KAUST Academy & Tech Camp AI Week

Presented By: Ali Alqutayfi & Hassain Alsayhah

Linear Regression

Logistic Regression

Neural Networks

Deep Learning

Artificial Intelligence and Machine Learning

Logistic Regression

Lecture 2: Outline

- Linear Regression (Quick Review)
- What is Classification?
- From Linear to Logistic Regression
- The Sigmoid Function
- Loss Function (Cross-Entropy)
- How Do We Minimize Cross-Entropy?
- Evaluating Classification
- Models Multiclass Classification

Linear Regression - Quick Review

What we learned:

- Linear regression finds the best line through data
- We use the equation: $y = mx + b$ (or $\hat{y} = \theta_0 + \theta_1 x$)
- We minimize Mean Squared Error (MSE)
- Goal: Predict continuous values (house prices, temperatures, etc.)
- **Problem:** What if we want to predict categories instead of numbers?

What is Classification?

Goal: Predict discrete categories/classes instead of continuous values

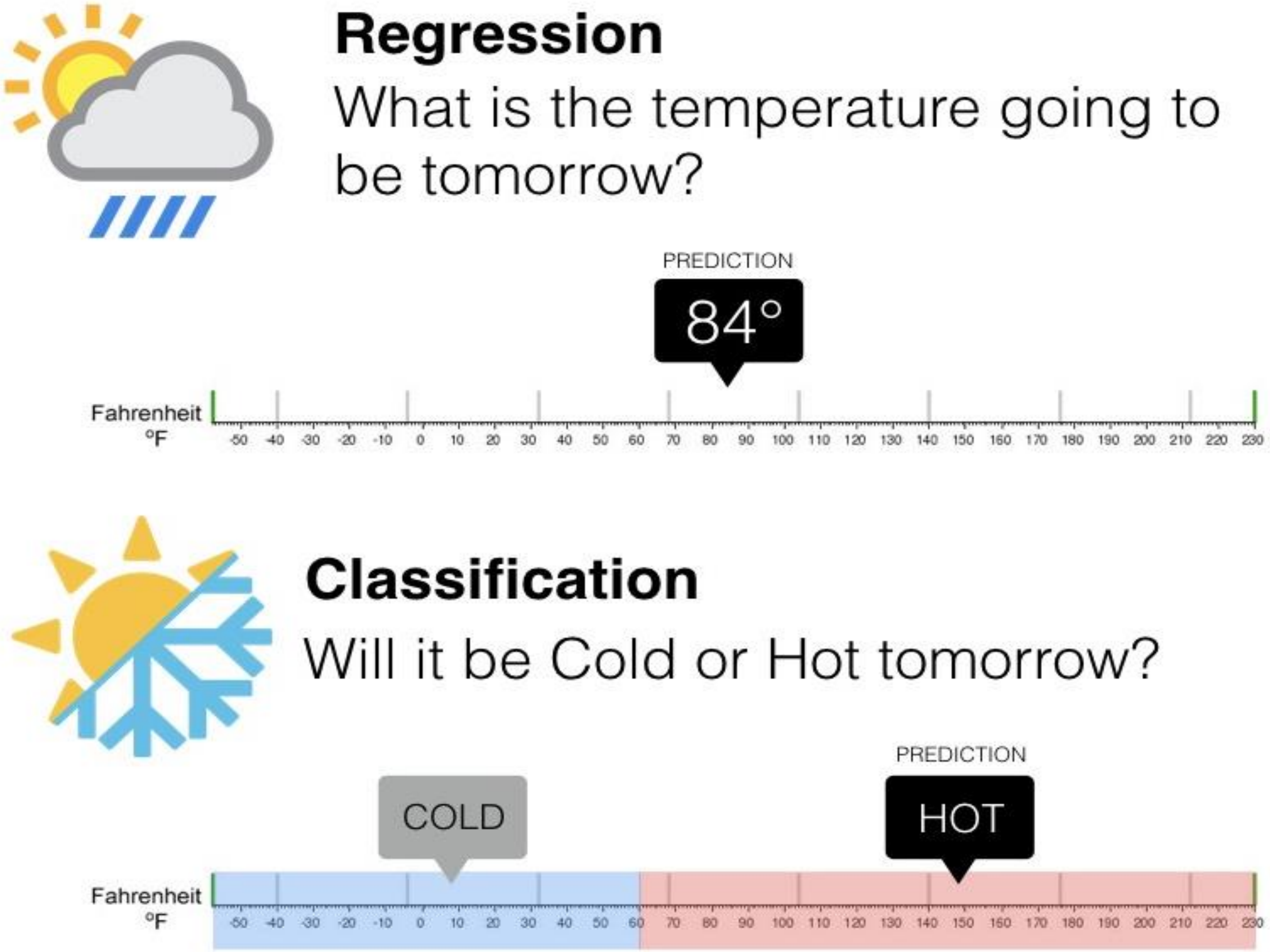
Examples:

- Email → Spam or Not Spam
- Image → Cat, Dog, or Bird
- Patient → Healthy or Sick
- Student → Pass or Fail

Key Difference:

- Regression: Predicts numbers (any value)
- Classification: Predicts categories (limited options)

Regression VS classification

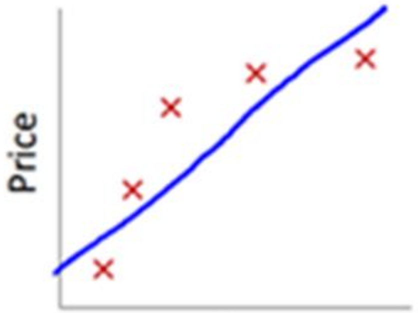


=> Continuous Values

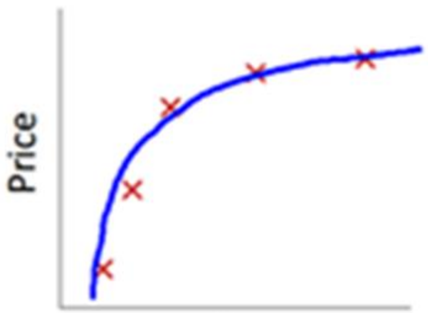
=> Discrete Values

Regression VS classification

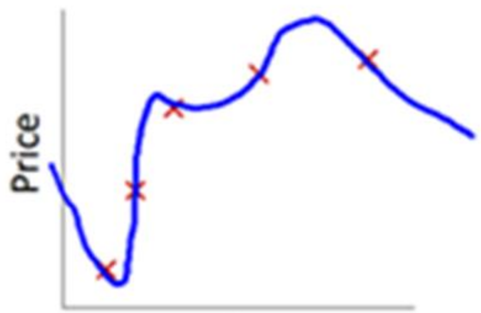
Regression:



Linear

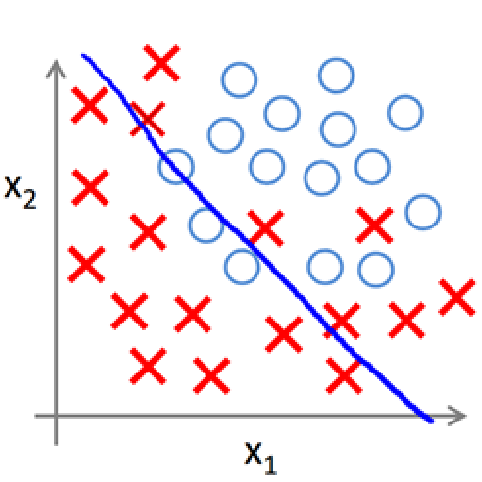


Polynomial

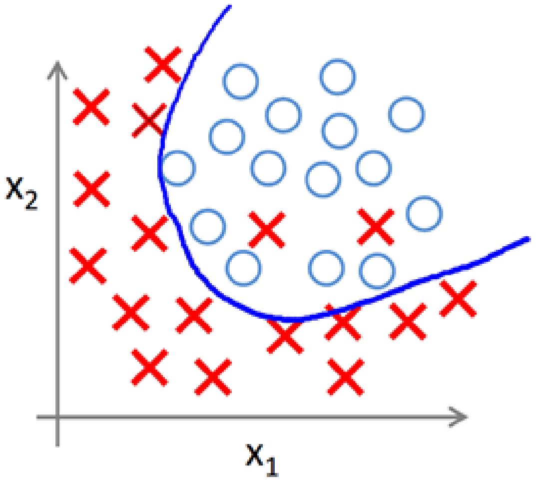


Very complex (overfitting)

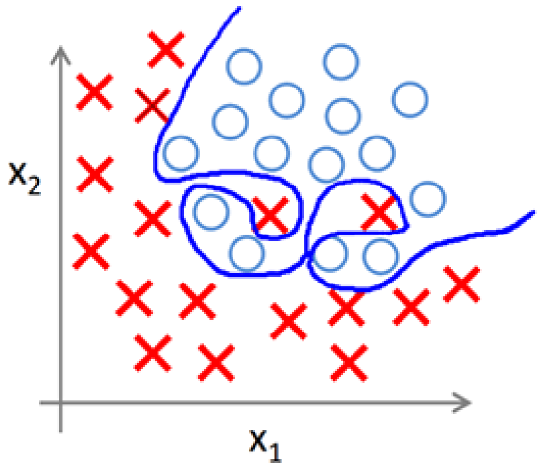
Classification:



Linear



Polynomial

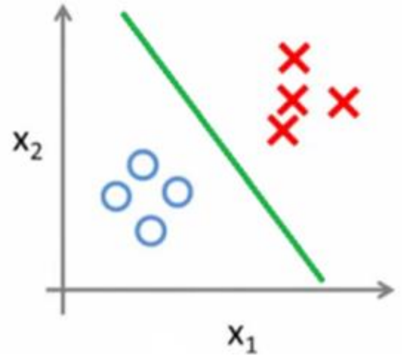


Very complex (overfitting)

Classification Types

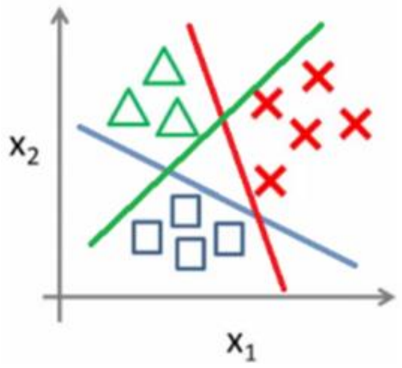
Binary Classification: Two possible outcomes

- Yes/No, Spam/Not Spam, Pass/Fail
- Uses one classifier



Multiclass Classification: More than two outcomes

- Cat/Dog/Bird, Grade A/B/C/D/F
- Uses multiple classifiers (one for each class)



The Problem with Linear Regression for Classification

What happens if we use linear regression for classification?

Example: Predicting Pass (1) or Fail (0) based on study hours

- Linear regression might predict 1.5 or -0.3
- But we only want 0 or 1!
- We need predictions between 0 and 1 (probabilities)

Solution: Transform linear regression output into probabilities

From Linear to Logistic Regression

Step 1: Start with linear regression

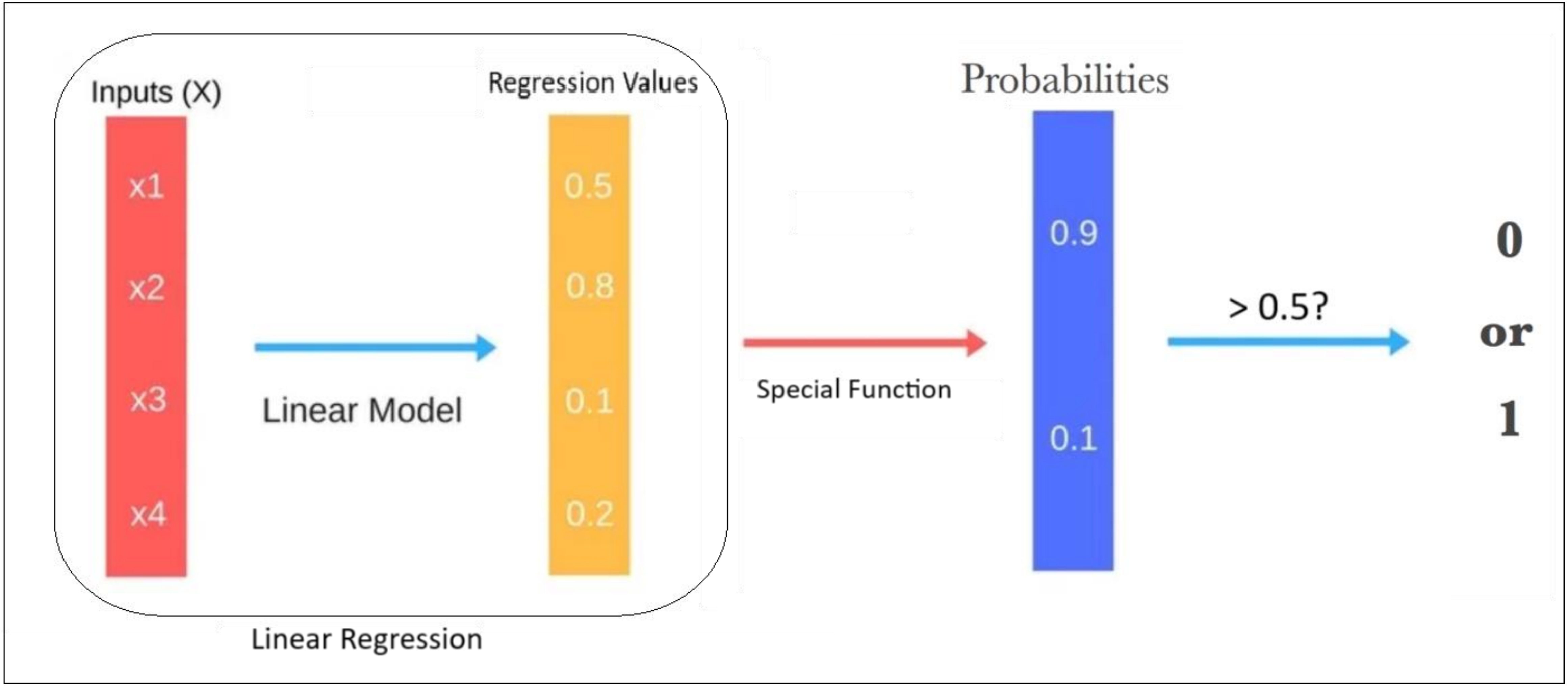
- $\hat{y} = \theta_0 + \theta_1 x$ (can output any value)

Step 2: Apply a special function to convert to probabilities

- Function should map any input to range [0,1]
- Function should be smooth (differentiable)

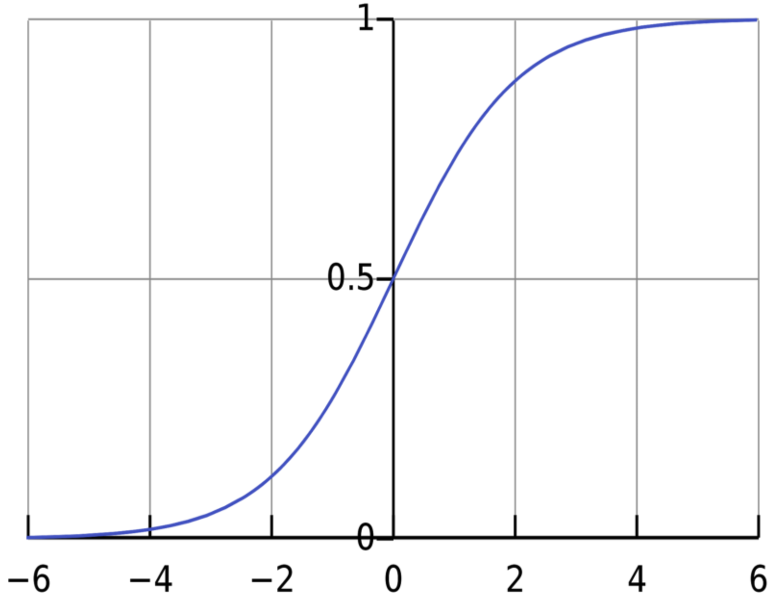
- **Step 3:** Use appropriate loss function for probabilities
  - MSE doesn't work well with probabilities
  - Need loss function designed for classification

From Linear to Logistic Regression



The Sigmoid Function

The magic function: $$\sigma(z) = \frac{1}{1 + exp(-z)}$$



$$\lim_{z \to -\infty} \sigma(z) = 0$$

$$\lim_{z \to \infty} \sigma(z) = 1$$

Properties:

- Input: Any real number (-∞ to +∞)
- Output: Always between 0 and 1
- Smooth S-shaped curve
- σ(0) = 0.5 (decision boundary)

Interpretation:

- Output close to 1 → Strong prediction for positive class
- Output close to 0 → Strong prediction for negative class
- Output around 0.5 → Uncertain prediction

But what if we have multiclass problem? 🤔

Multiclass Classification

Problem: What about more than 2 classes?

Solution: Extend logistic regression

- Binary: Cat vs Not-Cat
- Multiclass: Cat vs Dog vs Bird

Softmax Function: Generalization of sigmoid

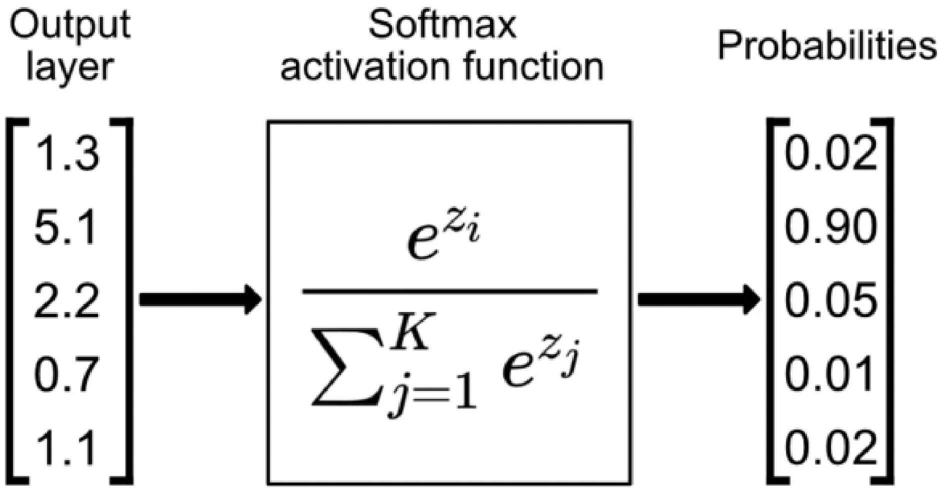
- Converts multiple outputs to probabilities
- All probabilities sum to 1
- Pick class with highest probability

Multiclass: Softmax Function

$$Softmax(z_i) = \frac{\exp(z_i)}{\sum_{j=1}^{n} \exp(z_j)}$$

Numerator: The exponential of the ith class $\exp(z_i)$.

Denominator: The sum of the exponentials of all classes $\sum_{j=1}^{n} \exp(z_j)$.



E.g. Divide the cat value by the cat, dog and deer values to get the cat probability.

Assuming you have K classes, its output would be the probability of the ith class. So, you will run it K times to get the probability of each class.

Logistic Regression Equation

Complete equation:

$$p = \sigma(\theta_0 + \theta_1 x) = \frac{1}{1 + e\hat{}(-(\theta_0 + \theta_1 x))}$$

Where:

- p = probability of positive class (between 0 and 1)
- θ_0 = intercept parameter
- θ_1 = slope parameter
- x = input feature

Making predictions:

- If $p > 0.5$ → Predict positive class (1)
- If $p \leq 0.5$ → Predict negative class (0)

Why Not Use MSE for Classification?

Problem with MSE for probabilities:

- MSE = (1/n) × Σ(actual - predicted)^2
- Doesn't penalize wrong confident predictions enough
- Example: Predicting 0.9 when actual is 0 should be heavily penalized

What we need:

- Loss function that works with probabilities
- Heavily penalizes confident wrong predictions
- Smooth and differentiable for optimization

Loss Function: Binary Cross-Entropy (LogLoss)

For one sample this can be represented mathematically by:

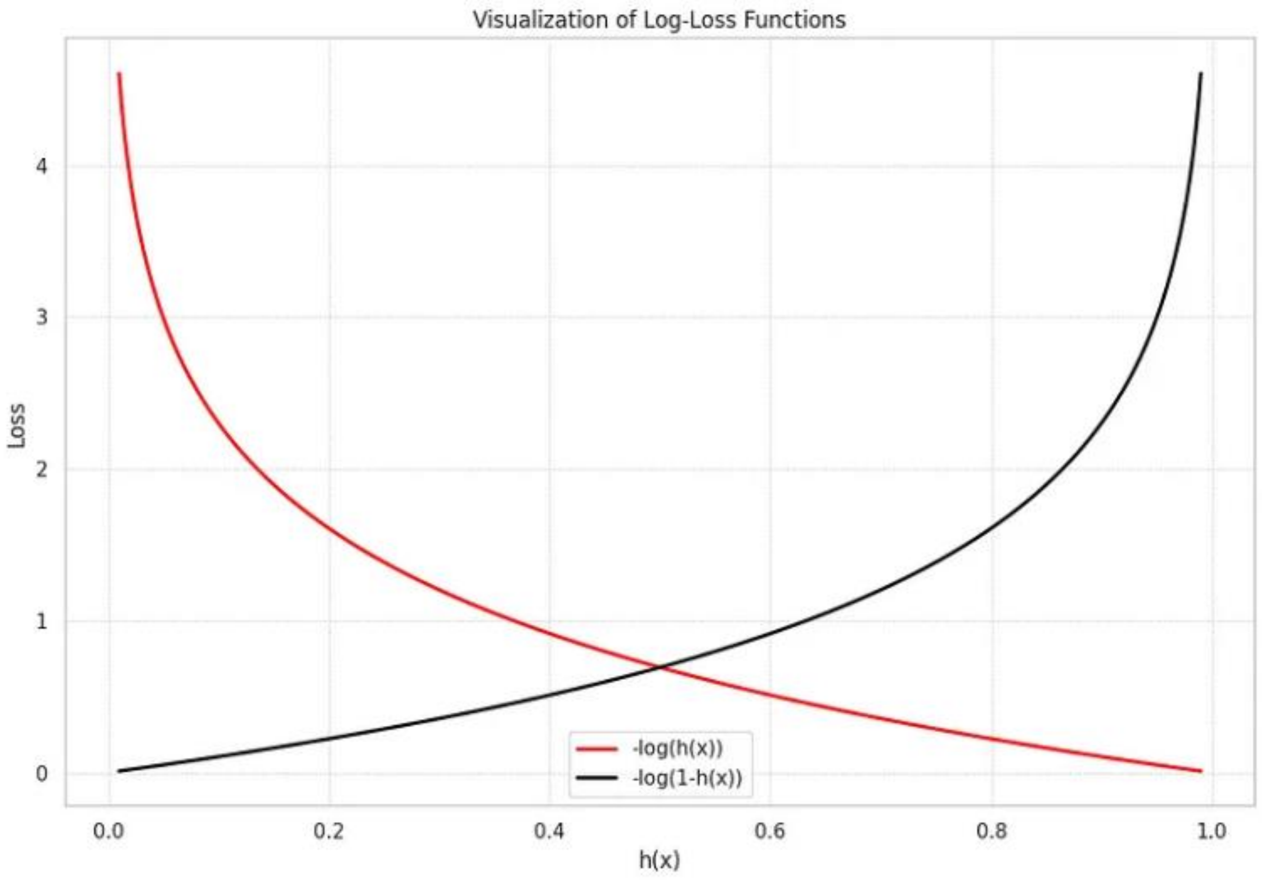
$$loss(y, p) = \begin{cases} -\log(p), & if\ y = 1 \\ -\log(1-p), & if\ y = 0 \end{cases}$$

Where:

- y = actual label (0 or 1)
- p = predicted probability

How it works:

- If y = 1 and p = 0.9 → Low loss (good!)
- If y = 1 and p = 0.1 → High loss (bad!)
- If y = 0 and p = 0.1 → Low loss (good!)
- If y = 0 and p = 0.9 → High loss (bad!)



Loss Function: Binary Cross-Entropy (LogLoss)

For one sample this can be represented mathematically by:

$$loss(y,p) = \begin{cases} -\log(p), & if\ y = 1 \\ -\log(1-p), & if\ y = 0 \end{cases}$$

Let’s rewrite it in one line:

$$loss(y,p) = -(y * \log(p) + (1-y) * \log(1-p))$$

But we have many samples, not only one, right?

Binary Cross-Entropy for All Samples (LogLoss)

For one sample this can be represented mathematically by:

$$loss(y, p) = \begin{cases} -\log(p), & if\ y = 1 \\ -\log(1-p), & if\ y = 0 \end{cases}$$

Let's rewrite it in one line:

$$loss(y, p) = -(y * \log(p) + (1-y) * \log(1-p))$$

- Sum and average:

$$\boxed{logloss(Y, P) = -\frac{1}{N}\sum_{i=1}^{N}(y_i * \log(p_i) + (1-y_i) * \log(1-p_i))}$$

Note: Unlike linear regression, this doesn't have a closed-form solution

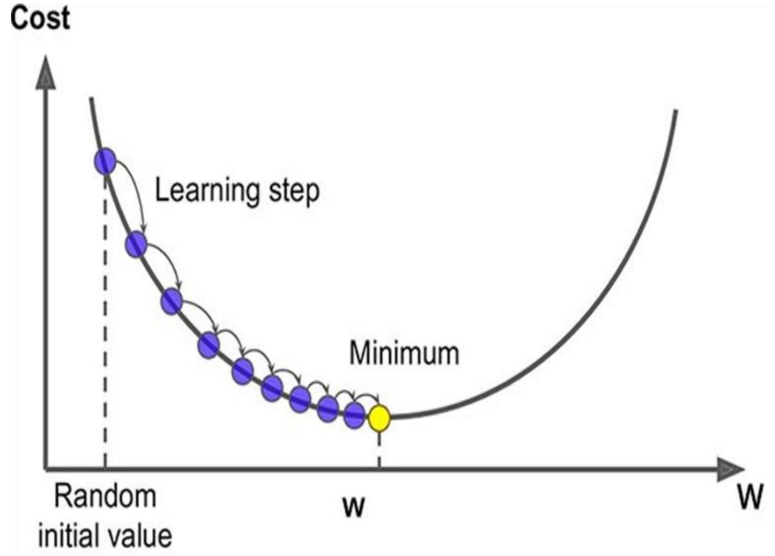
Solution: Use gradient descent (iterative optimization)

How Do We Minimize Cross-Entropy?

$$J(\mathbf{w}) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\sigma(\mathbf{w}^T\mathbf{x}_i)) + (1 - y_i)\log(1 - \sigma(\mathbf{w}^T\mathbf{x}_i))$$

Gradient Descent Algorithm:

1. **Start** with random values for θ_0 and θ_1
2. **Calculate** predictions using current parameters
3. **Compute** cross-entropy loss
4. **Calculate** gradients (how to adjust parameters)
5. **Update** parameters: θ = θ - η × gradient
6. **Repeat** until convergence



Learning rate (η): Controls how big steps we take

- Too large → Might overshoot minimum
- Too small → Very slow convergence

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta \nabla_{\mathbf{w}} J(\mathbf{w}^k)$$

Learning rate

Logistic vs Linear Regression

Aspect	Linear Regression	Logistic Regression
Purpose	Predict continuous values	Predict probabilities/classes
Output	Any real number	Probability (0 to 1)
Function	Straight line	S-shaped curve
Loss	Mean Squared Error	Cross-Entropy
Optimization	Closed-form solution	Gradient descent
Example	House price prediction	Spam detection

Evaluating Classification Models

Accuracy: Simple but sometimes misleading

$$Accuracy = \frac{Correct\ Samples}{All\ Sample}$$

Problem with accuracy:

- In imbalanced datasets, can be misleading
- Example: 95% of emails are not spam
- Model predicting "not spam" always = 95% accurate but useless!

We Need better metrics for classification

Confusion Matrix Basics

Four outcomes for binary classification:

- **True Positive (TP):** Correctly predicted positive
- **True Negative (TN):** Correctly predicted negative
- **False Positive (FP):** Incorrectly predicted positive
- **False Negative (FN):** Incorrectly predicted negative

		Classifier Prediction	
		Positive	Negative
Actual Value	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision and Recall

$$Precision = \frac{TP}{TP+FP} \quad , \quad Recall = \frac{TP}{TP+FN}$$

$$F1 - Score = \frac{2 \ X \ Percision \ X \ Recall}{Precision+Recall}$$

Precision: Of all positive predictions, how many were correct?

- Important when false alarms are costly (fraud detection)

Recall: Of all actual positives, how many did we catch?

- Important when missing positives is costly (cancer detection)

- **F1-Score:** Balance between precision and recall

		Classifier Prediction	
		Positive	Negative
Actual Value	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

Ready to Code!

What we've learned:

- Classification predicts categories, not continuous values
- Logistic regression = Linear regression + Sigmoid function + Cross-entropy loss
- Sigmoid maps any input to probabilities (0 to 1)
- Cross-entropy loss penalizes wrong confident predictions
- Use gradient descent for optimization (no closed-form solution)
- Accuracy isn't always enough - consider precision, recall, F1-score
- Extend to multiclass using softmax function

Ready to Code!

Next: Let's implement this in Python!

Tools we'll use:

- NumPy (for math)
- Pandas (for data)
- Scikit-learn (for models)
- Matplotlib (for visualization)

Exercise

~~**Salary Dataset - Simple linear regression**~~

~~Your task is to visit the link below and create a simple linear regression to **predict the salary** of employees based on the **years of experience** they have.~~

~~**Link:** https://www.kaggle.com/datasets/abhishek14398/salary-dataset-simple-linear-regression/data~~