Artificial Intelligence

Lecture 8: Logistic Regression & Classification

Prof. Antonio Khalil Moretti

Week 9

SCIS 432
Spelman College

Recap: What Linear Regression Does

Last Week

We learned to predict a **continuous value**:

$$\hat{y} = mx + b \quad \in (-\infty, +\infty)$$

Examples:

- Predict rent given square footage
- Predict disease progression from BMI
- Predict tomorrow's temperature

Evaluation: MSE, SSE, R^2

Today's Question

What if the target y is a **category**?

Binary classification: $y \in \{0, 1\}$

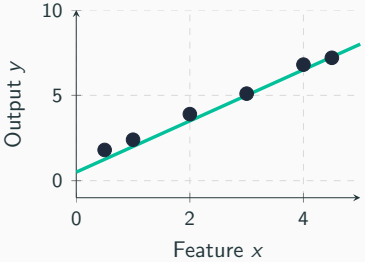
- Is this email **spam** or **not spam**?
- Is this tumor **malignant** or **benign**?
- Will this loan **default** or **not**?
- Is this transaction **fraudulent**?

Problem:

Linear regression predicts any real number. We need a number in $[0, 1]$ to represent a **probability**.

From Regression to Classification

Regression

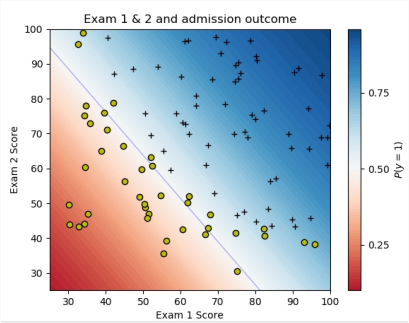


Output: any real number

$$\hat{y} \in (-\infty, +\infty)$$

$$\hat{y}_i = \beta_1 x_i + \beta_0$$

Binary Classification



Output: 0 or 1 (a label)

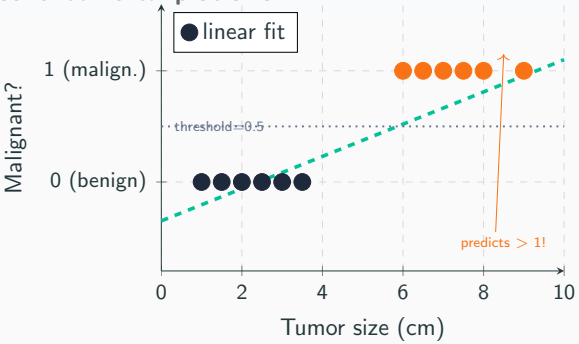
$$y \in \{0, 1\}$$

Core shift: Instead of predicting a value, we predict a **probability**

$P(y = 1 \mid x)$ — the probability that input x belongs to class 1.

Why Not Use Linear Regression for Classification?

Three fundamental problems



Problem 1 — Out-of-range outputs
Linear regression can predict values < 0 or > 1.
These cannot be probabilities!

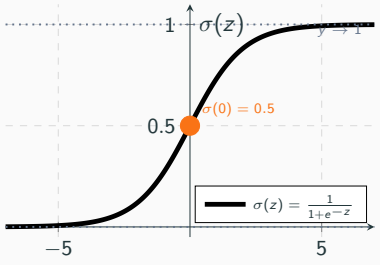
Problem 2 — Wrong loss function
MSE treats distances between classes as meaningful. Being "halfway between" 0 and 1 has no probabilistic interpretation.

Problem 3 — Sensitivity to outliers
A single extreme positive example drags the line up, shifting the decision boundary and misclassifying nearby negatives.

What we need:
A model that outputs a number in $[0, 1]$, interpretable as a probability.

The Sigmoid Function: A Squashing Function

Maps any real number into $(0, 1)$



Definition

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

Key properties:

- Output always in $(0, 1)$ ✓
- $\sigma(0) = 0.5$ (midpoint)
- $\sigma(z) \to 1$ as $z \to +\infty$
- $\sigma(z) \to 0$ as $z \to -\infty$
- **Monotonically increasing**
- Smooth and differentiable everywhere ✓

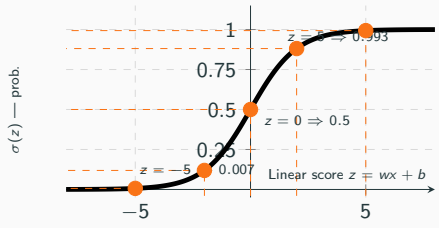
Usage in logistic regression:

$$P(y = 1 \mid x) = \sigma(\mathbf{w}^T \mathbf{x} + b)$$

We pass the linear output $z = \mathbf{w}^T \mathbf{x} + b$ through σ to get a valid probability.

The Sigmoid as a Squashing Function

Demonstrating the transformation from $\mathbb{R}$ to $(0, 1)$



Squashing in Action

z	e^{-z}	$\sigma(z)$
-5	148.4	0.007
-2	7.39	0.119
0	1	0.500
2	0.135	0.881
5	0.007	0.993

Interpretation:

- $z \ll 0$: strong evidence for class 0
- $z = 0$: maximum uncertainty
- $z \gg 0$: strong evidence for class 1

Key insight: $\sigma(z) \in (0, 1)$ always — the entire real line is compressed into one bounded interval.

The Logistic Regression Model

Full model for binary classification:

Step 1: Compute a linear score

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b = \mathbf{w}^T \mathbf{x} + b$$

Step 2: Squash through sigmoid

$$\hat{p} = P(y = 1 \mid \mathbf{x}) = \sigma(z) = \frac{1}{1+e^{-z}}$$

Step 3: Make a prediction

$$\hat{y} = \begin{cases} 1 & \text{if } \hat{p} \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

Why threshold at 0.5**?**
$\hat{p} \geq 0.5 \Leftrightarrow z \geq 0$

$$z = \mathbf{w}^T \mathbf{x} + b \geq 0$$

This is the **decision boundary** — the set of inputs where the model is maximally uncertain.

Parameters to learn:

- Weight vector $\mathbf{w} \in \mathbb{R}^d$
- Bias scalar $b \in \mathbb{R}$

Note the **decision boundary** $\mathbf{w}^T \mathbf{x} + b = 0$ is a *hyperplane* — it is linear even though the model is called "logistic".

Simplifying Notation: Absorbing the Bias into w

A standard trick that makes the algebra cleaner

Original model:

$$z = w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b$$

We have d weights *plus* a separate bias b — two kinds of parameters.

The trick: Prepend a constant feature $x_0 = 1$ to every data point:

$$\tilde{\mathbf{x}} = \begin{pmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}, \qquad \tilde{\mathbf{w}} = \begin{pmatrix} b \\ w_1 \\ w_2 \\ \vdots \\ w_d \end{pmatrix}$$

Then: $z = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}} = b \cdot 1 + w_1 x_1 + \cdots + w_d x_d$ ✓

Result:

$$z = \mathbf{w}^T \mathbf{x} \quad \text{(bias absorbed)}$$

Now we write simply $\mathbf{w}^T \mathbf{x}$ with the understanding that:

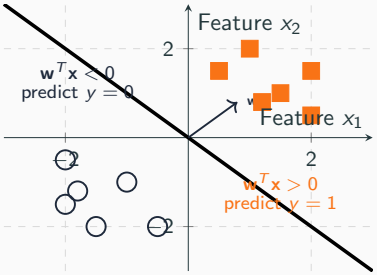
- The first component of $\mathbf{x}$ is always 1
- The first component of $\mathbf{w}$ is the bias b
- The design matrix $\mathbf{X}$ has a leading column of ones

Why bother?

- **Cleaner notation:** one vector $\mathbf{w}$ instead of $(\mathbf{w}, b)$
- **Unified gradient:** $\nabla_{\mathbf{w}} J = \frac{1}{n} \mathbf{X}^T (\hat{\mathbf{p}} - \mathbf{y})$ covers the bias update automatically

Visualizing the Linear Decision Boundary

The boundary $\mathbf{w}^T\mathbf{x} = 0$ separates the two classes



Key geometry

The decision boundary $\mathbf{w}^T\mathbf{x} = 0$ is a **line** in 2D, a **plane** in 3D, and a **hyperplane** in d dimensions.

On either side:

- $\mathbf{w}^T\mathbf{x} > 0 \Rightarrow \hat{p} > 0.5 \Rightarrow$ predict class 1
- $\mathbf{w}^T\mathbf{x} < 0 \Rightarrow \hat{p} < 0.5 \Rightarrow$ predict class 0

The weight vector w:
$\mathbf{w}$ is perpendicular (normal) to the decision boundary. Larger $\|\mathbf{w}\|$ means a steeper sigmoid — more confident predictions.

Limitation: Logistic regression can only draw *straight* decision boundaries. Data that isn't

Calculus Review: Product, Quotient, and Chain Rules

Tools we need for the sigmoid derivative

Product Rule

If $h(x) = f(x) \cdot g(x)$, then:

$$h'(x) = f'(x)\,g(x) + f(x)\,g'(x)$$

Example: $h(x) = x^2 e^x$

$$h'(x) = 2x\,e^x + x^2\,e^x = e^x(2x + x^2)$$

Quotient Rule

If $h(x) = \dfrac{f(x)}{g(x)}$, then:

$$h'(x) = \frac{f'(x)\,g(x) - f(x)\,g'(x)}{[g(x)]^2}$$

Example: $h(x) = \dfrac{1}{1 + e^{-x}}$

Chain Rule

If $h(x) = f(g(x))$, then:

$$h'(x) = f'(g(x)) \cdot g'(x)$$

Or equivalently, with $u = g(x)$:

$$\frac{dh}{dx} = \frac{df}{du} \cdot \frac{du}{dx}$$

Example: $h(x) = e^{-x^2}$, let $u = -x^2$

$$h'(x) = e^u \cdot (-2x) = -2x\,e^{-x^2}$$

Useful derivatives to know

$$\frac{d}{dx}e^u = e^u \frac{du}{dx} \qquad \frac{d}{dx}\log(u) = \frac{1}{u}\frac{du}{dx}$$

$$\frac{d}{dx}u^n = nu^{n-1}\frac{du}{dx} \qquad \frac{d}{dx}c = 0$$

Derivative of the Sigmoid Function

A beautiful self-referential result

Derivation

Let $\sigma(z) = (1 + e^{-z})^{-1}$. By the chain rule:

$$\frac{d\sigma}{dz} = -(1 + e^{-z})^{-2} \cdot (-e^{-z}) = \frac{e^{-z}}{(1 + e^{-z})^2}$$

Now rewrite cleverly:

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

Since $\frac{e^{-z}}{1 + e^{-z}} = 1 - \frac{1}{1 + e^{-z}} = 1 - \sigma(z)$, we get:

$$\frac{d\sigma}{dz} = \sigma(z)\big(1 - \sigma(z)\big)$$

What this means

The derivative can be written entirely in terms of the *output* $\sigma(z)$ — no need to recompute e^{-z} during backpropagation!

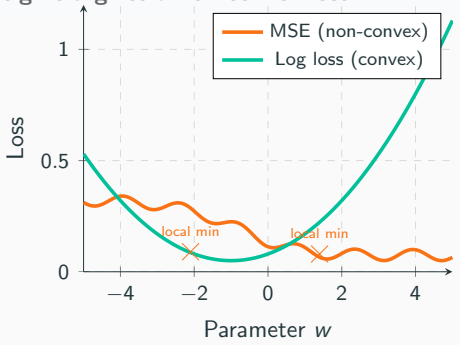
Numerical check:

z	$\sigma(z)$	$\sigma'(z)$
-3	0.047	0.045
0	0.500	0.250
3	0.953	0.045

Observation: The gradient is **largest at** $z = 0$ and shrinks toward zero at the extremes. This is the "**vanishing gradient**" problem for deep networks.

Why Not Use MSE for Classification?

MSE + sigmoid gives a non-convex loss



The problem with MSE here:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}\big(y_i - \sigma(z_i)\big)^2$$

When combined with the sigmoid, this produces a **non-convex** loss surface with many local minima. Gradient descent may get stuck.

We need a loss that:

- Is **convex** in the parameters
- Penalizes **confident wrong answers** very heavily
- Has a **probabilistic interpretation**

Solution: Binary cross-entropy loss, derived from Maximum Likelihood Estimation.

Connection to Maximum Likelihood Estimation

The probabilistic foundation of logistic regression

Probabilistic model

Each label $y_i \in \{0, 1\}$ is drawn from a **Bernoulli distribution**:

$$P(y_i \mid \mathbf{x}_i) = \hat{p}_i^{y_i}(1 - \hat{p}_i)^{1-y_i}$$

where $\hat{p}_i = \sigma(\mathbf{w}^T \mathbf{x}_i + b)$.

Check: if $y_i = 1$, this gives $\hat{p}_i$. If $y_i = 0$, it gives $1 - \hat{p}_i$. ✓

Likelihood of all n observations:
Assuming independence:

$$\mathcal{L}(\mathbf{w}, b) = \prod_{i=1}^{n} \hat{p}_i^{y_i}(1 - \hat{p}_i)^{1-y_i}$$

Log-Likelihood

Products are hard to differentiate. Take the log (monotone transformation, same maximizer):

$$\ell(\mathbf{w}, b) = \sum_{i=1}^{n} [y_i \log \hat{p}_i + (1 - y_i) \log(1 - \hat{p}_i)]$$

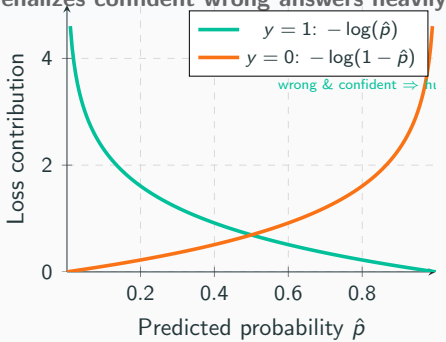
Maximize log-likelihood ≡ **Minimize** negative log-likelihood:

$$J = -\frac{1}{n} \sum_{i=1}^{n} \big[y_i \log \hat{p}_i + (1 - y_i) \log(1 - \hat{p}_i)\big]$$

This is the **binary cross-entropy loss**.

Understanding the Cross-Entropy Loss

Why it penalizes confident wrong answers heavily



Two cases:

When $y_i = 1$ (true positive):

$$\text{loss} = -\log(\hat{p}_i)$$

- $\hat{p} \approx 1$: loss ≈ 0 ✓
- $\hat{p} \approx 0$: loss $\to \infty$ (bad!)

When $y_i = 0$ (true negative):

$$\text{loss} = -\log(1 - \hat{p}_i)$$

- $\hat{p} \approx 0$: loss ≈ 0 ✓
- $\hat{p} \approx 1$: loss $\to \infty$ (bad!)

Key property: Being confidently wrong costs

Cross-Entropy Loss in Linear Algebra Form

Setup: Design matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, label vector $\mathbf{y} \in \{0,1\}^n$, parameters $\mathbf{w} \in \mathbb{R}^d$.

Linear scores (vectorized):

$$\mathbf{z} = \mathbf{X}\mathbf{w} \in \mathbb{R}^n \quad \text{(one score per data point)}$$

Predicted probabilities (element-wise sigmoid):

$$\hat{\mathbf{p}} = \sigma(\mathbf{z}) = \sigma(\mathbf{X}\mathbf{w}) \in (0,1)^n$$

Binary cross-entropy loss (vectorized):

$$J(\mathbf{w}) = -\frac{1}{n}\Big[\mathbf{y}^T \log(\hat{\mathbf{p}}) + (1-\mathbf{y})^T \log(1-\hat{\mathbf{p}})\Big]$$

where $\log(\hat{\mathbf{p}})$ and $\log(1-\hat{\mathbf{p}})$ are computed element-wise.

The two dot products replace the sum over i: $\mathbf{y}^T \log(\hat{\mathbf{p}}) = \sum_i y_i \log \hat{p}_i$.

Gradient Descent for Logistic Regression

Computing the gradient

We want $\nabla_{\mathbf{w}} J$. By chain rule:

$$\frac{\partial J}{\partial \mathbf{w}} = \frac{\partial J}{\partial \hat{p}_i} \cdot \frac{\partial \hat{p}_i}{\partial z_i} \cdot \frac{\partial z_i}{\partial \mathbf{w}}$$

Term 1: Derivative of loss w.r.t. $\hat{p}_i$:

$$\frac{\partial}{\partial \hat{p}_i} \left[-y_i \log \hat{p}_i - (1 - y_i) \log(1 - \hat{p}_i) \right] = \frac{\hat{p}_i - y_i}{\hat{p}_i(1 - \hat{p}_i)}$$

Term 2: Derivative of sigmoid:

$$\frac{d\hat{p}_i}{dz_i} = \sigma'(z_i) = \hat{p}_i(1 - \hat{p}_i)$$

Magical cancellation!

$$\text{Term 1} \times \text{Term 2} = \frac{\hat{p}_i - y_i}{\cancel{\hat{p}_i(1 - \hat{p}_i)}} \cdot \cancel{\hat{p}_i(1 - \hat{p}_i)} = \hat{p}_i - y_i$$

Final gradient and update

Using $\frac{\partial z_i}{\partial \mathbf{w}} = \mathbf{x}_i$ and summing over all points:

$$\nabla_{\mathbf{w}} J = \frac{1}{n} \mathbf{X}^T (\hat{\mathbf{p}} - \mathbf{y})$$

Gradient descent update:

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \cdot \frac{1}{n} \mathbf{X}^T (\hat{\mathbf{p}} - \mathbf{y})$$

Notation note:

We write $\hat{\mathbf{p}}$ (not $\hat{\mathbf{y}}$) to emphasize that the model outputs *probabilities* in $(0, 1)$, not arbitrary real values. The hard prediction $\hat{y} \in \{0, 1\}$ is obtained by thresholding $\hat{p}$.

The gradient has the same *form* as linear regression — residuals $(\hat{\mathbf{p}} - \mathbf{y})$ times features $\mathbf{X}^T$ — but $\hat{\mathbf{p}} = \sigma(\mathbf{X}\mathbf{w})$ instead of $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$.

The Full Gradient Descent Algorithm

Algorithm

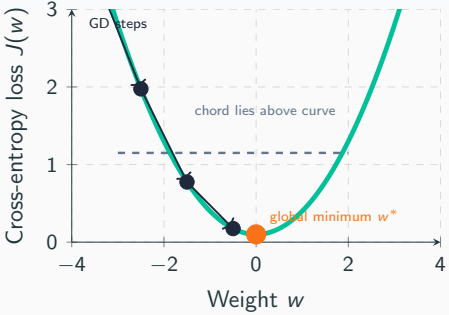
1. Initialize $\mathbf{w} = \mathbf{0}$ (or small random values)
2. Repeat until convergence:
   (a) Compute linear scores:
   $\mathbf{z} = \mathbf{X}\mathbf{w}$
   (b) Compute probabilities:
   $\hat{\mathbf{p}} = \sigma(\mathbf{z})$
   (c) Compute the loss:
   $J = -\frac{1}{n}\left[\mathbf{y}^T \log \hat{\mathbf{p}} + (1-\mathbf{y})^T \log(1-\hat{\mathbf{p}})\right]$
   (d) Compute gradient:
   $\mathbf{g} = \frac{1}{n}\mathbf{X}^T(\hat{\mathbf{p}} - \mathbf{y})$
   (e) Update weights:
   $\mathbf{w} \leftarrow \mathbf{w} - \alpha\,\mathbf{g}$

Python Implementation

```
1 def sigmoid(z):  return 1 / (1 + np.exp(-z))
2 def fit_logistic(X, y, lr = 0.1, n_iter = 1000) : n, d =
  X.shapew = np.zeros(d)
3 for _ in range(n_iter) : z = X@wp_hat = sigmoid(z)grad =
  (1/n) * X.T@(p_hat - y)w- = lr * grad
4 return w
5 def predict(X, w, threshold=0.5):
  p_hat = sigmoid(X@w)return(p_hat >= threshold).astype(int)
```

Convexity of the Cross-Entropy Loss

Why gradient descent is guaranteed to find the global minimum



Definition of convexity

A function f is **convex** if for any two points a, b and $t \in [0, 1]$:

$$f\big(ta + (1-t)b\big) \leq t\, f(a) + (1-t)\, f(b)$$

The chord between any two points lies **above** *the curve.*

What convexity guarantees:

- Any local minimum **is** the global minimum
- Gradient descent converges to $\mathbf{w}^*$ (given appropriate learning rate)
- No "bad" local minima or saddle points

Why cross-entropy is convex:

The log function and the sigmoid's exponential form combine to cancel out

Linear vs. Logistic Regression: Side-by-Side

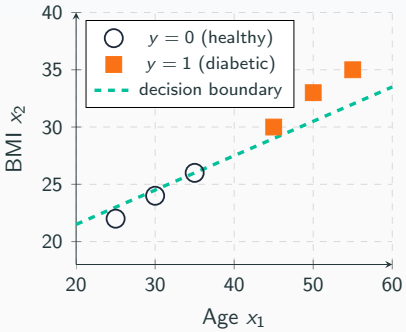
	Linear Regression	**Logistic Regression**
Task	Regression	Binary classification
Output	$\hat{y} = \mathbf{w}^T\mathbf{x} \in \mathbb{R}$	$\hat{p} = \sigma(\mathbf{w}^T\mathbf{x}) \in (0,1)$
Activation	None (identity)	Sigmoid $\sigma(z) = \frac{1}{1+e^{-z}}$
Loss	Mean Squared Error	Binary cross-entropy
Loss formula	$\frac{1}{n}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$	$-\frac{1}{n}[\mathbf{y}^T \log \hat{\mathbf{p}} + (1-\mathbf{y})^T \log(1-\hat{\mathbf{p}})]$
Gradient	$\frac{1}{n}\mathbf{X}^T(\hat{\mathbf{y}} - \mathbf{y})$	$\frac{1}{n}\mathbf{X}^T(\hat{\mathbf{p}} - \mathbf{y})$
Closed-form?	Yes — normal equation	No (must use gradient descent)
Convex?	Yes (MSE is convex)	Yes (cross-entropy is convex)
Decision boundary	N/A	Linear hyperplane $\mathbf{w}^T\mathbf{x} = 0$

Note: The gradient formulas are identical in structure — the only difference is what $\hat{\cdot}$ represents.

A Worked Example: Two-Feature Classification

Data: 6 patients, features x_1 (age), x_2 (BMI), label y (diabetic?).

x_1 (age)	x_2 (BMI)	y
25	22	0
30	24	0
35	26	0
45	30	1
50	33	1
55	35	1



After training, the decision boundary separates the two classes.

Model: $\hat{p} = \sigma(w_1 x_1 + w_2 x_2 + b)$

Forward pass (one iteration):

Start: $\mathbf{w} = [0, 0]$, $b = 0$.

Then $z_i = 0$ for all i, so $\hat{p}_i = 0.5$.

Loss $= -\frac{1}{6}[3\log(0.5) + 3\log(0.5)] = \log 2 \approx 0.693$.

Log-Odds and the Odds Ratio

Odds and Log-Odds

If $\hat{p} = P(y = 1 \mid \mathbf{x})$, the **odds** of class 1 are:

$$\text{odds} = \frac{\hat{p}}{1 - \hat{p}}$$

Odds of 2 means the outcome is *twice as likely* to be class 1 as class 0.

The **log-odds** (or **logit**) is:

$$\text{logit}(\hat{p}) = \log\left(\frac{\hat{p}}{1 - \hat{p}}\right)$$

The key identity:
Substituting $\hat{p} = \sigma(z) = \dfrac{1}{1 + e^{-z}}$ gives:

$$\log\left(\frac{\hat{p}}{1 - \hat{p}}\right) = \log(e^z) = z = \mathbf{w}^T\mathbf{x}$$

Interpretation

Logistic regression is a **linear model for the log-odds**:

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

What does each weight w_j mean?
Increasing x_j by 1 unit *adds* w_j to the log-odds — which *multiplies* the odds by e^{w_j}.

w_j	Effect on odds
0.5	$\times\, e^{0.5} \approx 1.65$ (65% increase)
0	No change
-0.7	$\times\, e^{-0.7} \approx 0.50$ (50% decrease)

This is why the name is "logistic" — it models the logit (log-odds).

Check Your Understanding: Log-Odds & Odds Ratio

Work through these before moving on

Question 1

A model outputs $\hat{p} = 0.8$ for a patient.

(a) What are the **odds** of the patient having the disease?

(b) What is the **log-odds**?

(c) If the model has a single feature x (age) with weight $w = 0.04$, what is the linear score z for a 50-year-old? Does your answer match $\hat{p}$?

Question 2

Suppose $\log\left(\frac{\hat{p}}{1-\hat{p}}\right) = -1$.

(a) Solve for $\hat{p}$.

(b) Is the model predicting class 0 or class 1?

(c) What does $\hat{p} < 0.5$ correspond to in terms

Question 3

A logistic regression model for loan default has learned:

$$z = 0.03\,x_1 - 0.5\,x_2 + 1.2\,x_3$$

where x_1 = age, x_2 = credit score (hundreds), x_3 = num. missed payments.

(a) Which feature is the strongest predictor of default?

(b) Increasing the credit score by 100 points multiplies the odds of default by $e^{-0.5}$. Is this an increase or a decrease? By roughly how much?

(c) Write out $P(\text{default} \mid \mathbf{x})$ as an explicit sigmoid expression.

Summary: What We Learned

Conceptual:

- Classification predicts a *category*, not a value
- Linear regression fails for classification — outputs are out of range and loss is non-convex
- The **sigmoid** squashes $\mathbb{R}$ into $(0, 1)$
- Logistic regression models the **probability** of class 1
- The decision boundary is a **linear hyperplane**

Mathematical:

- $\sigma(z) = \frac{1}{1+e^{-z}}$, $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Loss: $J = -\frac{1}{n}\left[\mathbf{y}^T \log \hat{\mathbf{p}} + (1 - \mathbf{y})^T \log(1 - \hat{\mathbf{p}})\right]$
- Gradient: $\nabla J = \frac{1}{n}\mathbf{X}^T(\hat{\mathbf{p}} - \mathbf{y})$

Key Insights:

- The sigmoid's derivative cancels with the loss's derivative — yielding a *clean* gradient
- Cross-entropy is **convex** $\Rightarrow$ gradient descent converges to the global minimum
- Same gradient structure as linear regression — only $\hat{y}$ is replaced by $\hat{p} = \sigma(\mathbf{Xw})$

Next Time:

- Evaluation metrics: accuracy, precision, recall, F1-score
- The confusion matrix
- Multi-class classification (softmax regression)
- Regularization for logistic regression (L_1, L_2)

Questions?

Next week:

Evaluation, Multi-class Classification, and Regularization

Recommended reading:

- Murphy, *Probabilistic Machine Learning: An Introduction*, Chapter 10
- Bishop, *Pattern Recognition and Machine Learning*, Chapter 4
- Hastie et al., *The Elements of Statistical Learning*, Chapter 4