**Overview**

- **Classification**: predicting a discrete-valued target
  - **Binary classification**: predicting a binary-valued target
  - **Multiclass classification**: predicting a discrete (> 2)-valued target

- **Examples of binary classification**
  - predict whether a patient has a disease, given the presence or absence of various symptoms
  - classify e-mails as spam or non-spam
  - predict whether a financial transaction is fraudulent
Overview

Binary linear classification

- **classification**: given a $D$-dimensional input $x \in \mathbb{R}^D$ predict a discrete-valued target
- **binary**: predict a binary target $t \in \{0, 1\}$
  - Training examples with $t = 1$ are called positive examples, and training examples with $t = 0$ are called negative examples. Sorry.
  - $t \in \{0, 1\}$ or $t \in \{-1, +1\}$ is for computational convenience.
- **linear**: model prediction $y$ is a linear function of $x$, followed by a threshold $r$:

$$z = w^\top x + b$$

$$y = \begin{cases} 
1 & \text{if } z \geq r \\
0 & \text{if } z < r 
\end{cases}$$
Some Simplifications

Eliminating the threshold

- We can assume without loss of generality (WLOG) that the threshold \( r = 0 \):

\[
\mathbf{w}^\top \mathbf{x} + b \geq r \iff \mathbf{w}^\top \mathbf{x} + (b - r) \geq 0. \\
\overset{\Delta}{=} w_0
\]

Eliminating the bias

- Add a dummy feature \( x_0 \) which always takes the value 1. The weight \( w_0 = b \) is equivalent to a bias (same as linear regression)

Simplified model

- Receive input \( \mathbf{x} \in \mathbb{R}^{D+1} \) with \( x_0 = 1 \):

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

\[
y = \begin{cases} 
1 & \text{if } z \geq 0 \\
0 & \text{if } z < 0
\end{cases}
\]
Examples

- Let’s consider some simple examples to examine the properties of our model
- Let’s focus on minimizing the training set error, and forget about whether our model will generalize to a test set.
Examples

\begin{center}
\begin{tabular}{ccc}
\hline
$x_0$ & $x_1$ & t \\
\hline
1 & 0 & 1 \\
1 & 1 & 0 \\
\hline
\end{tabular}
\end{center}

- Suppose this is our training set, with the dummy feature $x_0$ included.

- Which conditions on $w_0, w_1$ guarantee perfect classification?
  - When $x_1 = 0$, need: $z = w_0 x_0 + w_1 x_1 \geq 0 \iff w_0 \geq 0$
  - When $x_1 = 1$, need: $z = w_0 x_0 + w_1 x_1 < 0 \iff w_0 + w_1 < 0$

- Example solution: $w_0 = 1, w_1 = -2$

- Is this the only solution?
Examples

AND

\[ z = w_0 x_0 + w_1 x_1 + w_2 x_2 \]

<table>
<thead>
<tr>
<th>( x_0 )</th>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

need: \( w_0 < 0 \)

need: \( w_0 + w_2 < 0 \)

need: \( w_0 + w_1 < 0 \)

need: \( w_0 + w_1 + w_2 \geq 0 \)

Example solution: \( w_0 = -1.5, w_1 = 1, w_2 = 1 \)
Input Space, or Data Space for NOT example

Training examples are points

Weights (hypotheses) \( \mathbf{w} \) can be represented by half-spaces

\[
H_+ = \{ \mathbf{x} : \mathbf{w}^\top \mathbf{x} \geq 0 \}, \quad H_- = \{ \mathbf{x} : \mathbf{w}^\top \mathbf{x} < 0 \}
\]

- The boundaries of these half-spaces pass through the origin (why?)

The boundary is the decision boundary: \( \{ \mathbf{x} : \mathbf{w}^\top \mathbf{x} = 0 \} \)

- In 2-D, it’s a line, but in high dimensions it is a hyperplane

If the training examples can be perfectly separated by a linear decision rule, we say data is linearly separable.

\[
\begin{array}{ccc}
  x_0 & x_1 & t \\
  1 & 0 & 1 \\
  1 & 1 & 0 \\
\end{array}
\]
The Geometric Picture

Weight Space

- Weights (hypotheses) $\mathbf{w}$ are points.
- Each training example $\mathbf{x}$ specifies a half-space $\mathbf{w}$ must lie in to be correctly classified: $\mathbf{w}^\top \mathbf{x} \geq 0$ if $t = 1$.
- For NOT example:
  - $x_0 = 1, x_1 = 0, t = 1 \implies (w_0, w_1) \in \{\mathbf{w} : w_0 \geq 0\}$
  - $x_0 = 1, x_1 = 1, t = 0 \implies (w_0, w_1) \in \{\mathbf{w} : w_0 + w_1 < 0\}$
- The region satisfying all the constraints is the feasible region; if this region is nonempty, the problem is feasible, otw it is infeasible.
The Geometric Picture

- The **AND** example requires three dimensions, including the dummy one.
- To visualize data space and weight space for a 3-D example, we can look at a 2-D slice.
- The visualizations are similar.
  - Feasible set will always have a corner at the origin.
The Geometric Picture

Visualizations of the AND example

Data Space

- Slice for \( x_0 = 1 \) and
- example sol: \( w_0 = -1.5, w_1 = 1, w_2 = 1 \)
- decision boundary:

\[
 w_0 x_0 + w_1 x_1 + w_2 x_2 = 0 \\
\implies -1.5 + x_1 + x_2 = 0
\]

Weight Space

- Slice for \( w_0 = -1.5 \) for the constraints
- \( w_0 < 0 \)
- \( w_0 + w_2 < 0 \)
- \( w_0 + w_1 < 0 \)
- \( w_0 + w_1 + w_2 \geq 0 \)
Summary — Binary Linear Classifiers

- **Summary**: Targets $t \in \{0, 1\}$, inputs $x \in \mathbb{R}^{D+1}$ with $x_0 = 1$, and model is defined by weights $w$ and

  $$z = w^\top x$$

  $$y = \begin{cases} 
  1 & \text{if } z \geq 0 \\
  0 & \text{if } z < 0
  \end{cases}$$

- How can we find good values for $w$?
- If training set is linearly separable, we could solve for $w$ using linear programming
  - We could also apply an iterative procedure known as the *perceptron algorithm* (but this is primarily of historical interest).
- If it’s not linearly separable, the problem is harder
  - Data is almost never linearly separable in real life.
Towards Logistic Regression
Loss Functions

- Instead: define loss function then try to minimize the resulting cost function
  
  - Recall: cost is loss averaged (or summed) over the training set

- Seemingly obvious loss function: 0-1 loss

\[
\mathcal{L}_{0-1}(y, t) = \begin{cases} 
0 & \text{if } y = t \\
1 & \text{if } y \neq t 
\end{cases} = \mathbb{I}[y \neq t]
\]
Usually, the cost $\mathcal{J}$ is the averaged loss over training examples; for 0-1 loss, this is the misclassification rate:

$$
\mathcal{J} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}[y^{(i)} \neq t^{(i)}]
$$
Problem: how to optimize? In general, a hard problem (can be NP-hard)
This is due to the step function (0-1 loss) not being nice (continuous/smooth/convex etc)
Attempt 1: 0-1 loss

- Minimum of a function will be at its critical points.
- Let’s try to find the critical point of 0-1 loss
- Chain rule:
\[ \frac{\partial L_{0-1}}{\partial w_j} = \frac{\partial L_{0-1}}{\partial z} \frac{\partial z}{\partial w_j} \]

- But \( \frac{\partial L_{0-1}}{\partial z} \) is zero everywhere it’s defined!

\[ \frac{\partial L_{0-1}}{\partial w_j} = 0 \] means that changing the weights by a very small amount probably has no effect on the loss.

- Almost any point has 0 gradient!
Sometimes we can replace the loss function we care about with one which is easier to optimize. This is known as relaxation with a smooth surrogate loss function.

One problem with $L_{0-1}$: defined in terms of final prediction, which inherently involves a discontinuity.

Instead, define loss in terms of $\mathbf{w}^T \mathbf{x}$ directly

- Redo notation for convenience: $z = \mathbf{w}^T \mathbf{x}$
Attempt 2: Linear Regression

- We already know how to fit a linear regression model. Can we use this instead?

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

\[ \mathcal{L}_{SE}(z, t) = \frac{1}{2}(z - t)^2 \]

- Doesn’t matter that the targets are actually binary. Treat them as continuous values.

- For this loss function, it makes sense to make final predictions by thresholding \( z \) at \( \frac{1}{2} \) (why?)
The problem:

- The loss function hates when you make correct predictions with high confidence!
- If $t = 1$, it’s more unhappy about $z = 10$ than $z = 0$. 

$t = \frac{1}{2}$
Attempt 3: Logistic Activation Function

- There’s obviously no reason to predict values outside $[0, 1]$. Let’s squash $y$ into this interval.

- The **logistic function** is a kind of **sigmoid**, or S-shaped function:

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

- $\sigma^{-1}(y) = \log(y/(1 - y))$ is called the **logit**.

- A linear model with a logistic nonlinearity is known as **log-linear**:

  \[
  z = \mathbf{w}^\top \mathbf{x} \quad \quad y = \sigma(z) \quad \quad \mathcal{L}_{SE}(y, t) = \frac{1}{2} (y - t)^2.
  \]

- Used in this way, $\sigma$ is called an **activation function**.
Attempt 3: Logistic Activation Function

The problem:
(plot of $L_{SE}$ as a function of $z$, assuming $t = 1$)

\[
\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z} \frac{\partial z}{\partial w_j}
\]

- For $z \ll 0$, we have $\sigma(z) \approx 0$.
- $\frac{\partial L}{\partial z} \approx 0$ (check!) $\implies$ $\frac{\partial L}{\partial w_j} \approx 0$ $\implies$ derivative w.r.t. $w_j$ is small
  $\implies$ $w_j$ is like a critical point
- If the prediction is really wrong, you should be far from a critical point (which is your candidate solution).
Logistic Regression

- Because $y \in [0, 1]$, we can interpret it as the estimated probability that $t = 1$. If $t = 0$, then we want to heavily penalize $y \approx 1$.
- The pundits who were 99% confident Clinton would win were much more wrong than the ones who were only 90% confident.
- Cross-entropy loss (aka log loss) captures this intuition:

$$
\mathcal{L}_{CE}(y, t) = \begin{cases} 
- \log y & \text{if } t = 1 \\
- \log(1 - y) & \text{if } t = 0
\end{cases}
= -t \log y - (1 - t) \log(1 - y)
$$
Logistic Regression:

\[ z = \mathbf{w}^\top \mathbf{x} \]
\[ y = \sigma(z) = \frac{1}{1 + e^{-z}} \]
\[ \mathcal{L}_{CE} = -t \log y - (1 - t) \log(1 - y) \]

Plot is for target \( t = 1 \).
Logistic Regression — Numerical Instabilities

- If we implement logistic regression naively, we can end up with numerical instabilities.
- Consider: $t = 1$ but you’re really confident that $z \ll 0$.
- If $y$ is small enough, it may be numerically zero. This can cause very subtle and hard-to-find bugs.

\[
y = \sigma(z) \quad \Rightarrow \quad y \approx 0
\]

\[
\mathcal{L}_{CE} = -t \log y - (1 - t) \log(1 - y) \quad \Rightarrow \text{computes } \log 0
\]
Logistic Regression — Numerically Stable Version

- Instead, we combine the activation function and the loss into a single logistic-cross-entropy function.

\[ \mathcal{L}_{\text{LCE}}(z, t) = \mathcal{L}_{\text{CE}}(\sigma(z), t) = t \log(1 + e^{-z}) + (1 - t) \log(1 + e^{z}) \]

- Numerically stable computation:
  \[ E = t \ast \text{np.logaddexp}(0, -z) + (1-t) \ast \text{np.logaddexp}(0, z) \]
Logistic Regression

Comparison of loss functions: (for $t = 1$)
Gradient Descent for Logistic Regression

- How do we minimize the cost $J$ for logistic regression? No direct solution.
  - Taking derivatives of $J$ w.r.t. $w$ and setting them to 0 doesn’t have an explicit solution.

- However, the logistic loss is a convex function in $w$, so let’s consider the gradient descent method from last lecture.
  - Recall: we initialize the weights to something reasonable and repeatedly adjust them in the direction of steepest descent.
  - A standard initialization is $w = 0$. (why?)
Gradient of Logistic Loss

Back to logistic regression:

\[ \mathcal{L}_{CE}(y, t) = -t \log(y) - (1-t) \log(1-y) \]

\[ y = \frac{1}{1 + e^{-z}} \quad \text{and} \quad z = \mathbf{w}^\top \mathbf{x} \]

Therefore

\[ \frac{\partial \mathcal{L}_{CE}}{\partial w_j} = \frac{\partial \mathcal{L}_{CE}}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_j} = \left( -\frac{t}{y} + \frac{1-t}{1-y} \right) \cdot y(1-y) \cdot x_j \]

\[ = (y-t)x_j \]

(verify this)

Gradient descent (coordinatewise) update to find the weights of logistic regression:

\[ w_j \leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j} \]

\[ = w_j - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) x_j^{(i)} \]
Gradient Descent for Logistic Regression

Comparison of gradient descent updates:

- **Linear regression:**

  \[ w \leftarrow w - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) x^{(i)} \]

- **Logistic regression:**

  \[ w \leftarrow w - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)}) x^{(i)} \]

- Not a coincidence! These are both examples of generalized linear models. But we won’t go in further detail.

- Notice \( \frac{1}{N} \) in front of sums due to averaged losses. This is why you need smaller learning rate when cost is summed losses (\( \alpha' = \alpha / N \)).
Multiclass Classification and Softmax Regression
Overview

- **Classification**: predicting a discrete-valued target
  - Binary classification: predicting a binary-valued target
  - Multiclass classification: predicting a discrete (> 2)-valued target

- Examples of multi-class classification
  - predict the value of a handwritten digit
  - classify e-mails as spam, travel, work, personal
Multiclass Classification

- Classification tasks with more than two categories:
Multiclass Classification

- Targets form a discrete set \( \{1, \ldots, K\} \).
- It’s often more convenient to represent them as one-hot vectors, or a one-of-\(K\) encoding:

\[
t = (0, \ldots, 0, 1, 0, \ldots, 0) \in \mathbb{R}^K
\]

entry \(k\) is 1
Multiclass Linear Classification

- We can start with a linear function of the inputs.
- Now there are $D$ input dimensions and $K$ output dimensions, so we need $K \times D$ weights, which we arrange as a weight matrix $W$.
- Also, we have a $K$-dimensional vector $b$ of biases.
- A linear function of the inputs:

$$z_k = \sum_{j=1}^{D} w_{kj} x_j + b_k \quad \text{for} \quad k = 1, 2, ..., K$$

- We can eliminate the bias $b$ by taking $W \in \mathbb{R}^{K \times (D+1)}$ and adding a dummy variable $x_0 = 1$. So, vectorized:

$$z = Wx + b \quad \text{or with dummy} \quad x_0 = 1 \quad z = Wx$$
How can we turn this linear prediction into a one-hot prediction?

We can interpret the magnitude of $z_k$ as a measure of how much the model prefers $k$ as its prediction.

If we do this, we should set

\[ y_i = \begin{cases} 
1 & i = \arg \max_k z_k \\
0 & \text{otherwise}
\end{cases} \]

Exercise: how does the case of $K = 2$ relate to the prediction rule in binary linear classifiers?
Softmax Regression

- We need to soften our predictions for the sake of optimization.
- We want soft predictions that are like probabilities, i.e., \(0 \leq y_k \leq 1\) and \(\sum_k y_k = 1\).
- A natural activation function to use is the **softmax function**, a multivariable generalization of the logistic function:

\[
y_k = \text{softmax}(z_1, \ldots, z_K)_k = \frac{e^{z_k}}{\sum_{k'} e^{z_{k'}}}
\]

- Outputs can be interpreted as probabilities (positive and sum to 1)
- If \(z_k\) is much larger than the others, then \(\text{softmax}(z)_k \approx 1\) and it behaves like argmax.
- **Exercise**: how does the case of \(K = 2\) relate to the logistic function?
- The inputs \(z_k\) are called the **logits**.
Softmax Regression

- If a model outputs a vector of class probabilities, we can use cross-entropy as the loss function:

\[
\mathcal{L}_{\text{CE}}(y, t) = - \sum_{k=1}^{K} t_k \log y_k
\]

\[
= -t^\top (\log y),
\]

where the log is applied elementwise.

- Just like with logistic regression, we typically combine the softmax and cross-entropy into a softmax-cross-entropy function.
Softmax Regression

- **Softmax regression** (with dummy $x_0 = 1$):

  $$ z = Wx $$
  $$ y = \text{softmax}(z) $$
  $$ \mathcal{L}_{\text{CE}} = -t^\top (\log y) $$

- Gradient descent updates can be derived for each row of $W$:

  $$ \frac{\partial \mathcal{L}_{\text{CE}}}{\partial w_k} = \frac{\partial \mathcal{L}_{\text{CE}}}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_k} = (y_k - t_k) \cdot x $$

  $$ w_k \leftarrow w_k - \alpha \frac{1}{N} \sum_{i=1}^{N} (y_k^{(i)} - t_k^{(i)}) x^{(i)} $$

- Similar to linear/logistic reg (no coincidence) (verify the update)
Linear Classifiers vs. KNN
Linear Classifiers vs. KNN

Linear classifiers and KNN have very different decision boundaries:

Linear Classifier

K Nearest Neighbours
Linear Classifiers vs. KNN

Advantages of linear classifiers over KNN?

Advantages of KNN over linear classifiers?
A Few Basic Concepts

- A hypothesis is a function \( f : \mathcal{X} \rightarrow \mathcal{T} \) that we might use to make predictions (recall \( \mathcal{X} \) is the input space and \( \mathcal{T} \) is the target space).

- The hypothesis space \( \mathcal{H} \) for a particular machine learning model or algorithm is set of hypotheses that it can represent.
  - E.g., in linear regression, \( \mathcal{H} \) is the set of functions that are linear in the data features
  - The job of a machine learning algorithm is to find a good hypothesis \( f \in \mathcal{H} \)

- The members of \( \mathcal{H} \), together with an algorithm’s preference for some hypotheses of \( \mathcal{H} \) over others, determine an algorithm’s inductive bias.
  - Inductive biases can be understood as general natural patterns or domain knowledge that help our algorithms to generalize; E.g., linearity, continuity, simplicity (\( L_2 \) regularization) …
  - The so-called No Free Lunch (NFL) theorems assert that if datasets/problems were not naturally biased, no ML algorithm would be better than another
A Few Basic Concepts

- If an algorithm’s hypothesis space $\mathcal{H}$ can be defined using a finite set of parameters, denoted $\theta$, we say the algorithm is parametric.
  - In linear regression, $\theta = (w, b)$
  - Other examples: logistic regression, neural networks, $k$-means and Gaussian mixture models

- If the members of $\mathcal{H}$ are defined in terms of the data, we say that the algorithm is non-parametric.
  - In $k$-nearest neighbors, the learned hypothesis is defined in terms of the training data
  - Other examples: Gaussian processes, decision trees, support vector machines, kernel density estimation
  - These models can sometimes be understood as having an infinite number of parameters
Limits of Linear Classification

Some datasets are not linearly separable, e.g. **XOR**

Visually obvious, but how to show this?
Limits of Linear Classification

Showing that XOR is not linearly separable (proof by contradiction)

- If two points lie in a half-space, line segment connecting them also lie in the same halfspace.

- Suppose there were some feasible weights (hypothesis). If the positive examples are in the positive half-space, then the green line segment must be as well.

- Similarly, the red line segment must lie within the negative half-space.

- But the intersection can’t lie in both half-spaces. Contradiction!
Sometimes we can overcome this limitation using feature maps, just like for linear regression. E.g., for XOR:

\[ \psi(x) = \begin{pmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{pmatrix} \]

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( \psi_1(x) )</th>
<th>( \psi_2(x) )</th>
<th>( \psi_3(x) )</th>
<th>( t )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

This is linearly separable. (Try it!)
Next time...

Feature maps are hard to design well, so next time we’ll see how to learn nonlinear feature maps directly using neural networks...

...