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 $P(c|d)$.

$$\hat{c} = \arg\max_{c \in C} \left( \underbrace{P(d|c)}_{\text{likelihood}} \underbrace{P(c)}_{\text{prior}} \right)$$
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.
**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

\[
\begin{align*}
    z &= \left( \sum_{i=1}^{n} w_i x_i \right) + b \\
    &= w \cdot x + b
\end{align*}
\]

- The sigmoid (a special case of logistic function)

\[
y = \sigma(z) = \frac{1}{1 + e^{-z}}
\]
The sigmoid function $y = \frac{1}{1 + e^{-x}}$ 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.
\[ \hat{y} = P(y = 1|x) = \sigma(w \cdot x + b) \]
\[ = \frac{1}{1 + e^{-(w \cdot x + b)}} \]

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

<table>
<thead>
<tr>
<th>Var</th>
<th>Definition</th>
<th>Value in Fig. 5.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1$</td>
<td>count(positive lexicon) $\in$ doc</td>
<td>3</td>
</tr>
<tr>
<td>$x_2$</td>
<td>count(negative lexicon) $\in$ doc</td>
<td>2</td>
</tr>
<tr>
<td>$x_3$</td>
<td>$\begin{cases} 1 &amp; \text{if &quot;no&quot; } \in \text{doc} \ 0 &amp; \text{otherwise} \end{cases}$</td>
<td>1</td>
</tr>
<tr>
<td>$x_4$</td>
<td>count(1st and 2nd pronouns $\in$ doc)</td>
<td>3</td>
</tr>
<tr>
<td>$x_5$</td>
<td>$\begin{cases} 1 &amp; \text{if &quot;!&quot; } \in \text{doc} \ 0 &amp; \text{otherwise} \end{cases}$</td>
<td>0</td>
</tr>
<tr>
<td>$x_6$</td>
<td>log(word count of doc)</td>
<td>$\ln(64) = 4.15$</td>
</tr>
</tbody>
</table>
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 \(b = 0.1\).

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

\[
p(-|x) = P(Y = 0|x) = 1 - \sigma(w \cdot x + b) \\
= 0.31
\]
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.
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

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

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

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

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

\[
= - \frac{1}{m} \sum_{i=1}^{m} y^{(i)} \log \sigma(w \cdot x^{(i)} + b) + (1 - y^{(i)}) \log \left(1 - \sigma(w \cdot x^{(i)} + b)\right)
\]
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 \( \text{softmax}(z) \) is

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

```python
>>> 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**.

$$
L_{CE} (\hat{y}, y) = -\sum_{i=1}^{K} y_i \log \hat{y}_i
$$

$$
= -\sum_{i=1}^{K} \log \hat{y}_i
$$

$$
= -\sum_{i=1}^{K} \frac{e^{\hat{z}_i}}{\sum_{j=1}^{K} e^{\hat{z}_j}}
$$