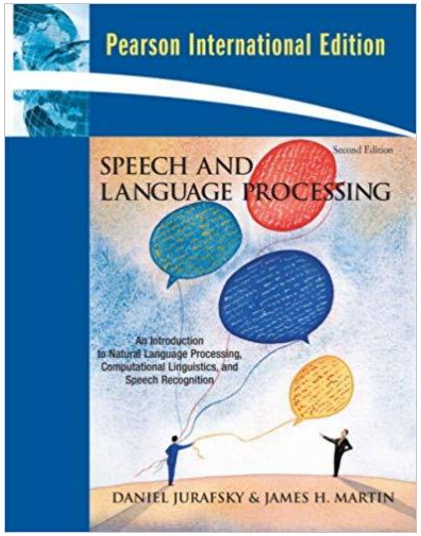
NLP & IR



Chapter 5

Logistic Regression

Dell Zhang
Birkbeck, University of London

Generative vs Discriminative

- A **generative** model like Naïve Bayes makes use of the *likelihood* term $P(d|c)$, which expresses how to generate the features of a document d if we knew it was of class c.
- A **discriminative** model like Logistic Regression in this text categorization scenario attempts to directly compute $P(c|d)$.

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

Components

1. A **feature representation** of the input. For each input observation $x^{(i)}$, this will be a vector of features $[x_1, x_2, ..., x_n]$. We will generally refer to feature i for input $x^{(j)}$ as $x_i^{(j)}$, sometimes simplified as x_i, but we will also see the notation f_i, $f_i(x)$, or, for multiclass classification, $f_i(c,x)$.
2. A classification function that computes $\hat{y}$, the estimated class, via $p(y|x)$. In the next section we will introduce the **sigmoid** and **softmax** tools for classification.
3. An objective function for learning, usually involving minimizing error on training examples. We will introduce the **cross-entropy loss function**
4. An algorithm for optimizing the objective function. We introduce the **stochastic gradient descent** algorithm.

Stages

training: we train the system (specifically the weights w and b) using stochastic gradient descent and the cross-entropy loss.

test: Given a test example x we compute $p(y|x)$ and return the higher probability label $y = 1$ or $y = 0$.

Classification Function

- The weights and bias

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

$$= w \cdot x + b$$

- The sigmoid
(a special case of logistic function)

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

Classification Function

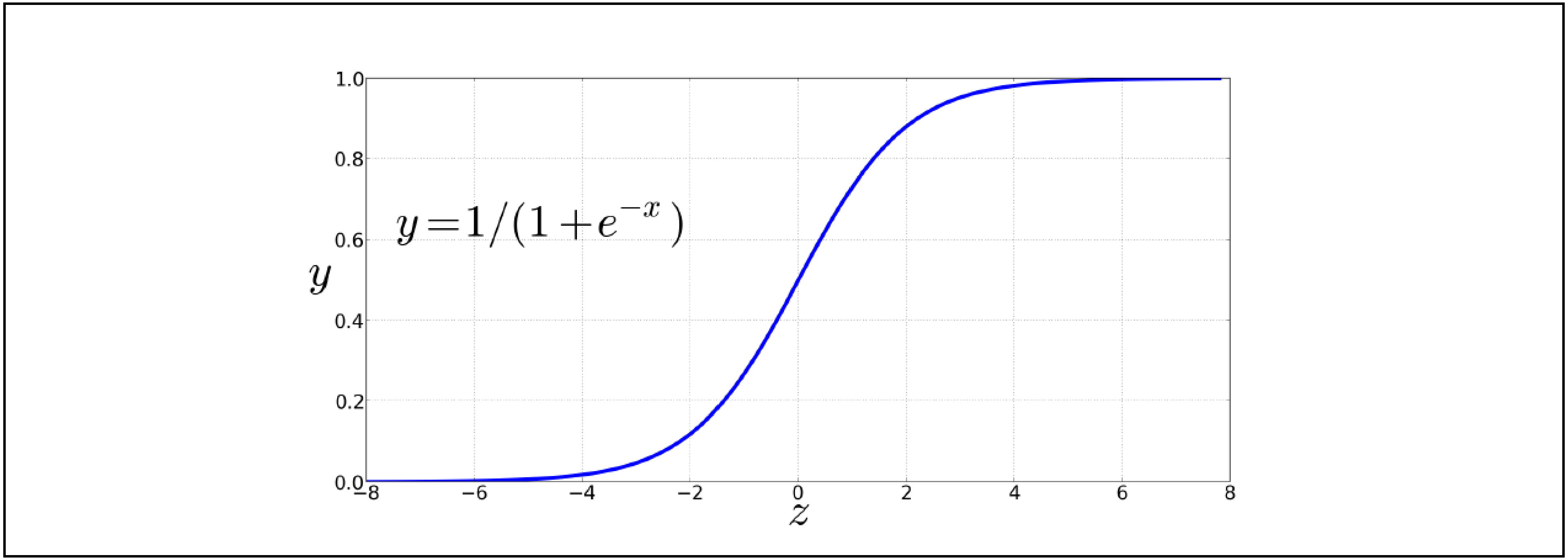


Figure 5.1 The sigmoid function $y = \frac{1}{1+e^{-z}}$ takes a real value and maps it to the range $[0, 1]$. Because it is nearly linear around 0 but has a sharp slope toward the ends, it tends to squash outlier values toward 0 or 1.

Classification Function

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

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

Example

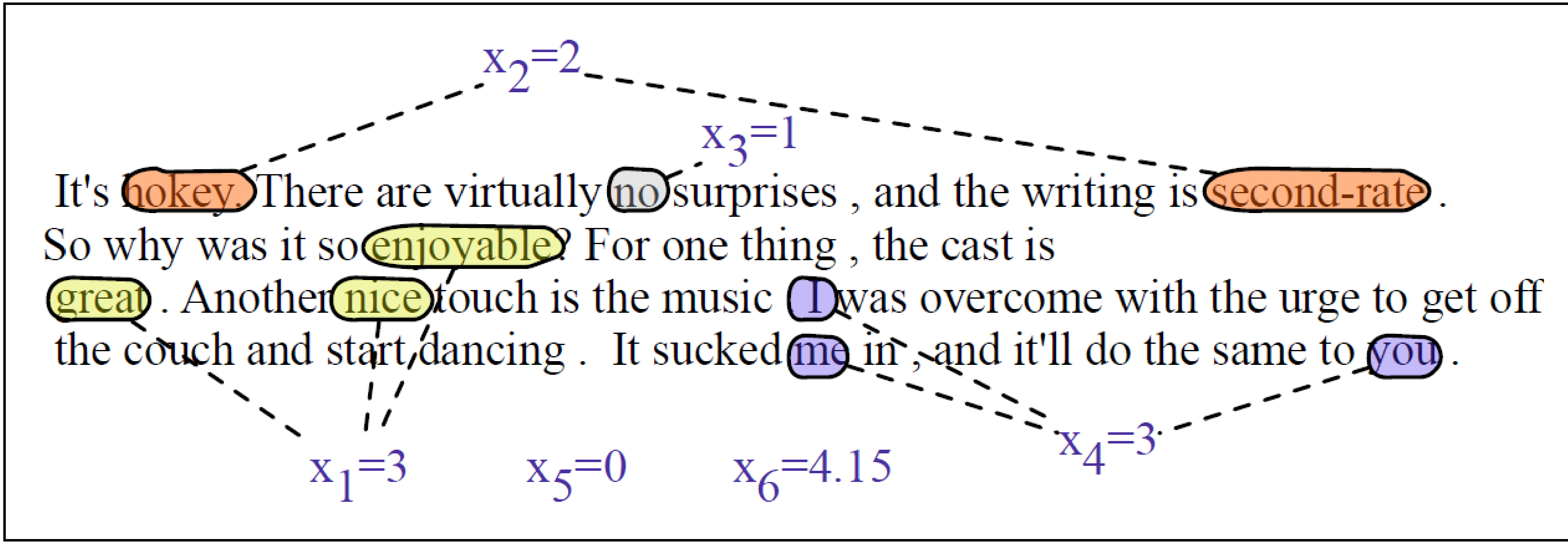


Figure 5.2 A sample mini test document showing the extracted features in the vector *x*.

Example

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	$\begin{cases} 1 & \text{if "no"} \in \text{doc} \\ 0 & \text{otherwise} \end{cases}$	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	log(word count of doc)	$\ln(64) = 4.15$

Example

Let's assume for the moment that we've already learned a real-valued weight for each of these features, and that the 6 weights corresponding to the 6 features are $[2.5, -5.0, -1.2, 0.5, 2.0, 0.7]$, while $b = 0.1$.

$$
\begin{aligned}
p(+|x) = P(Y=1|x) &= \sigma(w \cdot x + b) \\
&= \sigma([2.5, -5.0, -1.2, 0.5, 2.0, 0.7] \cdot [3, 2, 1, 3, 0, 4.15] + 0.1) \\
&= \sigma(0.805) \\
&= 0.69 \\
p(-|x) = P(Y=0|x) &= 1 - \sigma(w \cdot x + b) \\
&= 0.31
\end{aligned}
$$

Features

- Where do they come from?
  - Feature Engineering
  - Representation Learning (using Deep Learning methods etc.)

LR vs NB

- Naïve Bayes has overly strong conditional independence assumptions. By contrast, logistic regression is much more robust to correlated features.
- Thus when there are many correlated features, logistic regression will assign a more accurate probability than Naïve Bayes.
- So logistic regression generally works better on *large datasets or long documents*, and is a common default.

LR vs NB

- Despite the less accurate probabilities, Naïve Bayes still often makes the correct classification decision.
- Naïve Bayes works extremely well (even better than Logistic Regression) on *small datasets or short documents*.
- Furthermore, it is easy to implement and very fast to train (there’s no optimization step).
- So it’s still a reasonable approach to use in some situations.

Learning

- Objective:
  to minimize the **cross-entropy** loss function

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

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

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

$$Cost(w,b) = \frac{1}{m}\sum_{i=1}^{m} L_{CE}(\hat{y}^{(i)}, y^{(i)})$$

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

Learning

- Algorithm: Stochastic Gradient Descent
- Regularization

Multinomial Logistic Regression

- Also called the **softmax** regression (or, historically, the maxent classifier)

The softmax of an input vector $z = [z_1, z_2, \ldots, z_k]$ is:

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

The denominator $\sum_{i=1}^{k} e^{z_i}$ is used to normalize all the values into probabilities.

for each of the K classes:

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

Multinomial Logistic Regression

- The softmax function

Thus for example given a vector:

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

the result softmax(z) is

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

```
>>> import numpy as np
>>> z = [1.0, 2.0, 3.0, 4.0, 1.0, 2.0, 3.0]
>>> softmax = lambda z:np.exp(z)/np.sum(np.exp(z))
>>> softmax(z)
array([0.02364054, 0.06426166, 0.1746813 , 0.474833  , 0.02364054,
       0.06426166, 0.1746813 ])
```

Multinomial Logistic Regression

- The **cross-entropy** loss function (for K classes)
  - For a *hard classification* task (where only one class is the correct one for each document), this is just the **negative log-likelihood**.
  - Given a training document $\boldsymbol{x}$ in class k, i.e., $\boldsymbol{y} = [0, \ldots, 1, \ldots 0]$ where $y_k=1$ and $y_i=0$ for $i \neq k$.

$$L_{CE}(\widehat{\mathbf{y}}, \mathbf{y}) = \sum_{i=1}^{K} -y_i \log(\widehat{y}_i) = -\log(\widehat{y}_k) = -\log\left(\mathrm{softmax}(\mathbf{z})_k\right)$$

$$= -\log\left(\frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}\right) = -\log\left(\frac{e^{\mathbf{w}_k \mathbf{x} + b_k}}{\sum_{j=1}^{K} e^{\mathbf{w}_j \mathbf{x} + b_j}}\right)$$