1. Review of Logistic regression

2. Non-linear Decision Boundary

3. Multi-class Classification
   Multi-class Naive Bayes
   Multi-class Logistic Regression
Review of Logistic regression
Intuition: Logistic Regression

- \( x_1 = \# \) of times 'meet' appears in an email
- \( x_2 = \# \) of times 'lottery' appears in an email
- Define feature vector \( \mathbf{x} = [1, x_1, x_2] \)
- Learn the decision boundary \( w_0 + w_1 x_1 + w_2 x_2 = 0 \) such that
  - If \( \mathbf{w}^\top \mathbf{x} \geq 0 \) declare \( y = 1 \) (spam)
  - If \( \mathbf{w}^\top \mathbf{x} < 0 \) declare \( y = 0 \) (ham)

Key Idea: If 'meet' appears few times and 'lottery' appears many times than the email is spam
Intuition: Logistic Regression

- $x_1 =$ # of times 'lottery' appears in an email
- $x_2 =$ # of times 'meet' appears in an email
- Define feature vector $\mathbf{x} = [1, x_1, x_2]$
- Learn the decision boundary $w_0 + w_1 x_1 + w_2 x_2 = 0$ such that
  - If $\mathbf{w}^T \mathbf{x} \geq 0$ declare $y = 1$ (spam)
  - If $\mathbf{w}^T \mathbf{x} < 0$ declare $y = 0$ (ham)

$y = 1$ for spam, $y = 0$ for ham
Intuition: Logistic Regression

- Suppose we want to output the probability of an email being spam/ham instead of just 0 or 1
- This gives information about the confidence in the decision
- Use a function $\sigma(w^T x)$ that maps $w^T x$ to a value between 0 and 1
Intuition: Logistic Regression

- Suppose we want to output the probability of an email being spam/ham instead of just 0 or 1
- This gives information about the confidence in the decision
- Use a function $\sigma(w^T x)$ that maps $w^T x$ to a value between 0 and 1

Key Problem: Finding optimal weights $w$ that accurately predict this probability for a new email
Formal Setup: Binary Logistic Classification

- Input: $x = [1, x_1, x_2, \ldots, x_D] \in \mathbb{R}^{D+1}$
Formal Setup: Binary Logistic Classification

- Input: \( x = [1, x_1, x_2, \ldots x_D] \in \mathbb{R}^{D+1} \)
- Output: \( y \in \{0, 1\} \)
Formal Setup: Binary Logistic Classification

- Input: \( x = [1, x_1, x_2, \ldots x_D] \in \mathbb{R}^{D+1} \)
- Output: \( y \in \{0, 1\} \)
- Training data: \( D = \{(x_n, y_n), n = 1, 2, \ldots, N\} \)
Formal Setup: Binary Logistic Classification

- Input: \( x = [1, x_1, x_2, \ldots x_D] \in \mathbb{R}^{D+1} \)
- Output: \( y \in \{0, 1\} \)
- Training data: \( D = \{(x_n, y_n), n = 1, 2, \ldots, N\} \)
Formal Setup: Binary Logistic Classification

- Input: \( x = [1, x_1, x_2, \ldots x_D] \in \mathbb{R}^{D+1} \)
- Output: \( y \in \{0, 1\} \)
- Training data: \( \mathcal{D} = \{(x_n, y_n), n = 1, 2, \ldots, N\} \)
- Model:
  \[
p(y = 1|x; w) = \sigma[g(x)]
\]
Formal Setup: Binary Logistic Classification

- **Input:** \( \mathbf{x} = [1, x_1, x_2, \ldots, x_D] \in \mathbb{R}^{D+1} \)
- **Output:** \( y \in \{0, 1\} \)
- **Training data:** \( \mathcal{D} = \{(\mathbf{x}_n, y_n), n = 1, 2, \ldots, N\} \)
- **Model:**
  \[
  p(y = 1|\mathbf{x}; \mathbf{w}) = \sigma(g(\mathbf{x}))
  \]
Formal Setup: Binary Logistic Classification

- Input: \( x = [1, x_1, x_2, \ldots x_D] \in \mathbb{R}^{D+1} \)
- Output: \( y \in \{0, 1\} \)
- Training data: \( D = \{(x_n, y_n), n = 1, 2, \ldots , N\} \)
- Model:
  \[
  p(y = 1|x; w) = \sigma[g(x)]
  \]
  where
  \[
  g(x) = w_0 + \sum_d w_d x_d = w^\top x
  \]
Formal Setup: Binary Logistic Classification

- **Input:** \( x = [1, x_1, x_2, \ldots x_D] \in \mathbb{R}^{D+1} \)
- **Output:** \( y \in \{0, 1\} \)
- **Training data:** \( \mathcal{D} = \{(x_n, y_n), n = 1, 2, \ldots, N\} \)
- **Model:**
  \[
p(y = 1|x; w) = \sigma[g(x)]
\]
  where
  \[
g(x) = w_0 + \sum_d w_d x_d = w^\top x
\]
  and \( \sigma[\cdot] \) stands for the *sigmoid* function
  \[
  \sigma(a) = \frac{1}{1 + e^{-a}}
  \]
Why the sigmoid function?

What does it look like?

\[ \sigma(a) = \frac{1}{1 + e^{-a}} \]

Properties:
- Bounded between 0 and 1 ← thus, interpretable as probability
- Monotonically increasing ← thus, usable to derive classification rules
- \( \sigma(a) \geq 0.5 \), positive (classify as '1')
- \( \sigma(a) < 0.5 \), negative (classify as '0')
- Nice computational properties ← as we will see soon
Why the sigmoid function?

What does it look like?

\[ \sigma(a) = \frac{1}{1 + e^{-a}} \]

where

\[ a = \mathbf{w}^\top \mathbf{x} \]

Properties

• Bounded between 0 and 1 → thus, interpretable as probability
• Monotonically increasing → thus, usable to derive classification rules
• \( \sigma(a) \geq 0.5 \), positive (classify as '1')
• \( \sigma(a) < 0.5 \), negative (classify as '0')
• Nice computational properties → as we will see soon
Why the sigmoid function?

What does it look like?

\[ \sigma(a) = \frac{1}{1 + e^{-a}} \]

where

\[ a = w^\top x \]

Properties

- Bounded between 0 and 1 \( \leftarrow \) thus, interpretable as probability
Why the sigmoid function?

What does it look like?

\[
\sigma(a) = \frac{1}{1 + e^{-a}}
\]

where

\[a = \mathbf{w}^\top \mathbf{x}\]

Properties

- Bounded between 0 and 1 \(\leftarrow\) thus, interpretable as probability
- Monotonically increasing \(\leftarrow\) thus, usable to derive classification rules
Why the sigmoid function?

What does it look like?

\[ \sigma(a) = \frac{1}{1 + e^{-a}} \]

where

\[ a = w^\top x \]

Properties

- Bounded between 0 and 1 \( \leftarrow \) thus, interpretable as probability
- Monotonically increasing \( \leftarrow \) thus, usable to derive classification rules

- \( \sigma(a) \geq 0.5 \), positive (classify as '1')
Why the sigmoid function?

What does it look like?

\[
\sigma(a) = \frac{1}{1 + e^{-a}}
\]

where

\[
a = \mathbf{w}^\top \mathbf{x}
\]

Properties

- Bounded between 0 and 1 \(\leftarrow\) thus, interpretable as probability
- Monotonically increasing \(\leftarrow\) thus, usable to derive classification rules
  - \(\sigma(a) \geq 0.5\), positive (classify as '1')
  - \(\sigma(a) < 0.5\), negative (classify as '0')
Why the sigmoid function?

What does it look like?

\[ \sigma(a) = \frac{1}{1 + e^{-a}} \]

where

\[ a = w^\top x \]

Properties

- Bounded between 0 and 1 \( \leftarrow \) thus, interpretable as probability
- Monotonically increasing \( \leftarrow \) thus, usable to derive classification rules

\[ \begin{align*}
\sigma(a) &\geq 0.5, \text{ positive (classify as '1')} \\
\sigma(a) &< 0.5, \text{ negative (classify as '0')} 
\end{align*} \]

- Nice computational properties \( \leftarrow \) as we will see soon
How to optimize $w$? Consider the Data Likelihood

Probability of a single training sample $(x_n, y_n)$

$$p(y_n|x_n; w) = \begin{cases} 
\sigma(w^\top x_n) & \text{if } y_n = 1 \\
1 - \sigma(w^\top x_n) & \text{otherwise}
\end{cases}$$
How to optimize $w$? Consider the Data Likelihood

Probability of a single training sample $(x_n, y_n)$

$$p(y_n|x_n; w) = \begin{cases} 
\sigma(w^\top x_n) & \text{if } y_n = 1 \\
1 - \sigma(w^\top x_n) & \text{otherwise} 
\end{cases}$$

Compact expression, exploring that $y_n$ is either 1 or 0

$$p(y_n|x_n; w) = \sigma(w^\top x_n)^{y_n} [1 - \sigma(w^\top x_n)]^{1-y_n}$$

Probability that predicted label is 1 (spam)
Log Likelihood or Cross Entropy Error

Log-likelihood of the whole training data $\mathcal{D}$

$$\log P(\mathcal{D}) = \sum_n \{y_n \log \sigma(w^\top x_n) + (1 - y_n) \log [1 - \sigma(w^\top x_n)] \}$$
Log-Likelihood or Cross Entropy Error

Log-likelihood of the whole training data $\mathcal{D}$

$$\log P(\mathcal{D}) = \sum_n \{y_n \log \sigma(w^\top x_n) + (1 - y_n) \log [1 - \sigma(w^\top x_n)]\}$$

It is convenient to work with its negation, which is called cross-entropy error function

$$\mathcal{E}(b, w) = -\sum_n \{y_n \log \sigma(w^\top x_n) + (1 - y_n) \log [1 - \sigma(w^\top x_n)]\}$$
How to find the optimal parameters for logistic regression?

We will minimize the error function

$$E(w) = -\sum_{n} \{ y_n \log \sigma(w^\top x_n) + (1 - y_n) \log [1 - \sigma(w^\top x_n)] \}$$

However, this function is complex and we cannot find the simple solution as we did in Naive Bayes. So we need to use numerical methods.

• Numerical methods are messier, in contrast to cleaner closed-form solutions.
• In practice, we often have to tune a few optimization parameters — patience is necessary.
• A popular method: gradient descent

Finding the gradient of $E(w)$ looks very hard, but it turns out to be simple and intuitive.
How to find the optimal parameters for logistic regression?

We will minimize the error function

\[ \mathcal{E}(w) = -\sum_n \{ y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)] \} \]
How to find the optimal parameters for logistic regression?

We will minimize the error function

\[ \mathcal{E}(\mathbf{w}) = -\sum_n \{y_n \log \sigma(\mathbf{w}^\top \mathbf{x}_n) + (1 - y_n) \log[1 - \sigma(\mathbf{w}^\top \mathbf{x}_n)]\} \]

However, this function is complex and we cannot find the simple solution as we did in Naive Bayes. So we need to use \textit{numerical} methods.

- Numerical methods are messier, in contrast to cleaner closed-form solutions.
- In practice, we often have to tune a few optimization parameters — patience is necessary.
- A popular method: \textit{gradient descent}

Finding the gradient of \( \mathcal{E}(\mathbf{w}) \) looks very hard, but it turns out to be simple and intuitive
Simple fact: derivatives of $\sigma(a)$

$$
\frac{d}{da} \sigma(a) = \frac{d}{da} \left(1 + e^{-a}\right)^{-1}
$$

$$
= \frac{-1}{\left(1 + e^{-a}\right)^2} \frac{d}{da} (1 + e^{-a})
$$

$$
= \frac{e^{-a}}{(1 + e^{-a})^2}
$$

$$
= \frac{1}{1 + e^{-a}} \frac{e^{-a}}{1 + e^{-a} - 1}
$$

$$
= \sigma(a)[1 - \sigma(a)]
$$
Gradients of the cross-entropy error function

Cross-entropy Error Function

\[ E(w) = - \sum_n \{ y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)] \} \]

Remark

- \[ e_n = \{ \sigma(w^\top x_n) - y_n \} \]
  is called error for the \( n \)th training sample.
Gradients of the cross-entropy error function

Cross-entropy Error Function

\[ E(w) = - \sum_n \{ y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)] \} \]

Gradients

\[ \frac{\partial E(w)}{\partial w} = - \sum_n \left\{ y_n \frac{\sigma(w^\top x)}{\sigma(w^\top x)} [1 - \sigma(w^\top x_n)] x_n - (1 - y_n) \frac{1 - \sigma(w^\top x)}{1 - \sigma(w^\top x)} \sigma(w^\top x_n) x_n \right\} \]
Gradients of the cross-entropy error function

Cross-entropy Error Function

\[ E(w) = - \sum_n \{ y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)] \} \]

Gradients

\[ \frac{\partial E(w)}{\partial w} = - \sum_n \left\{ y_n \frac{\sigma(w^\top x)}{\sigma(w^\top x)} [1 - \sigma(w^\top x_n)] x_n - (1 - y_n) \frac{1 - \sigma(w^\top x)}{1 - \sigma(w^\top x)} \sigma(w^\top x_n) x_n \right\} \]

\[ = - \sum_n \{ y_n [1 - \sigma(w^\top x_n)] x_n - (1 - y_n) \sigma(w^\top x_n) x_n \} \]
Gradients of the cross-entropy error function

Cross-entropy Error Function

\[ E(w) = - \sum_n \{ y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)] \} \]

Gradients

\[
\frac{\partial E(w)}{\partial w} = - \sum_n \left\{ y_n \frac{\sigma(w^\top x)}{\sigma(w^\top x)} [1 - \sigma(w^\top x_n)] x_n - (1 - y_n) \frac{1 - \sigma(w^\top x)}{1 - \sigma(w^\top x)} \sigma(w^\top x_n) x_n \right\}
\]

\[
= - \sum_n \left\{ y_n [1 - \sigma(w^\top x_n)] x_n - (1 - y_n) \sigma(w^\top x_n) x_n \right\}
\]

\[
= \sum_n \left\{ \sigma(w^\top x_n) - y_n \right\} x_n
\]

Remark

\[ e_n = \begin{cases} \sigma(w^\top x_n) - y_n \end{cases} \] is called error for the \( n \)th training sample.
Gradients of the cross-entropy error function

Cross-entropy Error Function

$$\mathcal{E}(\mathbf{w}) = - \sum_{n} \{ y_n \log \sigma(\mathbf{w}^\top \mathbf{x}_n) + (1 - y_n) \log[1 - \sigma(\mathbf{w}^\top \mathbf{x}_n)] \}$$

Gradients

$$\frac{\partial \mathcal{E}(\mathbf{w})}{\partial \mathbf{w}} = - \sum_{n} \left\{ y_n \frac{\sigma(\mathbf{w}^\top \mathbf{x})}{\sigma(\mathbf{w}^\top \mathbf{x})} [1 - \sigma(\mathbf{w}^\top \mathbf{x}_n)] \mathbf{x}_n - (1 - y_n) \frac{1 - \sigma(\mathbf{w}^\top \mathbf{x})}{1 - \sigma(\mathbf{w}^\top \mathbf{x})} \sigma(\mathbf{w}^\top \mathbf{x}_n)] \mathbf{x}_n \right\}$$

$$= - \sum_{n} \left\{ y_n [1 - \sigma(\mathbf{w}^\top \mathbf{x}_n)] \mathbf{x}_n - (1 - y_n) \sigma(\mathbf{w}^\top \mathbf{x}_n)] \mathbf{x}_n \right\}$$

$$= \sum_{n} \left\{ \sigma(\mathbf{w}^\top \mathbf{x}_n) - y_n \right\} \mathbf{x}_n$$

Remark

- $e_n = \left\{ \sigma(\mathbf{w}^\top \mathbf{x}_n) - y_n \right\}$ is called error for the $n$th training sample.
Gradient descent for logistic regression

- Choose a proper step size $\eta > 0$
- Iteratively update the parameters following the negative gradient to minimize the error function

$$
\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \eta \sum_n \left\{ \sigma(\mathbf{w}^\top \mathbf{x}_n) - y_n \right\} \mathbf{x}_n
$$
Example: Spam Classification

<table>
<thead>
<tr>
<th></th>
<th>free</th>
<th>bank</th>
<th>meet</th>
<th>time</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email 1</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Ham</td>
</tr>
<tr>
<td>Email 4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Ham</td>
</tr>
</tbody>
</table>

Perform gradient descent to learn weights \( w \)
Example: Spam Classification

<table>
<thead>
<tr>
<th></th>
<th>free</th>
<th>bank</th>
<th>meet</th>
<th>time</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email 1</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Ham</td>
</tr>
<tr>
<td>Email 4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Ham</td>
</tr>
</tbody>
</table>

Perform gradient descent to learn weights $w$

- Feature vector for email 1 $x_1 = [1, 5, 3, 1, 1]^T$
Example: Spam Classification

<table>
<thead>
<tr>
<th></th>
<th>free</th>
<th>bank</th>
<th>meet</th>
<th>time</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email 1</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Ham</td>
</tr>
<tr>
<td>Email 4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Ham</td>
</tr>
</tbody>
</table>

Perform gradient descent to learn weights $w$

- Feature vector for email 1 $x_1 = [1, 5, 3, 1, 1]^T$
- Initial weights $w = [0.5, 0.5, 0.5, 0.5, 0.5]^T$
Perform gradient descent to learn weights $w$

- Feature vector for email 1 $x_1 = [1, 5, 3, 1, 1]^T$
- Initial weights $w = [0.5, 0.5, 0.5, 0.5, 0.5]^T$
- Prediction $\sigma(w^T x_1) = [0.996, 0.989, 0.989, 0.989]^T$
Perform gradient descent to learn weights $w$

- Prediction $\sigma(w^T x_1) = [0.996, 0.989, 0.989, 0.989]^T$
Example: Spam Classification, Batch Gradient Descent

<table>
<thead>
<tr>
<th></th>
<th>free</th>
<th>bank</th>
<th>meet</th>
<th>time</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email 1</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Ham</td>
</tr>
<tr>
<td>Email 4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Ham</td>
</tr>
</tbody>
</table>

Perform gradient descent to learn weights $w$

- Prediction $\sigma(w^T x_1) = [0.996, 0.989, 0.989, 0.989]^T$
- Difference from labels $y = [1, 1, 0, 0]^T$ is $[-0.004, -0.011, 0.989, 0.989]^T$
Perform gradient descent to learn weights $w$

- Prediction $\sigma(w^T x_1) = [0.996, 0.989, 0.989, 0.989]^T$
- Difference from labels $y = [1, 1, 0, 0]^T$ is $[-0.004, -0.011, 0.989, 0.989]^T$
- Gradient is $g_1 = (\sigma(w^T x_n) - y)x_1 = [1.96, 2.9, 2.93, 4.93, 4.93]$
Example: Spam Classification, Batch Gradient Descent

<table>
<thead>
<tr>
<th></th>
<th>free</th>
<th>bank</th>
<th>meet</th>
<th>time</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email 1</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Ham</td>
</tr>
<tr>
<td>Email 4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Ham</td>
</tr>
</tbody>
</table>

Perform gradient descent to learn weights $w$

- Prediction $\sigma(\mathbf{w}^\top \mathbf{x}_1) = [0.996, 0.989, 0.989, 0.989]^\top$
- Difference from labels $y = [1, 1, 0, 0]^\top$ is $[-0.004, -0.011, 0.989, 0.989]^\top$
- Gradient is $g_1 = (\sigma(\mathbf{w}^\top \mathbf{x}_n) - y)\mathbf{x}_1 = [1.96, 2.9, 2.93, 4.93, 4.93]$
- $\mathbf{w} \leftarrow \mathbf{w} - 0.01 \sum_n g_n$
Example: Spam Classification, Batch Gradient Descent

<table>
<thead>
<tr>
<th></th>
<th>free</th>
<th>bank</th>
<th>meet</th>
<th>time</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email 1</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Ham</td>
</tr>
<tr>
<td>Email 4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Ham</td>
</tr>
</tbody>
</table>

Predictions for Emails 3 and 4 are initially close to 1 (spam), but they converge towards the correct value 0 (ham)
Example: Spam Classification, Test Phase

<table>
<thead>
<tr>
<th></th>
<th>free</th>
<th>bank</th>
<th>meet</th>
<th>time</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email 1</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Ham</td>
</tr>
<tr>
<td>Email 4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Ham</td>
</tr>
</tbody>
</table>

- Final $\mathbf{w} = [0.187, 0.482, 0.179, -0.512, -0.524]^T$ after 50 batch gradient descent iterations
Example: Spam Classification, Test Phase

<table>
<thead>
<tr>
<th></th>
<th>free</th>
<th>bank</th>
<th>meet</th>
<th>time</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email 1</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Ham</td>
</tr>
<tr>
<td>Email 4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Ham</td>
</tr>
</tbody>
</table>

• Final $\mathbf{w} = [0.187, 0.482, 0.179, -0.512, -0.524]^\top$ after 50 batch gradient descent iterations

• Given a new email with feature vector $\mathbf{x} = [1, 1, 3, 4, 2]$, the probability of the email being spam is estimated as $\sigma(\mathbf{w}^\top \mathbf{x}) = \sigma(-1.889) = 0.13$
Example: Spam Classification, Test Phase

<table>
<thead>
<tr>
<th></th>
<th>free</th>
<th>bank</th>
<th>meet</th>
<th>time</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email 1</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 2</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Spam</td>
</tr>
<tr>
<td>Email 3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Ham</td>
</tr>
<tr>
<td>Email 4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>Ham</td>
</tr>
</tbody>
</table>

- Final $\mathbf{w} = [0.187, 0.482, 0.179, -0.512, -0.524]^\top$ after 50 batch gradient descent iterations
- Given a new email with feature vector $\mathbf{x} = [1, 1, 3, 4, 2]$, the probability of the email being spam is estimated as $\sigma(\mathbf{w}^\top \mathbf{x}) = \sigma(-1.889) = 0.13$
- Since this is less than 0.5 we predict ham
Contrast Naive Bayes and Logistic Regression

Both classification models are linear functions of features.

Joint vs. conditional distribution:
- Naive Bayes models the joint distribution:
  \[ P(X, Y) = P(Y) P(X|Y) \]
- Logistic regression models the conditional distribution:
  \[ P(Y|X) \]

Correlated vs. independent features:
- Naive Bayes assumes independence of features and multiple occurrences.
- Logistic Regression implicitly captures correlations when training weights.

Generative vs. Discriminative:
- Naive Bayes is a generative model.
- Logistic Regression is a discriminative model.
Both classification models are linear functions of features.
Contrast Naive Bayes and Logistic Regression

Both classification models are linear functions of features

**Joint vs. conditional dist**

Naive Bayes models the *joint* distribution: \( P(X, Y) = P(Y)P(X|Y) \)

Logistic regression models the *conditional* distribution: \( P(Y|X) \)
Both classification models are linear functions of features

**Joint vs. conditional dist**
Naive Bayes models the *joint* distribution: \( P(X, Y) = P(Y)P(X|Y) \)
Logistic regression models the *conditional* distribution: \( P(Y|X) \)

**Correlated vs. independent features**
Naive Bayes assumes independence of features and multiple occurrences
Logistic Regression implicitly captures correlations when training weights
Contrast Naive Bayes and Logistic Regression

Both classification models are linear functions of features

**Joint vs. conditional dist**

Naive Bayes models the *joint* distribution:  \( P(X, Y) = P(Y)P(X|Y) \)

Logistic regression models the *conditional* distribution:  \( P(Y|X) \)

**Correlated vs. independent features**

Naive Bayes assumes independence of features and multiple occurrences.
Logistic Regression implicitly captures correlations when training weights.

**Generative vs. Discriminative**

NB is a *generative* model, LR is a *discriminative* model.
Contrast Naive Bayes and Logistic Regression

Consider spam classification problem

- First Strategy:
  - Use training set to find a decision boundary in the feature space that separates spam and non-spam emails
  - Given a test point, predict its label based on which side of the boundary it is on.
Consider spam classification problem

• First Strategy:
  • Use training set to find a decision boundary in the feature space that separates spam and non-spam emails
  • Given a test point, predict its label based on which side of the boundary it is on.

• Second Strategy:
  • Look at spam emails and build a model of what they look like. Similarly, build a model of what non-spam emails look like.
  • To classify a new email, match it against both the spam and non-spam models to see which is the better fit.
Consider spam classification problem

**First Strategy:**
- Use training set to find a decision boundary in the feature space that separates spam and non-spam emails.
- Given a test point, predict its label based on which side of the boundary it is on.

**Second Strategy:**
- Look at spam emails and build a model of what they look like. Similarly, build a model of what non-spam emails look like.
- To classify a new email, match it against both the spam and non-spam models to see which is the better fit.

First strategy is discriminative (e.g., logistic regression)
Second strategy is generative (e.g., naive bayes)
Outline

1. Review of Logistic regression

2. Non-linear Decision Boundary

3. Multi-class Classification
   - Multi-class Naive Bayes
   - Multi-class Logistic Regression
Non-linear Decision Boundary
How to handle more complex decision boundaries?

This data is not linear separable

- Use non-linear basis functions to add more features
How to handle more complex decision boundaries?

- This data is not linear separable
How to handle more complex decision boundaries?

- This data is not linear separable
- Use non-linear basis functions to add more features
Adding polynomial features

- New feature vector is \( x = [1, x_1, x_2, x_1^2, x_2^2] \)
Adding polynomial features

- New feature vector is \( x = [1, x_1, x_2, x_1^2, x_2^2] \)
- \( \Pr(y = 1|\mathbf{x}) = \sigma(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2) \)
Adding polynomial features

- New feature vector is $\mathbf{x} = [1, x_1, x_2, x_1^2, x_2^2]$
- $\Pr(y = 1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2)$
- If $\mathbf{w} = [-1, 0, 0, 1, 1]$, the boundary is $-1 + x_1^2 + x_2^2 = 0$
Adding polynomial features

- New feature vector is \( \mathbf{x} = [1, x_1, x_2, x_1^2, x_2^2] \)
- \( \Pr(y = 1|\mathbf{x}) = \sigma(w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2) \)
- If \( \mathbf{w} = [-1, 0, 0, 1, 1] \), the boundary is \(-1 + x_1^2 + x_2^2 = 0\)
  - If \(-1 + x_1^2 + x_2^2 \geq 0\) declare spam
Adding polynomial features

- New feature vector is \( \mathbf{x} = [1, x_1, x_2, x_1^2, x_2^2] \)
- \( \Pr(y = 1|\mathbf{x}) = \sigma(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2) \)
- If \( \mathbf{w} = [-1, 0, 0, 1, 1] \), the boundary is \(-1 + x_1^2 + x_2^2 = 0\)
  - If \(-1 + x_1^2 + x_2^2 \geq 0\) declare spam
  - If \(-1 + x_1^2 + x_2^2 < 0\) declare ham
• New feature vector is $x = [1, x_1, x_2, x_1^2, x_2^2]$

• $Pr(y = 1|x) = \sigma(w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2)$

• If $w = [-1, 0, 0, 1, 1]$, the boundary is $-1 + x_1^2 + x_2^2 = 0$
  • If $-1 + x_1^2 + x_2^2 \geq 0$ declare spam
  • If $-1 + x_1^2 + x_2^2 < 0$ declare ham
Adding polynomial features

- New feature vector is \( \mathbf{x} = [1, x_1, x_2, x_1^2, x_2^2] \)
- \( \Pr(y = 1|\mathbf{x}) = \sigma(w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2) \)
- If \( \mathbf{w} = [-1, 0, 0, 1, 1] \), the boundary is \(-1 + x_1^2 + x_2^2 = 0\)
  - If \(-1 + x_1^2 + x_2^2 \geq 0\) declare spam
  - If \(-1 + x_1^2 + x_2^2 < 0\) declare ham

\(-1 + x_1^2 + x_2^2 = 0\)
Adding polynomial features

- What if we add many more features and define
  \[ \mathbf{x} = [1, x_1, x_2, x_1^2, x_2^2, x_1^3, x_2^3, \ldots] \]?
Adding polynomial features

- What if we add many more features and define
  \[ x = [1, x_1, x_2, x_1^2, x_2^2, x_1^3, x_2^3, \ldots] \]?
- We get a complex decision boundary
Adding polynomial features

- What if we add many more features and define
  \[ \mathbf{x} = [1, x_1, x_2, x_1^2, x_2^2, x_1^3, x_2^3, \ldots] \]?
- We get a complex decision boundary
Adding polynomial features

- What if we add many more features and define
  \[ x = [1, x_1, x_2, x_1^2, x_2^2, x_1^3, x_2^3, \ldots] \]?
- We get a complex decision boundary
Adding polynomial features

- What if we add many more features and define
  \[ x = [1, x_1, x_2, x_1^2, x_2^2, x_1^3, x_2^3, \ldots] \]?
- We get a complex decision boundary

Can result in overfitting and bad generalization to new data points
Concept-check: Bias-Variance Trade-off

First boundary has high bias, and the second has high variance.
Concept-check: Bias-Variance Trade-off

First boundary has high bias, and the second has high variance
Solution to Overfitting: Regularization

- Add regularization term to be cross entropy loss function

\[ E(w) = - \sum_n \{ y_n \log \sigma(w^\top x_n) + (1-y_n) \log[1-\sigma(w^\top x_n)] \} + \frac{1}{2} \lambda \|w\|^2 \]

regularization
Solution to Overfitting: Regularization

• Add regularization term to be cross entropy loss function

\[ E(w) = -\sum_n \{y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)]\} + \frac{1}{2} \lambda \| w \|_2^2 \]

• Perform gradient descent on this regularized function
Solution to Overfitting: Regularization

- Add regularization term to be cross entropy loss function

\[ E(w) = -\sum_n \{y_n \log \sigma (w^\top x_n) + (1-y_n) \log [1-\sigma (w^\top x_n)]\} + \frac{1}{2} \lambda \|w\|_2^2 \]

- Perform gradient descent on this regularized function
- Often, we do NOT regularize the bias term \( w_0 \) (you will see this in the homework)
Solution to Overfitting: Regularization

- Add regularization term to be cross entropy loss function

\[
E(w) = - \sum_n \{y_n \log \sigma(w^\top x_n) + (1 - y_n) \log[1 - \sigma(w^\top x_n)]\} + \frac{1}{2} \lambda \|w\|_2^2
\]

- Perform gradient descent on this regularized function
- Often, we do NOT regularize the bias term \(w_0\) (you will see this in the homework)
Solution to Overfitting: Regularization

- Add regularization term to be cross entropy loss function

\[ E(w) = -\sum_n \{ y_n \log \sigma(w^\top x_n) + (1-y_n) \log[1-\sigma(w^\top x_n)] \} + \frac{1}{2\lambda} \|w\|^2 \]

- Perform gradient descent on this regularized function
- Often, we do NOT regularize the bias term \( w_0 \) (you will see this in the homework)
1. Review of Logistic regression

2. Non-linear Decision Boundary

3. Multi-class Classification
   - Multi-class Naive Bayes
   - Multi-class Logistic Regression
Multi-class Classification
What if there are more than 2 classes?

- Dog vs. cat. vs crocodile

![Scatter plot with three distinct clusters representing different classes.](image)
What if there are more than 2 classes?

- Dog vs. cat. vs crocodile
- Movie genres (action, horror, comedy, ...)

\[
x_1 \quad x_2
\]
What if there are more than 2 classes?

- Dog vs. cat. vs crocodile
- Movie genres (action, horror, comedy, ...)
- Part of speech tagging (verb, noun, adjective, ...)

![Data points in a 2D scatter plot](image)
What if there are more than 2 classes?

- Dog vs. cat. vs crocodile
- Movie genres (action, horror, comedy, . . .)
- Part of speech tagging (verb, noun, adjective, . . .)
- . . .
Predict multiple classes/outcomes $C_1, C_2, \ldots, C_K$:

- Weather prediction: sunny, cloudy, raining, etc
- Optical character recognition: 10 digits + 26 characters (lower and upper cases) + special characters, etc.

Methods we’ve studied for binary classification:

- Naive Bayes
- Logistic regression
1. Review of Logistic regression

2. Non-linear Decision Boundary

3. Multi-class Classification
   - Multi-class Naive Bayes
   - Multi-class Logistic Regression
**Formal Definition**

Given a random vector $\mathbf{X} \in \mathbb{R}^K$ and a dependent variable $Y \in [C]$, the Naive Bayes model defines the joint distribution

$$P(\mathbf{X} = \mathbf{x}, Y = c) = P(Y = c)P(\mathbf{X} = \mathbf{x} | Y = c)$$

(1)

$$= P(Y = c) \prod_{k=1}^{K} P(\text{word}_k | Y = c)^{x_k}$$

(2)

$$= \pi_c \prod_{k=1}^{K} \theta_{ck}^{x_k}$$

(3)

where $x_k$ is the number of occurrences of the $k$th word, $\pi_c$ is the prior probability of class $c$, and $\theta_{ck}$ is the weight of the $k$th word for the $c$th class.
Learning problem

Training data

$$\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N \rightarrow \mathcal{D} = \{\{x_{nk}\}_{k=1}^K, y_n\}_{n=1}^N$$

Goal

Learn $\pi_c, c = 1, 2, \cdots, C,$ and $\theta_{ck}, \forall c \in [C], k \in [K]$ under the constraints:

$$\sum_c \pi_c = 1$$

and

$$\sum_k \theta_{ck} = \sum_k P(\text{word}_k|Y = c) = 1$$

as well as $\pi_c, \theta_{ck} \geq 0.$
Log-Likelihood of the training data

\[
\mathcal{L} = \log P(D) = \log \prod_{n=1}^{N} \pi_{y_n} P(x_n | y_n)
\]

\[
= \log \prod_{n=1}^{N} \left( \pi_{y_n} \prod_{k} \theta_{y_n k}^{x_{nk}} \right)
\]

\[
= \sum_{n} \left( \log \pi_{y_n} + \sum_{k} x_{nk} \log \theta_{y_n k} \right)
\]

\[
= \sum_{n} \log \pi_{y_n} + \sum_{n,k} x_{nk} \log \theta_{y_n k}
\]

Optimize it!

\[
(\pi^*_c, \theta^*_c) = \arg \max \sum_{n} \log \pi_{y_n} + \sum_{n,k} x_{nk} \log \theta_{y_n k}
\]
Our hammer: maximum likelihood estimation

**Optimization Problem**

\[(\pi^*_c, \theta^*_{ck}) = \operatorname{arg\ max} \sum_n \log \pi_{y,n} + \sum_{n,k} x_{nk} \log \theta_{y,n,k}\]
Our hammer: maximum likelihood estimation

Optimization Problem

\[(\pi^*_c, \theta^*_ck) = \arg \max \sum_n \log \pi_{yn} + \sum_{n,k} x_{nk} \log \theta_{yn,k}\]

Solution

\[
\theta^*_ck = \frac{\text{#of times word } k \text{ shows up in data points labeled as } c}{\text{#total trials for data points labeled as } c}
\]

\[
\pi^*_c = \frac{\text{#of data points labeled as } c}{N}
\]
1. Review of Logistic regression

2. Non-linear Decision Boundary

3. Multi-class Classification
   Multi-class Naive Bayes
   Multi-class Logistic Regression
Logistic regression for predicting multiple classes?

- The linear decision boundary that we optimized was specific to binary classification.

\[ \sigma(w^\top x) \geq 0.5 \text{ declare } y = 1 \text{ (spam)} \]

\[ \sigma(w^\top x) < 0.5 \text{ declare } y = 0 \text{ (ham)} \]

How to extend it to multi-class classification?

\[ w^\top x, \text{ Linear comb. of features} \]

\[ \text{Prob}(y=1|x) \]

\[ y = 1 \text{ for spam, } y = 0 \text{ for ham} \]
Logistic regression for predicting multiple classes?

- The linear decision boundary that we optimized was specific to binary classification.
  - If $\sigma(w^T x) \geq 0.5$ declare $y = 1$ (spam)

\[
\begin{align*}
\text{SPAM} & \quad \text{HAM} \\
\text{Prob}(y=1|x) & \\
& (w^T x, \text{Linear comb. of features})
\end{align*}
\]

- $y = 1$ for spam, $y = 0$ for ham
Logistic regression for predicting multiple classes?

- The linear decision boundary that we optimized was specific to binary classification.
  - If $\sigma(w^T x) \geq 0.5$ declare $y = 1$ (spam)
  - If $\sigma(w^T x) < 0.5$ declare $y = 0$ (ham)

$y = 1$ for spam, $y = 0$ for ham
Logistic regression for predicting multiple classes?

- The linear decision boundary that we optimized was specific to binary classification.
  - If $\sigma(w^T x) \geq 0.5$ declare $y = 1$ (spam)
  - If $\sigma(w^T x) < 0.5$ declare $y = 0$ (ham)
- How to extend it to multi-class classification?

\[ y = 1 \text{ for spam, } y = 0 \text{ for ham} \]
Logistic regression for predicting multiple classes?

- The linear decision boundary that we optimized was specific to binary classification.
  - If $\sigma(w^\top x) \geq 0.5$ declare $y = 1$ (spam)
  - If $\sigma(w^\top x) < 0.5$ declare $y = 0$ (ham)
- How to extend it to multi-class classification?

$y = 1$ for spam, $y = 0$ for ham
The linear decision boundary that we optimized was specific to binary classification.

- If $\sigma(w^\top x) \geq 0.5$ declare $y = 1$ (spam)
- If $\sigma(w^\top x) < 0.5$ declare $y = 0$ (ham)

How to extend it to multi-class classification?

Idea: Express as multiple binary classification problems
The One-versus-Rest or One-Versus-All Approach

- For each class $C_k$, change the problem into binary classification
  1. Relabel training data with label $C_k$, into POSITIVE (or ‘1’)
  2. Relabel all the rest data into NEGATIVE (or ‘0’)
- Repeat this multiple times: Train $K$ binary classifiers, using logistic regression to differentiate the two classes each time
The One-versus-Rest or One-Versus-All Approach

- For each class $C_k$, change the problem into binary classification
  1. Relabel training data with label $C_k$, into POSITIVE (or ‘1’)
  2. Relabel all the rest data into NEGATIVE (or ‘0’)
- Repeat this multiple times: Train $K$ binary classifiers, using logistic regression to differentiate the two classes each time
The One-versus-Rest or One-Versus-All Approach

- For each class $C_k$, change the problem into binary classification
  1. Relabel training data with label $C_k$, into **POSITIVE** (or ‘1’)
  2. Relabel all the rest data into **NEGATIVE** (or ‘0’)
- Repeat this multiple times: Train $K$ binary classifiers, using logistic regression to differentiate the two classes each time
How to combine these linear decision boundaries?

- There is ambiguity in some of the regions (the 4 triangular areas)
How to combine these linear decision boundaries?

- There is ambiguity in some of the regions (the 4 triangular areas)
- How do we resolve this?
The One-versus-Rest or One-Versus-All Approach

How to combine these linear decision boundaries?

- Use the confidence estimates $\Pr(y = C_1 | x) = \sigma(w_1^\top x)$,
  $\ldots$ $\Pr(y = C_K | x) = \sigma(w_K^\top x)$
- Declare class $C_k^*$ that maximizes
  $$k^* = \arg \max_{k=1,\ldots,K} \Pr(y = C_k | x) = \sigma(w_k^\top x)$$
The One-Versus-One Approach

- For each **pair** of classes $C_k$ and $C_{k'}$, change the problem into binary classification
  1. Relabel training data with label $C_k$, into **POSITIVE** (or ‘1’)
  2. Relabel training data with label $C_{k'}$ into **NEGATIVE** (or ‘0’)
  3. **Disregard** all other data
The One-Versus-One Approach

- How many binary classifiers for $K$ classes?
• How many binary classifiers for $K$ classes?
The One-Versus-One Approach

- How many binary classifiers for $K$ classes? $K(K - 1)/2$
- How to combine their outputs?

Given $x$, count the $K(K - 1)/2$ votes from outputs of all binary classifiers and declare the winner as the predicted class. Use confidence scores to resolve ties.
The One-Versus-One Approach

- How many binary classifiers for $K$ classes? $K(K - 1)/2$
- How to combine their outputs?
- Given $x$, count the $K(K - 1)/2$ votes from outputs of all binary classifiers and declare the winner as the predicted class.
The One-Versus-One Approach

- How many binary classifiers for $K$ classes? $\frac{K(K - 1)}{2}$
- How to combine their outputs?
- Given $x$, count the $\frac{K(K - 1)}{2}$ votes from outputs of all binary classifiers and declare the winner as the predicted class.
- Use confidence scores to resolve ties
Number of Binary Classifiers to be trained

- **One-Versus-All**: $K$ classifiers.
- **One-Versus-One**: $K(K - 1)/2$ classifiers – bad if $K$ is large
Contrast these approaches

**Number of Binary Classifiers to be trained**

- **One-Versus-All**: $K$ classifiers.
- **One-Versus-One**: $K(K - 1)/2$ classifiers – bad if $K$ is large

**Effect of Relabeling and Splitting Training Data**

- **One-Versus-All**: imbalance in the number of positive and negative samples can cause bias in each trained classifier
- **One-Versus-One**: each classifier trained on a small subset of data (only those labeled with those two classes would be involved), which can result in high variance
Contrast these approaches

**Number of Binary Classifiers to be trained**

- **One-Versus-All**: $K$ classifiers.
- **One-Versus-One**: $K(K - 1)/2$ classifiers – bad if $K$ is large

**Effect of Relabeling and Splitting Training Data**

- **One-Versus-All**: imbalance in the number of positive and negative samples can cause bias in each trained classifier
- **One-Versus-One**: each classifier trained on a small subset of data (only those labeled with those two classes would be involved), which can result in high variance

Any other ideas?
Contrast these approaches

Number of Binary Classifiers to be trained

- **One-Versus-All**: $K$ classifiers.
- **One-Versus-One**: $K(K - 1)/2$ classifiers – bad if $K$ is large

Effect of Relabeling and Splitting Training Data

- **One-Versus-All**: imbalance in the number of positive and negative samples can cause bias in each trained classifier
- **One-Versus-One**: each classifier trained on a small subset of data (only those labeled with those two classes would be involved), which can result in high variance

Any other ideas?

- Hierarchical classification – we will see this in decision trees
Contrast these approaches

**Number of Binary Classifiers to be trained**

- **One-Versus-All**: $K$ classifiers.
- **One-Versus-One**: $K(K - 1)/2$ classifiers – bad if $K$ is large

**Effect of Relabeling and Splitting Training Data**

- **One-Versus-All**: imbalance in the number of positive and negative samples can cause bias in each trained classifier
- **One-Versus-One**: each classifier trained on a small subset of data (only those labeled with those two classes would be involved), which can result in high variance

**Any other ideas?**

- Hierarchical classification – we will see this in decision trees
- Multinomial Logistic Regression – directly output probabilities of $y$ being in each of the $K$ classes, instead of reducing to a binary classification problem.
Intuition:

from the decision rule of our naive Bayes classifier

\[
y^* = \arg \max_k p(y = C_k | x) = \arg \max_k \log p(x | y = C_k) p(y = C_k)
= \arg \max_k \log \pi_k + \sum_i x_i \log \theta_{ki} = \arg \max_k w_k^\top x
\]
Intuition:
from the decision rule of our naive Bayes classifier

\[ y^* = \arg \max_k p(y = C_k | x) = \arg \max_k \log p(x | y = C_k) p(y = C_k) \]
\[ = \arg \max_k \log \pi_k + \sum_i x_i \log \theta_{ki} = \arg \max_k \mathbf{w}_k^T \mathbf{x} \]

Essentially, we are comparing

\[ \mathbf{w}_1^T \mathbf{x}, \mathbf{w}_2^T \mathbf{x}, \cdots, \mathbf{w}_K^T \mathbf{x} \]

with one for each category.
So, can we define the following conditional model?

\[ p(y = C_k | x) = \sigma[w_k^\top x]. \]
So, can we define the following conditional model?

\[ p(y = C_k | x) = \sigma[w_k^\top x]. \]

This would not work because:

\[ \sum_k p(y = C_k | x) = \sum_k \sigma[w_k^\top x] \neq 1. \]

each summand can be any number (independently) between 0 and 1.

But we are close!
So, can we define the following conditional model?

\[ p(y = C_k | x) = \sigma [ w_k^\top x ] \].

This would not work because:

\[ \sum_k p(y = C_k | x) = \sum_k \sigma [ w_k^\top x ] \neq 1. \]

Each summand can be any number (independently) between 0 and 1. But we are close!

Learn the \( K \) linear models jointly to ensure this property holds!
Multinomial logistic regression

- Model: For each class $C_k$, we have a parameter vector $w_k$ and model the posterior probability as:

$$ p(C_k|x) = \frac{e^{w_k^\top x}}{\sum_{k'} e^{w_{k'}^\top x}} \quad ⇐ \quad \text{This is called softmax function} $$

- Decision boundary: Assign $x$ with the label that is the maximum of posterior:

$$ \arg \max_k P(C_k|x) \rightarrow \arg \max_k w_k^\top x. $$
How does the softmax function behave?

Suppose we have

$$w_1^T x = 100, \quad w_2^T x = 50, \quad w_3^T x = -20.$$
How does the softmax function behave?

Suppose we have

\[ w_1^\top x = 100, \quad w_2^\top x = 50, \quad w_3^\top x = -20. \]

We would pick the **winning** class label 1.

**Softmax translates these scores into well-formed conditional probabilities**

\[ p(y = 1|x) = \frac{e^{100}}{e^{100} + e^{50} + e^{-20}} < 1 \]

- preserves relative ordering of scores
- maps scores to values between 0 and 1 that also sum to 1
Multinomial model reduce to binary logistic regression when $K = 2$

$$p(C_1|x) = \frac{e^{w_1^T x}}{e^{w_1^T x} + e^{w_2^T x}} = \frac{1}{1 + e^{-(w_1 - w_2)^T x}}$$

Multinomial thus generalizes the (binary) logistic regression to deal with multiple classes.
**Discriminative approach**: maximize conditional likelihood

\[
\log P(D) = \sum_n \log P(y_n|x_n)
\]
**Discriminative approach:** maximize conditional likelihood

\[
\log P(D) = \sum_n \log P(y_n | x_n)
\]

We will change \( y_n \) to \( y_n = [y_{n1} \ y_{n2} \ \cdots \ y_{nK}]^\top \), a \( K \)-dimensional vector using 1-of-\( K \) encoding.

\[
y_{nk} = \begin{cases} 
1 & \text{if } y_n = k \\
0 & \text{otherwise}
\end{cases}
\]

Ex: if \( y_n = 2 \), then, \( y_n = [0 \ 1 \ 0 \ 0 \ \cdots \ 0]^\top \).
**Discriminative approach:** maximize conditional likelihood

\[
\log P(D) = \sum_n \log P(y_n | x_n)
\]

We will change \(y_n\) to \(y_n = [y_{n1} \ y_{n2} \ \cdots \ y_{nK}]^\top\), a \(K\)-dimensional vector using 1-of-\(K\) encoding.

\[
y_{nk} = \begin{cases} 
1 & \text{if } y_n = k \\
0 & \text{otherwise}
\end{cases}
\]

Ex: if \(y_n = 2\), then, \(y_n = [0 \ 1 \ 0 \ 0 \ \cdots \ 0]^\top\).

\[
\Rightarrow \sum_n \log P(y_n | x_n) = \sum_n \log \prod_{k=1}^K P(C_k | x_n)^{y_{nk}} = \sum_n \sum_k y_{nk} \log P(C_k | x_n)
\]
Cross-entropy error function

**Definition:** negative log likelihood

\[
\mathcal{E}(\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K) = - \sum_n \sum_k y_{nk} \log P(C_k | x_n)
\]

\[
= - \sum_n \sum_k y_{nk} \log \left( \frac{e^{\mathbf{w}_k^\top x_n}}{\sum_{k'} e^{\mathbf{w}_{k'}^\top x_n}} \right)
\]
**Definition**: negative log likelihood

\[
\mathcal{E}(w_1, w_2, \ldots, w_K) = -\sum_n\sum_k y_{nk} \log P(C_k | x_n)
\]

\[
= -\sum_n\sum_k y_{nk} \log \left( \frac{e^{w_k^\top x_n}}{\sum_{k'} e^{w_{k'}^\top x_n}} \right)
\]

**Properties**

- Convex, therefore unique global optimum
- Optimization requires numerical procedures, analogous to those used for binary logistic regression
You should know

- What is logistic regression and solving for $w$ using gradient descent on the cross entropy loss function
- Difference between Naive Bayes and Logistic Regression
- How to solve for the model parameters using gradient descent
- How to handle multiclass classification: one-versus-all, one-versus-one, multinomial regression