$

So let's try to maximize probability of the entire dataset - **maximum likelihood estimation**

Learning Weights

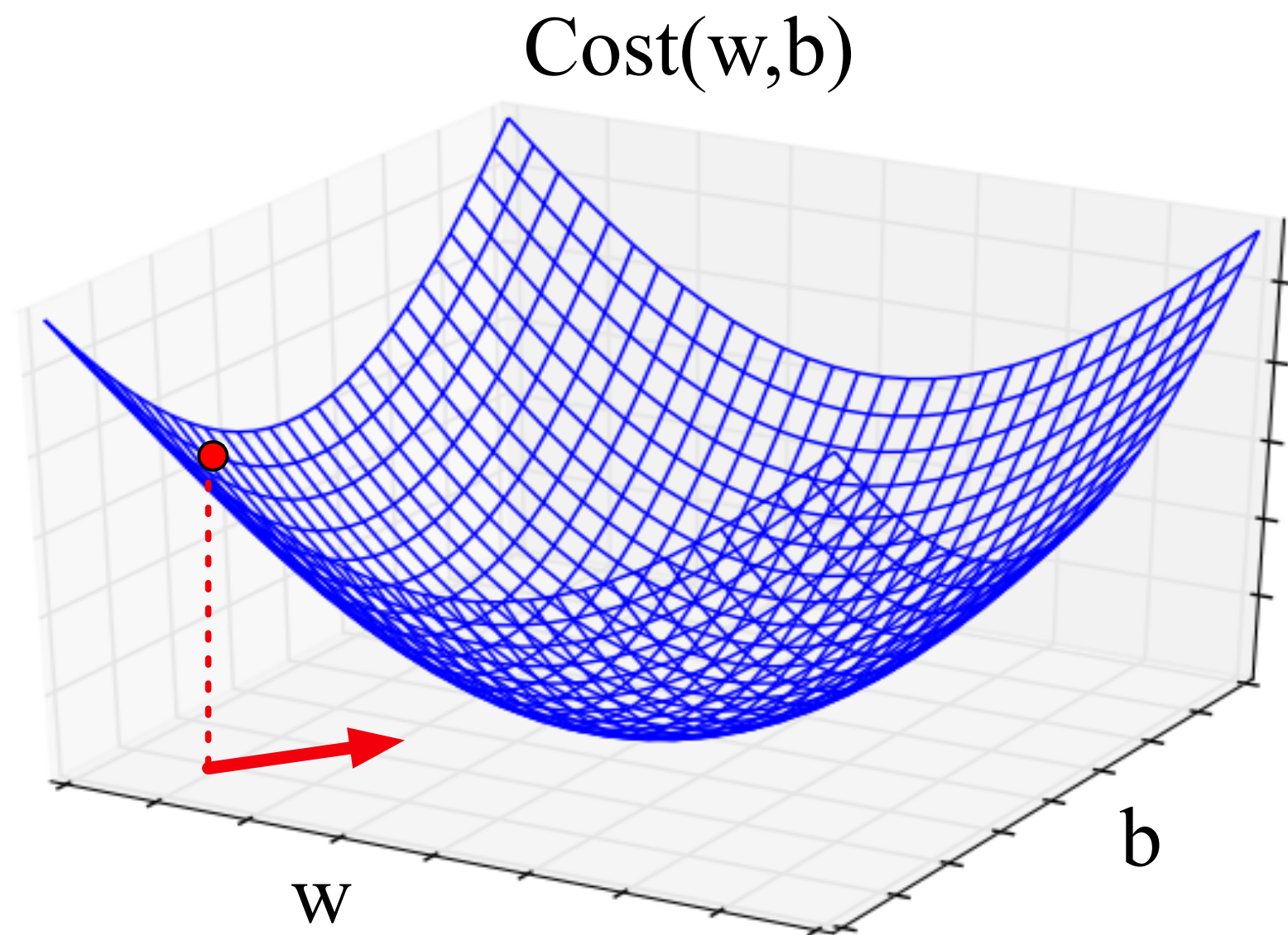
So let's try to maximize probability of the entire dataset - **maximum likelihood estimation**

$$\beta^{MLE} = \arg \max_{\beta} \log P(y_0, \dots, y_n | \mathbf{x}_0, \dots, \mathbf{x}_n; \beta)$$

Gradient ascent/descent learning

$$\beta^{MLE} = \arg \max_{\beta} \log P(y_0, \dots, y_n | \mathbf{x}_0, \dots, \mathbf{x}_n; \beta)$$

- Follow direction of *steepest ascent*. Iterate: $\beta^{(new)} = \beta^{(old)} + \eta \frac{\partial \ell}{\partial \beta}$



$\left(\frac{\partial \ell}{\partial \beta_1}, \dots, \frac{\partial \ell}{\partial \beta_J} \right)$: Gradient vector
(vector of per-element derivatives)

GD is a generic method for optimizing differentiable functions — widely used in machine learning!

Logistic regression wrap-up

- Given you can extract features from your text, logistic regression is the best, easy-to-use, method
- Logistic regression with BOW features is an excellent baseline method to try at first
- Will be a foundation for more sophisticated models, later in course
- Always regularize your LR model
- We recommend using the implementation in scikit-learn
 - Useful: CountVectorizer to help make BOW count vectors
- Next: but where do the LABELS in supervised learning come from?