Discriminative classifier: Logistic Regression
Probabilistic Classifier

• Given an instance \( x \), what does a probabilistic classifier do differently compared to, say, perceptron?
• It does not directly predict \( y \)
• Instead, it first computes the probability that the instance belongs to different classes, i.e., \( P(y|x) \) – the posterior probability of \( y \) given \( x \)
• Given \( p(y|x) \), it then makes a prediction using decision theory
Decision Theory

• Goal 1: Minimizing the probability of mistake
  
  \[ y^* = \arg \max_y P(y|x) \]
  
  – i.e., predict the class that has the maximum posterior probability

• Goal 2: minimizing the expected loss

  » Given a cost matrix specifying the cost of different types of mistakes

<table>
<thead>
<tr>
<th>True label→Predicted ↓</th>
<th>Spam</th>
<th>Non-spam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spam</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Non-spam</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ y^* = \arg \min_y \sum_{y,y'} L(y,y') P(y'|x) \]

Expected cost if we predict \( y \)
Example

• Suppose our probabilistic spam-filter gives the following posterior for an incoming email $x$:
  
  $\quad P(y = \text{spam}|x) = 0.6$

• The expected cost if predict spam?

<table>
<thead>
<tr>
<th>True label $\rightarrow$ Predicted</th>
<th>Spam</th>
<th>Non-spam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spam</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Non-spam</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• What if we predict non-spam?

\[
y^* = \arg \min_y \sum_{y' \in \{s, ns\}} L(y, y')P(y'|x) = ?\]
Example

• Suppose our probabilistic spam-filter gives the following posterior for an incoming email x:
  \[ P(y = \text{spam}|x) = 0.6 \]

  - The expected cost if predict spam?
    - If it is a spam: no cost (0.6 prob)
    - If it is not – cost of 10 (0.4 prob)
    \[ 0.6 \times 0 + 0.4 \times 10 = 4 \]

  - What if we predict non-spam?
    - \[ 0.6 \times 1 + 0.4 \times 0 = 0.6 \]

<table>
<thead>
<tr>
<th>True label → Predicted</th>
<th>Spam</th>
<th>Non-spam</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spam</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>Non-spam</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

\[ y^* = \arg \min_y \sum_{y' \in \{s, ns\}} L(y, y')P(y'|x) = \text{non-spam} \]
Rejection option

• For some applications, we can have a rejection option
• Medical diagnosis

<table>
<thead>
<tr>
<th>True label</th>
<th>Predicted</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>0</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>100</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Reject</td>
<td>10</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>
Two main approaches

- To learn a probabilistic classifier, there are two types of approaches

  - **Generative:**
    - Learn $P(y)$ and $P(x|y)$
    - Compute $P(y|x)$ using Bayes rule
      \[
      P(y|x) = \frac{P(x|y)P(y)}{P(x)} = \frac{P(x|y)P(y)}{\sum_y P(x, y)}
      \]

  - **Discriminative:**
    - Learn $P(y|x)$ directly
    - Logistic regression is one of such techniques
Logistic Regression

• Given training set D, logistic regression directly learns the conditional distribution $P(y \mid x)$

• We will assume only two classes $y \in \{0, 1\}$ and a parametric form for $P(y = 1 \mid x; w)$ where $w$ is the parameter vector

$$P(y = 1 \mid x; w) = p_1(x) = \frac{1}{1 + e^{-w^T x}}$$

$$P(y = 0 \mid x; w) = 1 - p_1(x)$$

• It is easy to show that this is equivalent to

$$\log \frac{P(y = 1 \mid x; w)}{P(y = 0 \mid x; w)} = w^T x$$

• i.e. the log odds of class 1 is a linear function of $x$. 
The Logistic (Sigmoid) Function

\[ g(x, w) = \frac{1}{1 + \exp(-w^T x)} \]

The output of a linear function \( w^T x \) has range \((-\infty, \infty)\). A logistic sigmoid function transforms the value of \( w^T x \) into a range between 0 and 1.
Logistic Regression Yields a Linear Classifier

• Given $P(y \mid x)$, suppose we use the decision rule for minimizing classification error: i.e., predict $y^* = 1$ if $P(y = 1 \mid x) > P(y = 0 \mid x)$
  – More generally, $P(y = 1 \mid x) > \theta$, where $\theta$ is a threshold
  – Depending on the loss function, $\theta$ can be different values

• This yields a linear classifier

$$P(y = 1 \mid x) > P(y = 0 \mid x) \Rightarrow \frac{P(y = 1 \mid x)}{P(y = 0 \mid x)} > 1$$
$$\Rightarrow \log \frac{P(y = 1 \mid x)}{P(y = 0 \mid x)} > 0 \Rightarrow w^T x > 0$$

For more general decision rule, this will be replaced with a different threshold $w_0$
Learning Setup

• Given a set of training instances: 
  \((x^1, y^1), \ldots, (x^N, y^N)\)

• We assume that \(x\) and \(y\) are probabilistically related (parameterized by \(w\))

\[
P(y = 1|x; w) = \frac{1}{1 + \exp(-w^T x)}
\]

• Goal: learning \(w\) from the training data using Maximum Likelihood Estimation
Likelihood Function

• We assume each training example \((x^i, y^i)\) is drawn \(\text{IID}\) from the same (but unknown) distribution \(P(x, y)\):

\[
\log P(D; w) = \log \prod_i P(x^i, y^i; w) = \sum_i \log P(x^i, y^i; w)
\]

• Joint distribution \(P(a, b)\) can be factored as \(P(a \mid b)P(b)\)

\[
\text{arg max } \log P(D; w) = \text{arg max } \sum_i \log P(x^i, y^i; w)
\]

\[
= \text{arg max } \sum_i \log P(y^i \mid x^i; w)P(x^i; w)
\]

• Further, \(P(x; w)\) can be dropped because it does not depend on \(w\):

\[
\text{arg max } \log P(D; w) = \text{arg max } \sum_i \log P(y^i \mid x^i; w)
\]
Computing the Likelihood

\[
\arg \max_w \log P(D \mid w) = \arg \max_w \sum_{i} \log P(y^i \mid x^i, w)
\]

Let

\[
\hat{y}^i = p(y^i = 1 \mid x^i; w) = \frac{1}{1 + e^{-w \cdot x^i}}
\]

\[
p(y^i = 0 \mid x^i; w) = 1 - \hat{y}^i
\]

This can be compactly written as

\[
p(y^i \mid x^i; w) = (\hat{y}^i)^{y^i} (1 - \hat{y}^i)^{(1-y^i)}
\]

We will take our learning objective function to be:

\[
l(w) = \sum_i \log P(y^i \mid x^i, w)
\]

\[
= \sum_i [y^i \log \hat{y}^i + (1 - y^i) \log(1 - \hat{y}^i)]
\]
Gradient Ascent

\[ l(w) = \sum_i \log P(y^i | x^i; w) = \sum_i [y^i \log \hat{y}^i + (1 - y^i) \log(1 - \hat{y}^i)] \]

\[ \frac{\partial l_i(w)}{\partial w_j} = \frac{\partial}{\partial w_j} [y^i \log \hat{y}^i + (1 - y^i) \log(1 - \hat{y}^i)] \]

\[ = \frac{y^i}{\hat{y}^i} \left( \frac{\partial \hat{y}^i}{\partial w_j} \right) + \frac{1 - y^i}{1 - \hat{y}^i} \left( - \frac{\partial \hat{y}^i}{\partial w_j} \right) = \left[ \frac{y^i}{\hat{y}^i} - \frac{1 - y^i}{1 - \hat{y}^i} \right] \frac{\partial \hat{y}^i}{\partial w_j} = \left[ \frac{y^i - \hat{y}^i}{\hat{y}^i (1 - \hat{y}^i)} \right] \frac{\partial \hat{y}^i}{\partial w_j} \]

Recall that

\[ \hat{y}^i = \frac{1}{1 + \exp(-w^T x^i)} \]

\[ \frac{\partial \hat{y}^i}{\partial w_j} = \hat{y}^i (1 - \hat{y}^i) \frac{\partial (w^T x^i)}{\partial w_j} = \hat{y}^i (1 - \hat{y}^i) x_j^i \]

for \( g(t) = \frac{1}{1 + \exp(-t)} \) we have

\[ g'(t) = \frac{\exp(-t)}{(1 + \exp(-t))^2} = g(t)(1 - g(t)) \]

Small tip

\[ \frac{\partial l_i(w)}{\partial w_j} = (y^i - \hat{y}^i) x_j^i \]

\[ \frac{\partial l(w)}{\partial w_j} = \sum_{i=1}^{N} (y^i - \hat{y}^i) x_j^i \]

\[ \nabla l(w) = \sum_{i=1}^{N} (y^i - \hat{y}^i) x^i \]
Batch Gradient Ascent for LR

Given: training examples \((x^i, y^i), i = 1, ..., N\)

Let \(w \leftarrow (0,0,0,...,0)\)

Repeat until convergence

\[
\begin{align*}
    d &\leftarrow (0,0,0,...,0) \\
    \text{For } i &= 1 \text{ to } N \text{ do} \\
    \hat{y}^i &\leftarrow \frac{1}{1 + e^{-w^T x^i}} \\
    \text{error} &\leftarrow y^i - \hat{y}^i \\
    d &= d + \text{error} \cdot x^i \\
    w &\leftarrow w + \eta d
\end{align*}
\]

- **Online** gradient ascent algorithm can be easily constructed
Connection Between Logistic Regression & Perceptron Algorithm

- Both methods learn a linear function of the input features
- LR uses the logistic function, Perceptron uses a step function

\[
h_w(x) = \frac{1}{1 + e^{-w \cdot x}}
\]

\[
h_w(x) = \begin{cases} 
1 & \text{if } w \cdot x > 0 \\
0 & \text{otherwise}
\end{cases}
\]

- Both algorithms take a similar update rule:

\[w = w + \eta (y^i - h_w(x^i))x^i\]
Multi-Class Classification

- **One vs. rest**
  - build $K-1$ binary classifiers
  - Each classifier separates class $k$ from the rest, for $k = 1, \ldots, K - 1$
  - If all classifier says no, then predict class $K$

- **Pairwise**
  - Build $\binom{K}{2}$ classifiers
  - Each classifier separate class $i$ from $j$, for $i, j = 1, \ldots, K$ and $i \neq j$
  - Vote for the final prediction

*One vs. rest* and *Pairwise* diagrams are shown in the image.
Multi-Class Logistic Regression

• We define the posterior probability using a so-called softmax function

\[
p(y = k | x) = \hat{y}_k = \frac{\exp(\alpha_k)}{\sum_{j=1}^{K} \exp \alpha_j}
\]

where \( \alpha_k \) is given by

\[
\alpha_k = w_k^T x
\]

• Going through the same MLE derivations, we arrive at the following gradient:

\[
\nabla_{w_k} L = \sum_{i=1}^{N} (y_k^i - \hat{y}_k^i) x^i
\]

where \( y_k^i = 1 \) if \( y^i = k \), and 0 otherwise
Bayesian VS Frequentist

• When it comes to parameter estimation, there are two different statistical views
  – Frequentist: parameter is deterministic, it takes an unknown value
  – Bayesian: parameter is a random variable with a unknown distribution

• We can express our belief about the parameter using priors
• After observing the data, we can update our belief to obtain the posterior distribution of the parameter

\[
p(\theta|D) = \frac{p(\theta)p(D|\theta)}{p(D)} = \frac{p(\theta)p(D|\theta)}{\int p(D|\theta)p(\theta)d\theta}
\]
Maximum A Posterior (MAP) as a penalty method

\[ \hat{\theta}_{\text{map}} = \arg \max_{\theta} p(\theta|D) \]

\[ = \arg \max_{\theta} p(D|\theta)p(\theta) \]

\[ = \arg \max_{\theta} \log p(D|\theta) + \log p(\theta) \]

\textit{penalty}
MAP for Logistic Regression

$$\arg\max_w P(w|D) = \arg\max_w P(D|w) P(w)$$
$$= \arg\max_w \log P(D|w) + \log P(w)$$

- $P(D|w)$: the likelihood of $w$
  $$\sum_i \log P(y^i | x^i, w)$$

- $P(w)$: prior distribution $N(0, \sigma^2)$ for $i = 1, \ldots, d$
  - Large weight values correspond to more complex hypotheses, so this prior prefers simpler hypothesis ($\mu = 0$)
  - It is typical to have no prior on $w_0$
Logistic Regression: MAP

\[
\log \mathbb{P}(D | \mathbf{w}) + \log \mathbb{P}(\mathbf{w})
\]

\[
= \arg\max_{\mathbf{w}} \sum_{j=1,...,N} \log p(y^j | x^j, \mathbf{w}) + \log \prod_{i=1,...,d} N(w_i; 0, \sigma^2)
\]

\[
= \arg\max_{\mathbf{w}} \sum_{j} \log p(y^j | x^j, \mathbf{w}) + \sum_{i} \log\left( \frac{1}{\sqrt{2\pi}\sigma} \exp \left( \frac{-w_i^2}{2\sigma^2} \right) \right)
\]

\[
= \arg\max_{\mathbf{w}} \sum_{j} \log p(y^j | x^j, \mathbf{w}) + \sum_{i} \frac{-w_i^2}{2\sigma^2}
\]

\[
= \arg\max_{\mathbf{w}} \sum_{j} \log p(y^j | x^j, \mathbf{w}) - \frac{\lambda}{2} \sum_{i} w_i^2 \quad \text{Regularization}
\]

Old delta:

\[
\nabla L(\mathbf{w}) = \sum_{i=1}^{N} (y^i - \hat{y}^i)x^i
\]

\[
\nabla L(\mathbf{w}) = \sum_{i=1}^{N} (y^i - \hat{y}^i)x^i - \lambda \mathbf{w}
\]
Impact of $\lambda$

- $\lambda$ is inversely proportional to the variance of our prior belief $\lambda = \frac{1}{\sigma^2}$

- Use cross-validation to choose $\lambda$
Summary of Logistic Regression

- Discriminative classifier
- Learns conditional probability distribution
  - $P(y \mid x)$ defined by a logistic function
  - Produces a linear decision boundary
  - Nonlinear classifier can be achieved by using basis functions
- Maximum likelihood estimation
  - Gradient ascent bears strong similarity with perceptron
  - Unstable for linearly separable case, should use with regularization term to avoid this issue
  - Easily extended to multi-class problem using the soft-max function
- Maximum posterior estimation (MAP)
  - Using Gaussian prior on the weights, we arrive at L-2 regularization
  - Controls overfitting by adjusting $\lambda$