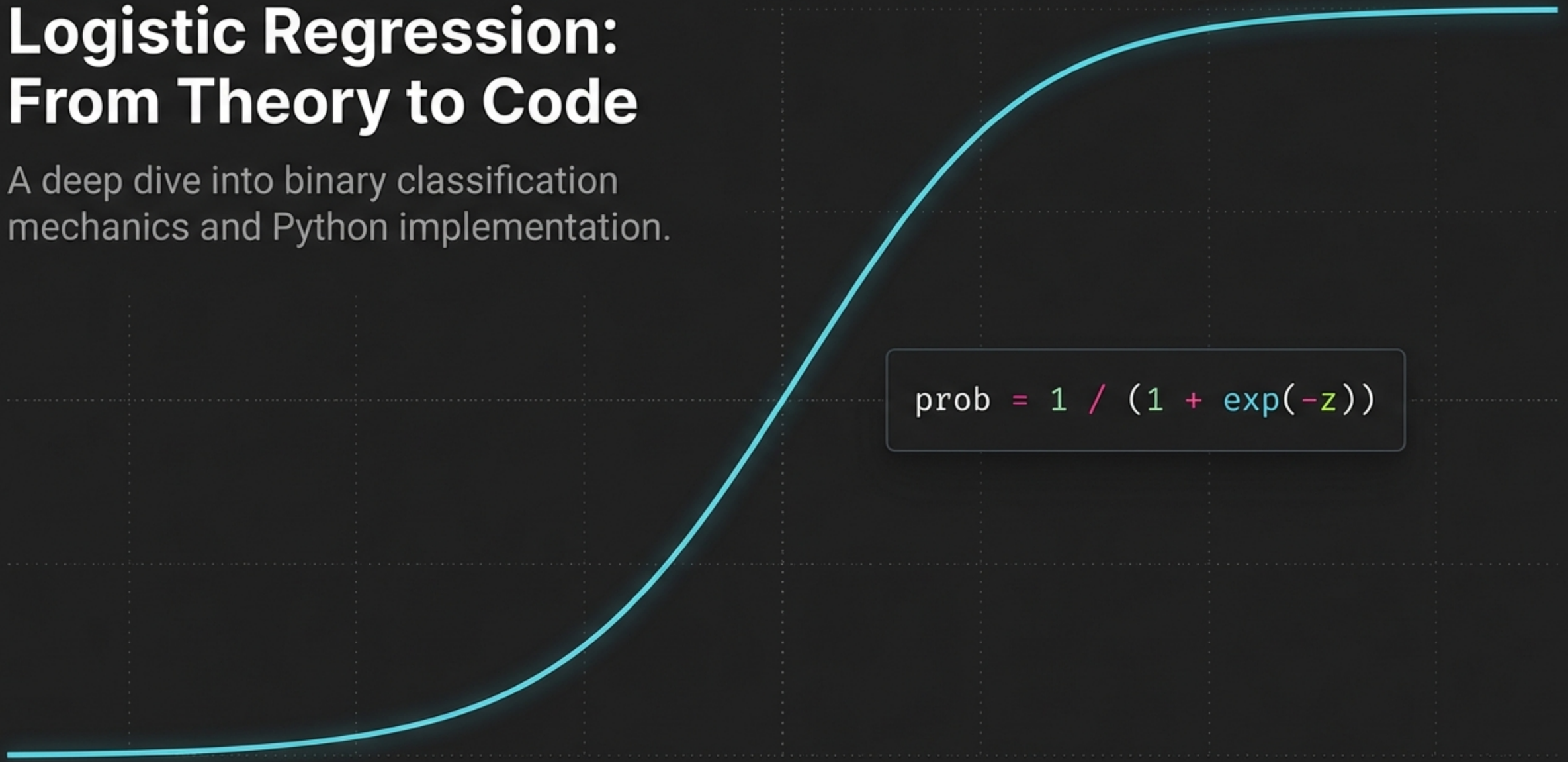
Logistic Regression: From Theory to Code

A deep dive into binary classification mechanics and Python implementation.



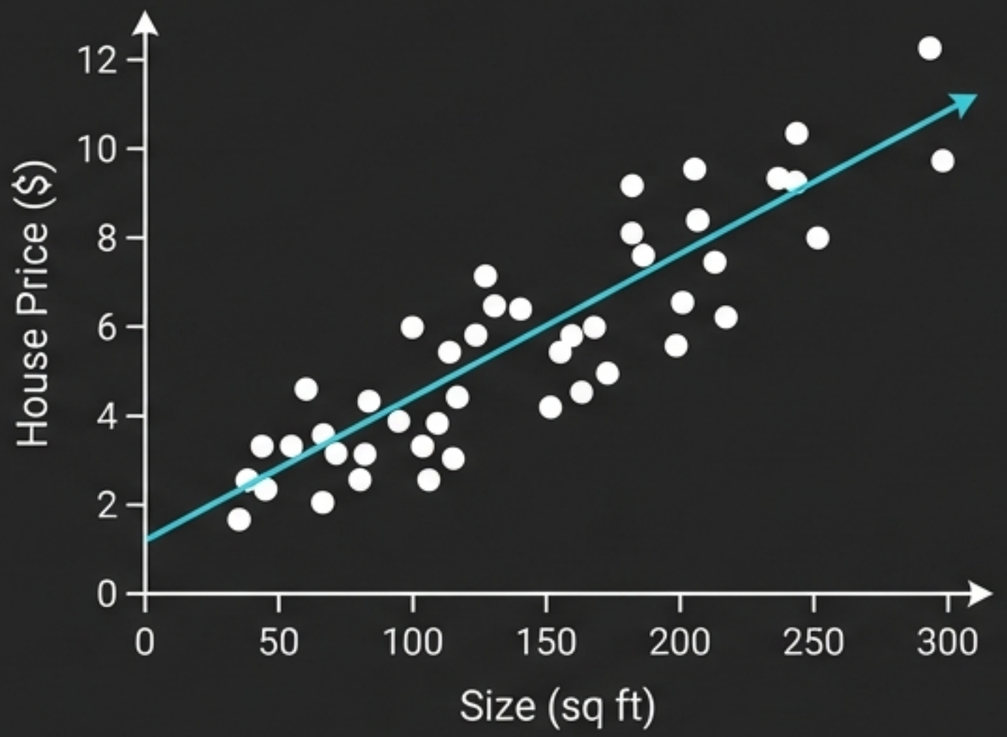
```
prob = 1 / (1 + exp(-z))
```

Supervised Machine Learning Series

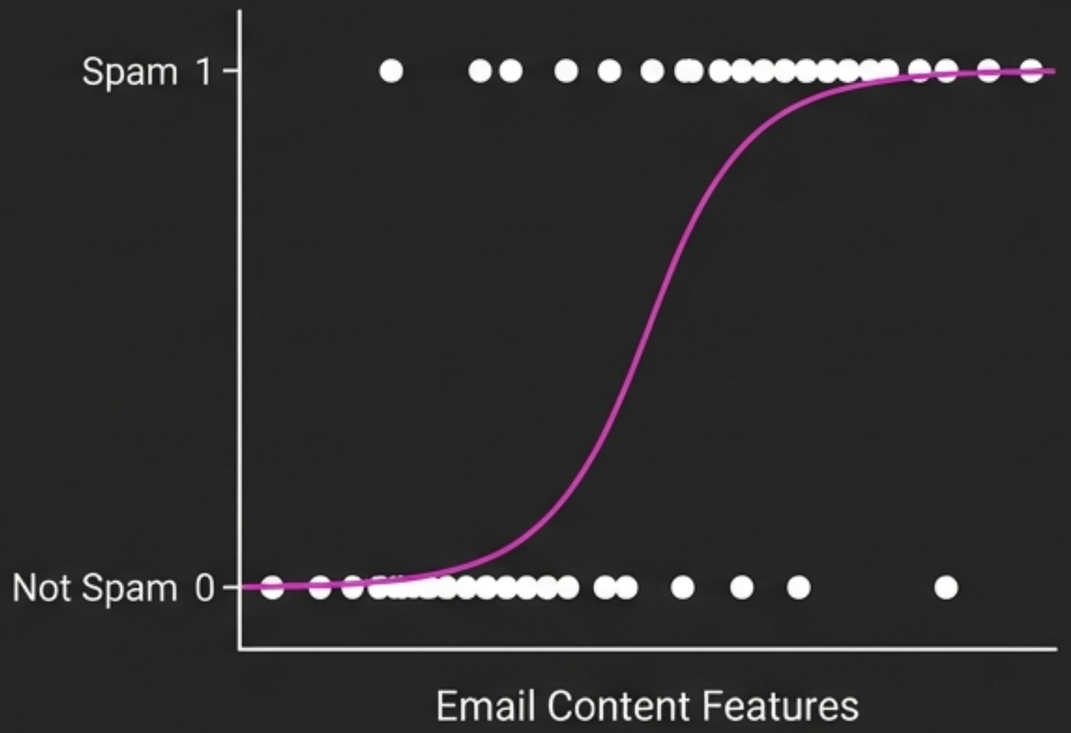
NotebookLM

The Shift from Regression to Classification

Linear Regression (Continuous)



Logistic Regression (Discrete)

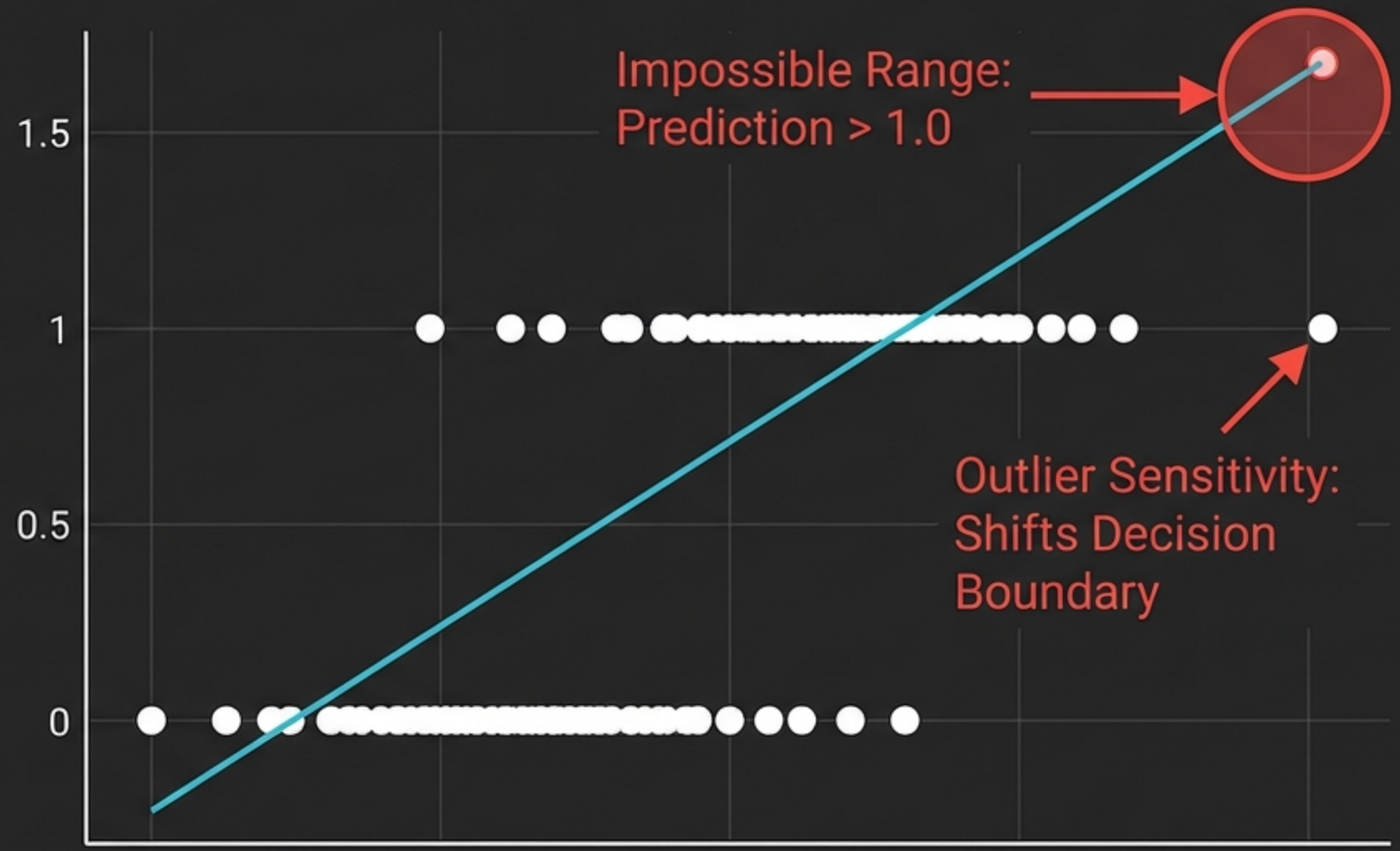


The goal shifts from predicting an infinite value to calculating the probability (0 to 1) of a class membership.

NotebookLM

Why Linear Regression Fails at Classification

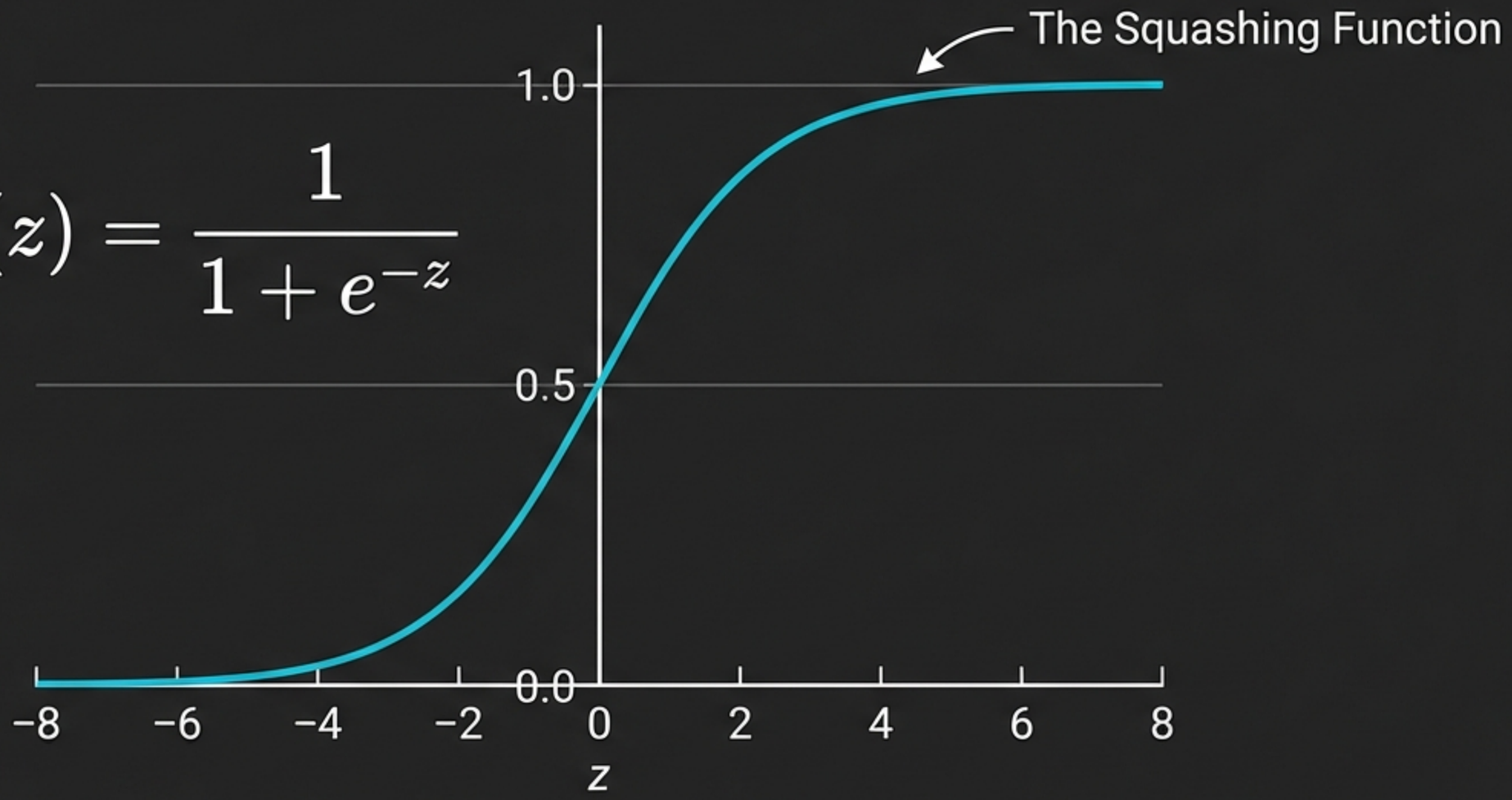
Linear models are unbounded. A prediction of 1.5 or -0.2 is mathematically invalid for probability.



NotebookLM

The Solution: The Sigmoid Function

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

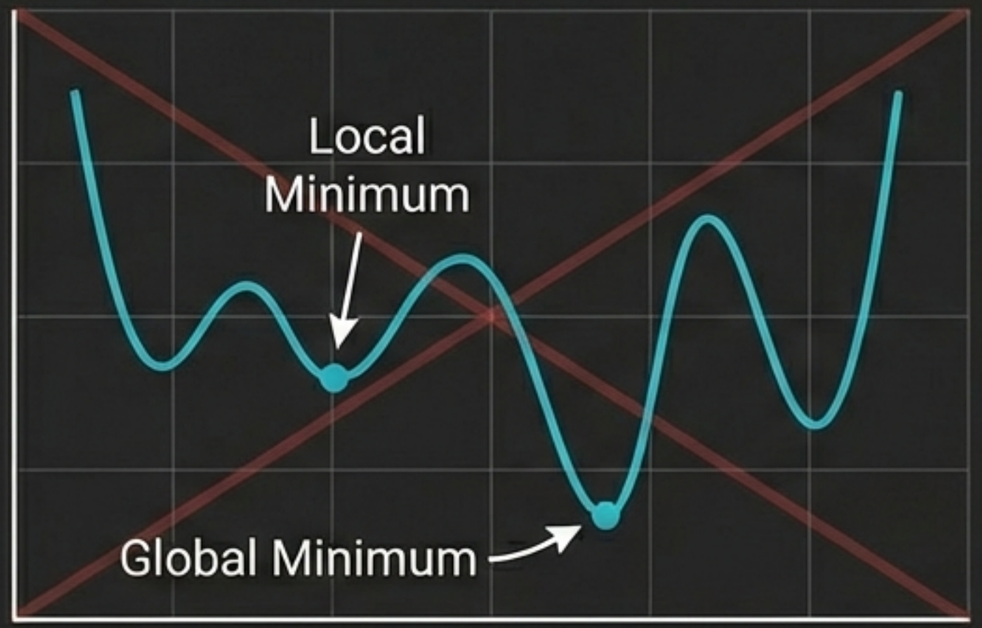


Maps any input z ($-\infty$ to $+\infty$) to a strict probability range of 0 to 1.

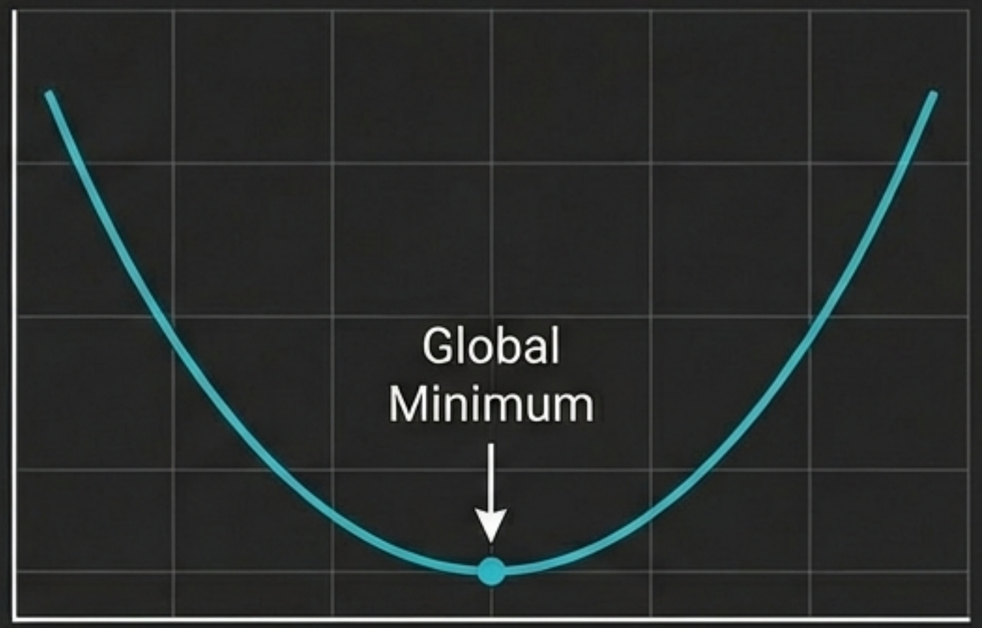
NotebookLM

The Cost Function: Log Loss

MSE with Logistic



Log Loss / Binary Cross-Entropy

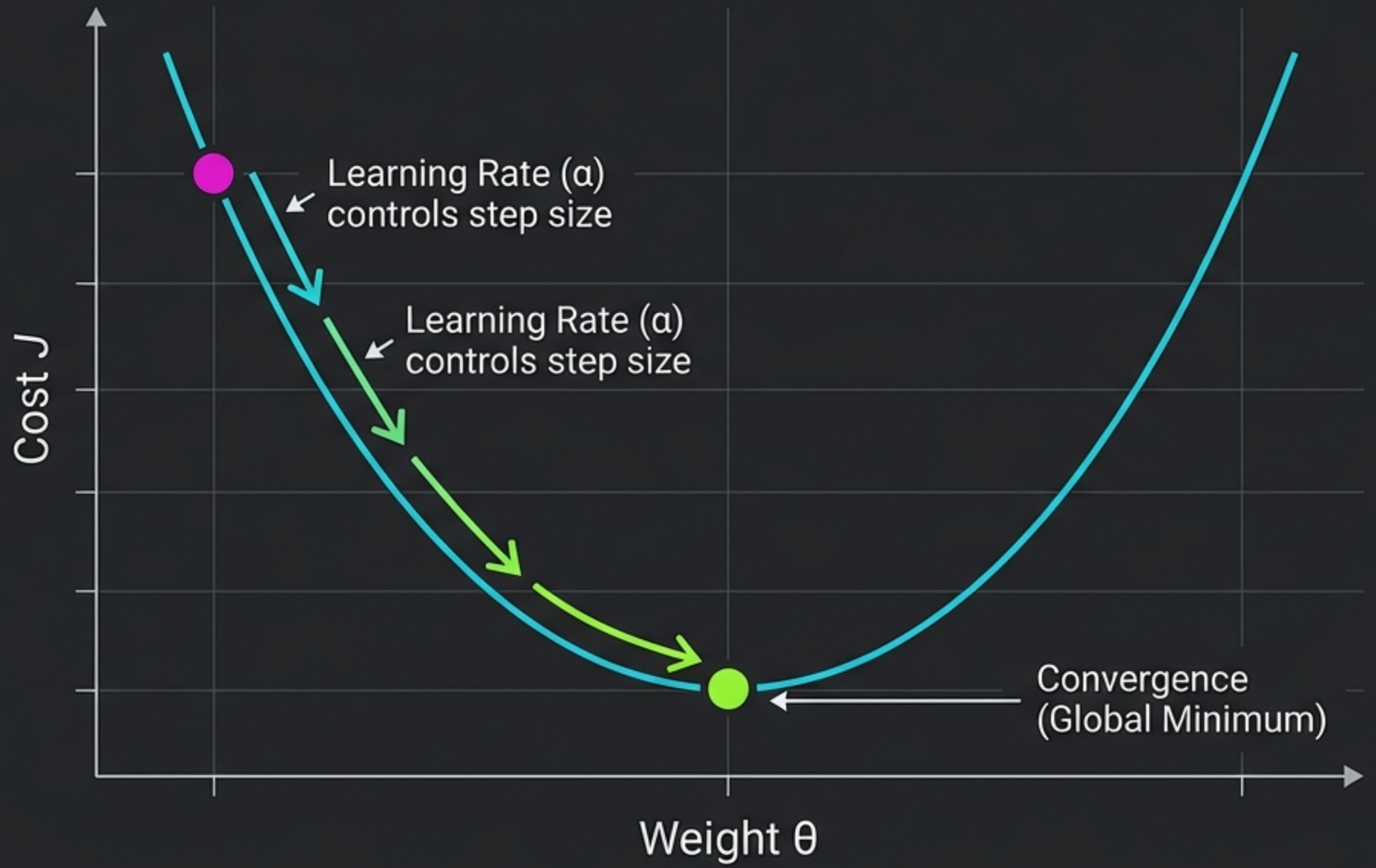


$$\text{Cost} = -y * \log(\hat{y}) - (1-y) * \log(1-\hat{y})$$

We need a convex function (right) to ensure Gradient Descent can always find the lowest error without getting stuck in local valleys.

NotebookLM

Reaching Convergence with Gradient Descent



Iteratively adjusts weights (θ) by moving against the gradient of the error.

NotebookLM

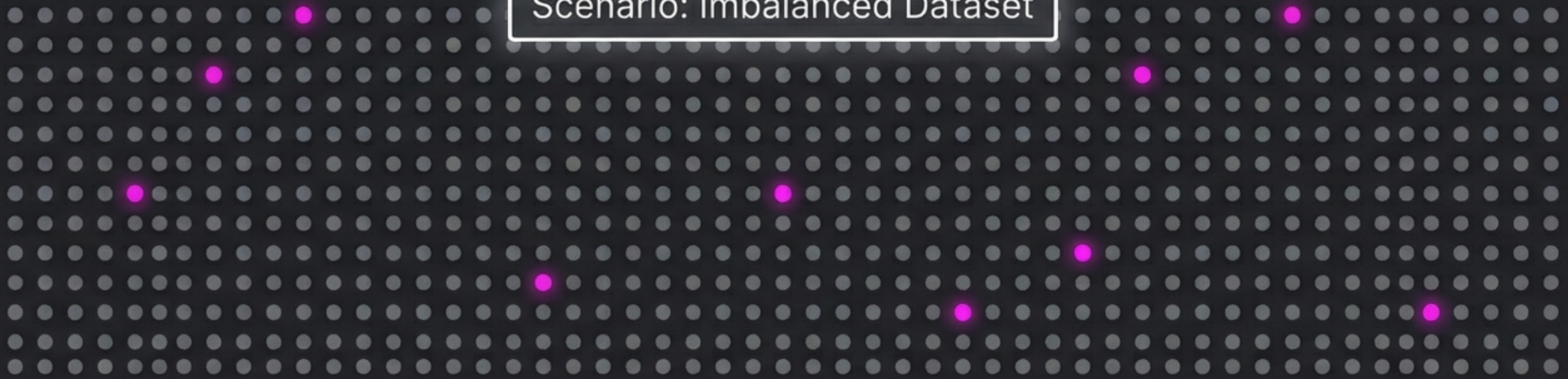
Beyond Accuracy: The Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positive (TP)	False Positive (FP) - Type I Error
Predicted Negative (0)	False Negative (FN) - Type II Error	True Negative (TN)

NotebookLM

The Accuracy Trap

Scenario: Imbalanced Dataset



Model predicts "All Negative"

Captured Positives: 0

Calculated Accuracy: 99%

High Accuracy ≠ Good Model

NotebookLM

Precision vs. Recall

Precision Priority
(Spam Filter)

Recall Priority
(Cancer Diagnosis)

Formula: TP / (TP + FP).

Minimizes False Positives (False Alarms).

Formula: TP / (TP + FN).

Minimizes False Negatives (Missed Detections).

NotebookLM

The F-Beta Score

$$F_\beta = \frac{(1 + \beta^2) * (\text{Precision} * \text{Recall}}{(\beta^2 * \text{Precision}) + \text{Recall}}$$

- β = 1: Harmonic Balance (F1 Score)
- β < 1: Prioritizes **Precision**
- β > 1: Prioritizes **Recall**

A single metric to evaluate performance when you need to balance the trade-off.

NotebookLM

Implementation: Generating the Data

train.py

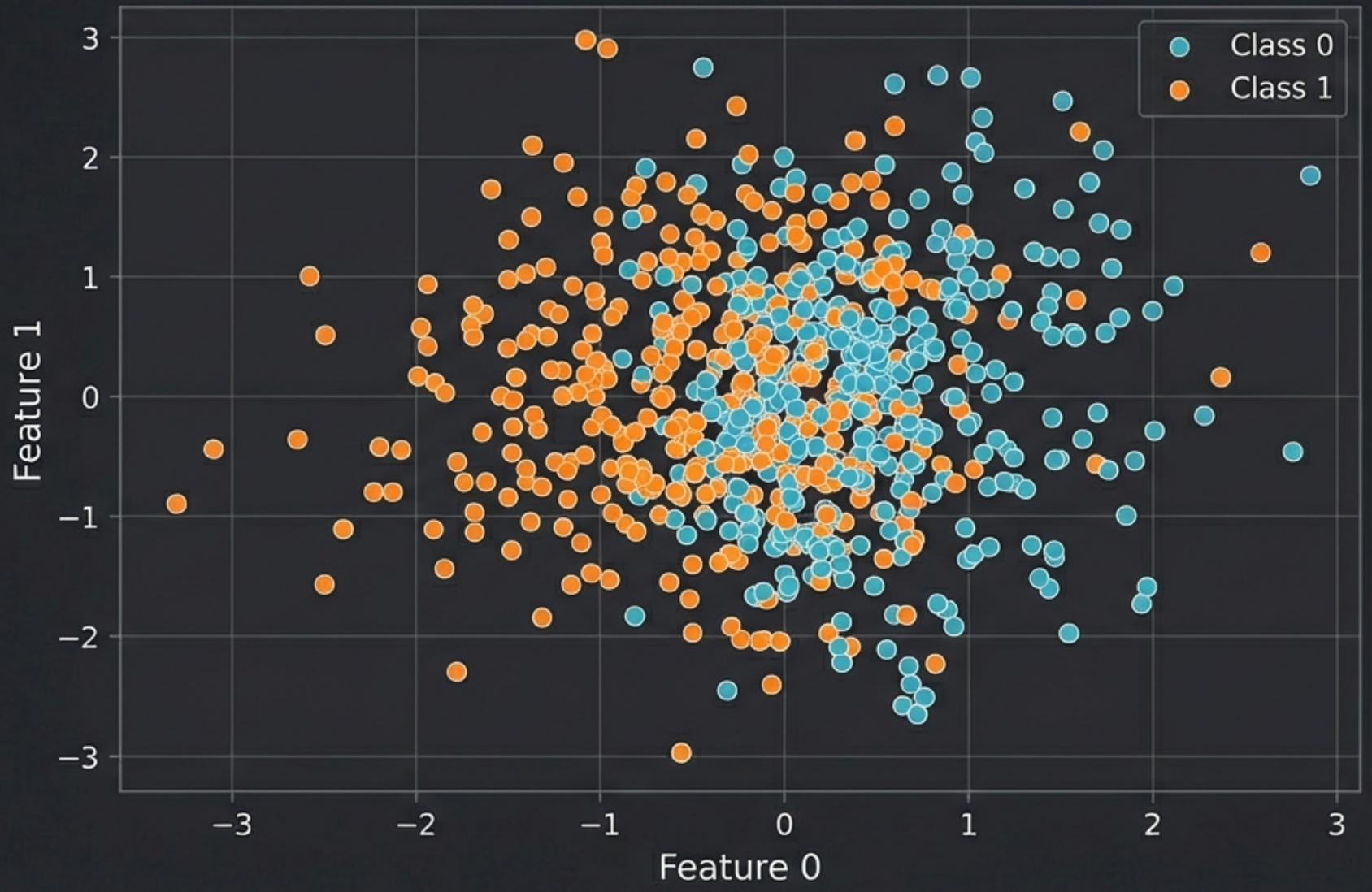
```python
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# Generate synthetic binary data
X, y = make_classification(n_samples=1000, n_classes=2, n_features=10)

# Split into training (75%) and testing (25%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

Dataset Size

Binary Problem (0 or 1)

NotebookLM

Visualizing the Class Separation



Real-world data is rarely perfectly separable. The model must find the probabilistic boundary.

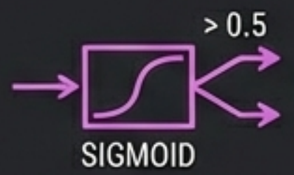
NotebookLM

Training the Model

train.py

```python
from sklearn.linear_model import LogisticRegression

# Initialize
logistic = LogisticRegression()

# 1. Learn the weights (Gradient Descent)
logistic.fit(X_train, y_train)

# 2. Generate predictions (Sigmoid > 0.5)
y_pred = logistic.predict(X_test)
```

Learning



Decision

NotebookLM

Evaluating Performance

```
              precision    recall  f1-score   support

           0       0.86      0.85      0.86       375
           1       0.85      0.86      0.86       375
           1       0.85      0.86      0.86       375

    accuracy                           0.86       750

Confusion Matrix: [[320, 55], [51, 324]]
```

Result Analysis

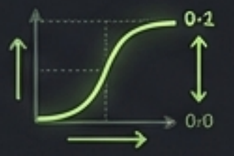
Training Accuracy: 0.884
Test Accuracy: 0.858

Stable Model. No significant overfitting detected.

NotebookLM

Key Takeaways

1. Logistic Regression predicts **probability**, **not continuous values**.
2. The **Sigmoid Curve** maps input data to a **0-1 range**.



3. **Log Loss (Cost Function)** guides the model to the **global minimum**.

4. **Accuracy** is deceptive; use **Precision/Recall** for **imbalanced data**.

5. **Scikit-Learn** implementation requires just 3 steps: `Import`, `Fit`, `Predict`.

```
import LogisticRegression
model.fit()
model.predict()
```

NotebookLM