Machine Learning

Regression, Classification

Context: How Machines Actually Learn

The Optimization Loop

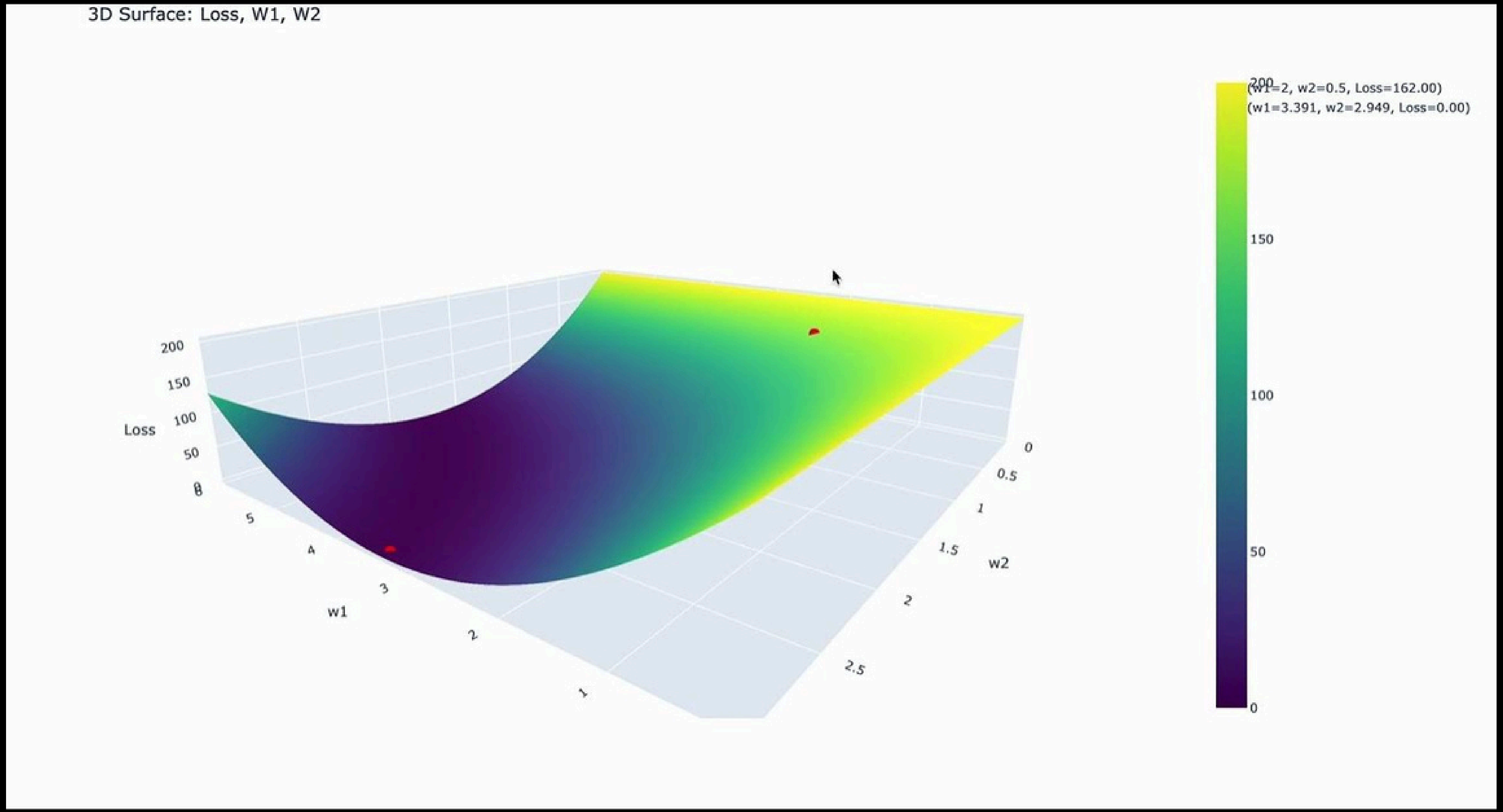
Machine learning isn't magic; it's math. The goal is to minimize error.

- **1. Forward Pass:** Model makes a guess.
- **2. Loss Function:** Calculates how wrong the guess was (Error).
- **3. Optimizer:** Adjusts the weights to reduce error (e.g., Gradient Descent).
- **4. Repeat:** Until the error stops decreasing.

Key Concept: Weights & Biases

The "brain" of the model is just a list of numbers (parameters). "Learning" simply means finding the best numbers for that list.

Visualizing Optimization



Gradient Descent: *The algorithm walks "downhill" on the error surface to find the global minimum (lowest error).*

Step 0: Preprocessing

Train / Test Split

We hide data from the model to test it later. Usually an 80/20 split.

```python
from sklearn.model_selection import train_test_split
X_train, X_test = train_test_split(X, test_size=0.2)
```

Feature Scaling

Algorithms like KNN and SVM break if numbers are on different scales (e.g., Age vs Salary).

```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

The Two Main Paradigms

Supervised

Labeled Data (Inputs + Answers)

We teach the model what to predict.

- Regression (Price)
- Classification (Spam/Not Spam)

Unsupervised

Unlabeled Data (Inputs Only)

The model finds hidden patterns on its own.

- Clustering (Customer segments)
- Dimensionality Reduction

Topic 1: Regression

Predicting Quantity

Regression maps input variables to a **continuous** output.

- Stock Prices
- Temperature
- House Value

$$y = f(x) + \varepsilon$$

Linear Regression

The Equation

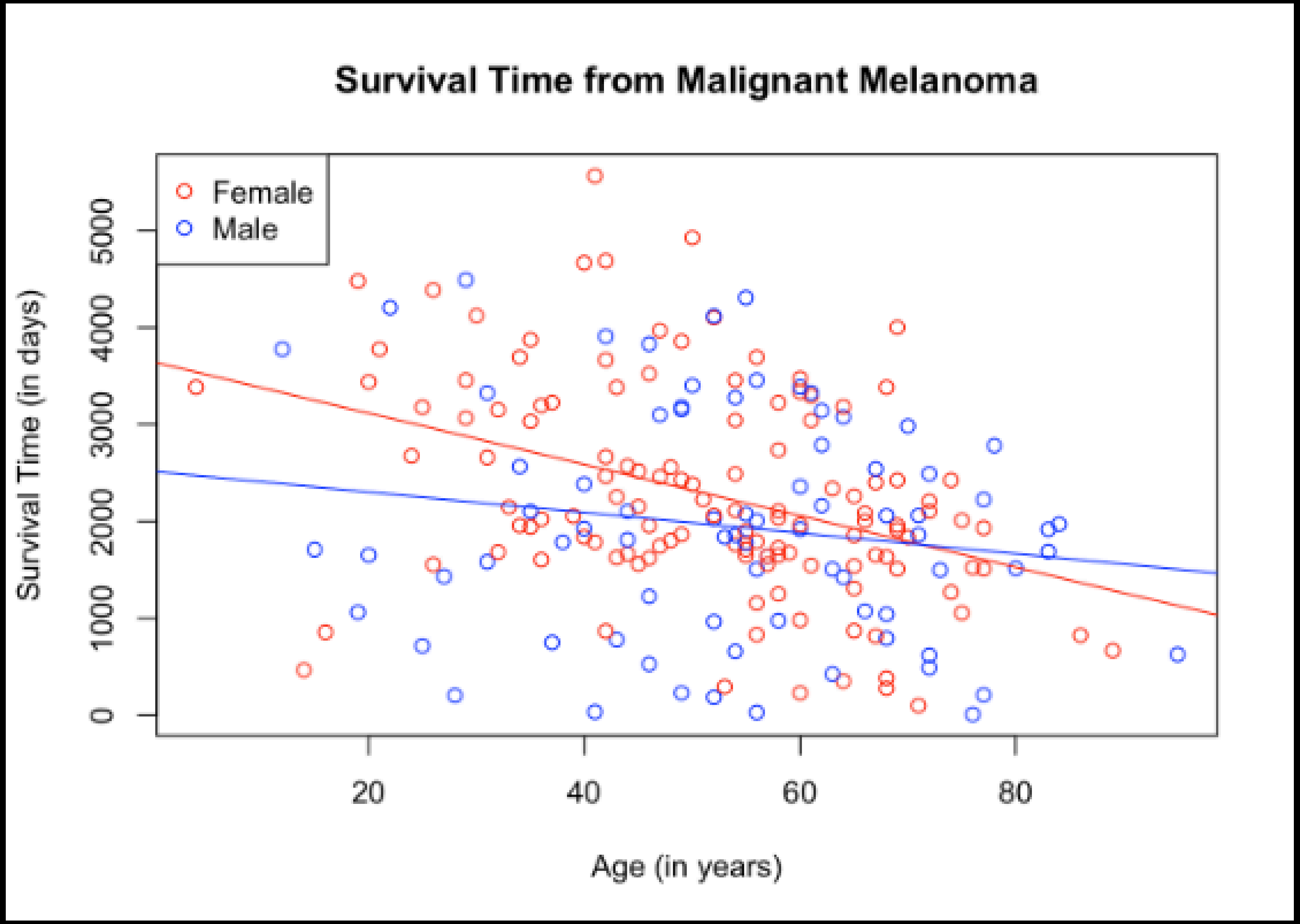
Fitting a straight line through data points.

$$y = \beta_0 + \beta_1 x$$

The model learns β_0 (intercept) and β_1 (slope) to minimize the distance between the line and the dots.

```
from sklearn.linear_model import LinearRegression model =
LinearRegression() model.fit(X_train, y_train)
# Predicts a continuous value preds = model.predict(X_test)
```

Visualizing the Fit



The blue line represents the model's predictions. The vertical distance from each dot to the line is the ***Residual (Error)****.*

Non-Linear Regression

Polynomial Features

Not everything is a straight line. By adding powers of X (x^2 x^3), we can model curves.

Warning: Higher degrees make the model more complex, risking *Overfitting*.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline # Creates x, x^2,
x^3 features poly = make_pipeline(PolynomialFeatures(3),
LinearRegression()) poly.fit(X_train, y_train)
```

Context: Bias-Variance Tradeoff

Underfitting (High Bias)

Model is too simple. It misses the trend.

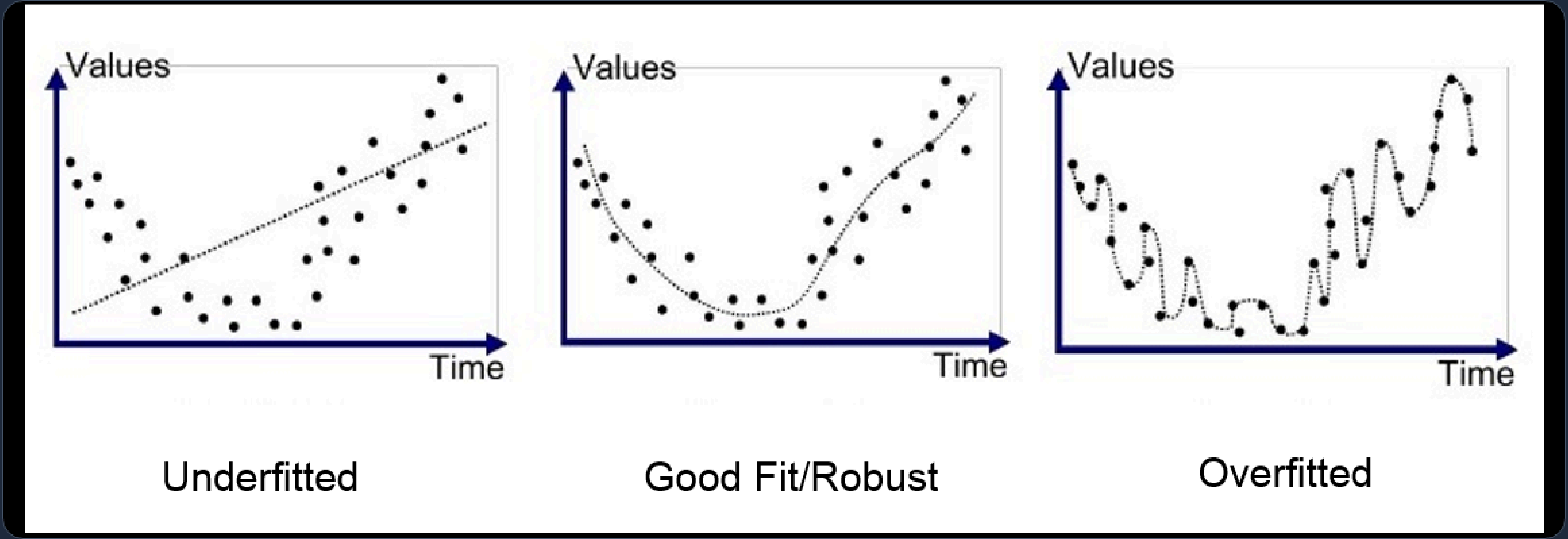
Example: Fitting a straight line to a parabola.

Overfitting (High Variance)

Model is too complex. It memorizes noise.

Example: A squiggly line that touches every single dot.

The Sweet Spot



Our goal is the middle graph: Capturing the general trend without capturing the random noise.

Metrics: MAE vs MSE

MAE (Mean Absolute Error)

Average of absolute errors.

Pros: Easy to explain. "We are off by $5k on average."

Cons: Treats all errors equally.

MSE (Mean Squared Error)

Average of squared errors.

Pros: Penalizes outliers heavily. Large errors become HUGE.

Cons: Harder to interpret (units are squared).

```
from sklearn.metrics import mean_squared_error,
mean_absolute_error
print("MAE:", mean_absolute_error(y_test, predictions))
print("MSE:", mean_squared_error(y_test, predictions))
```

Metrics: RMSE & R-Squared

RMSE

Root Mean Squared Error. Just the square root of MSE. It brings the units back to normal (e.g., dollars), but keeps the penalty for large errors.

R-Squared (R^2)

Score: 0 to 1.

Tells you how much of the variance in the data your model explains. 1.0 is a perfect fit; 0.0 means the model is useless.

Topic 2: Classification

Predicting Category

Mapping input variables to a **discrete** label.

- Spam vs Ham (Binary)
- Tumor: Benign vs Malignant
- Handwriting: 0-9 (Multi-class)

$$P(y = 1 \mid x)$$

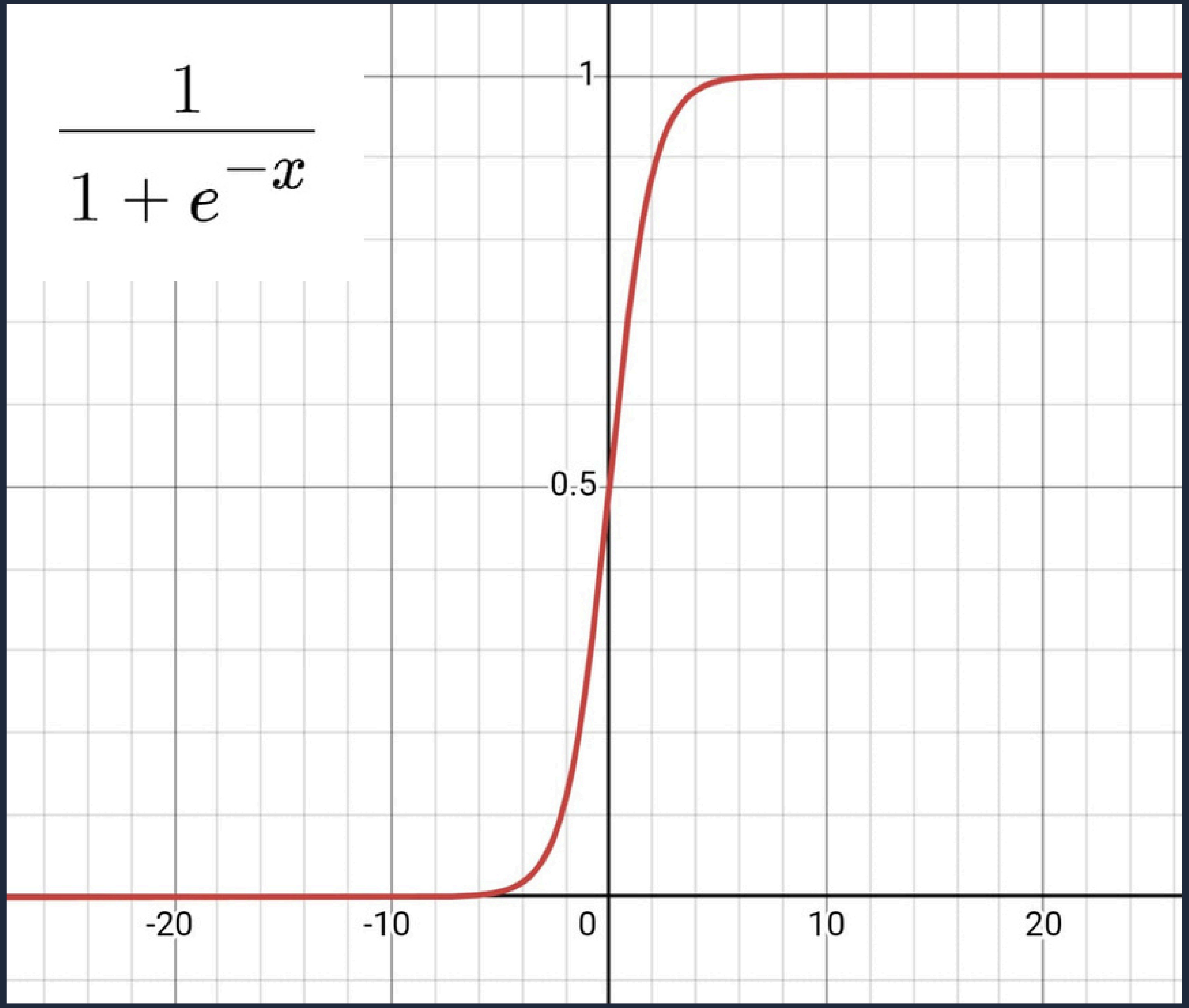
Logistic Regression

It's a Classifier!

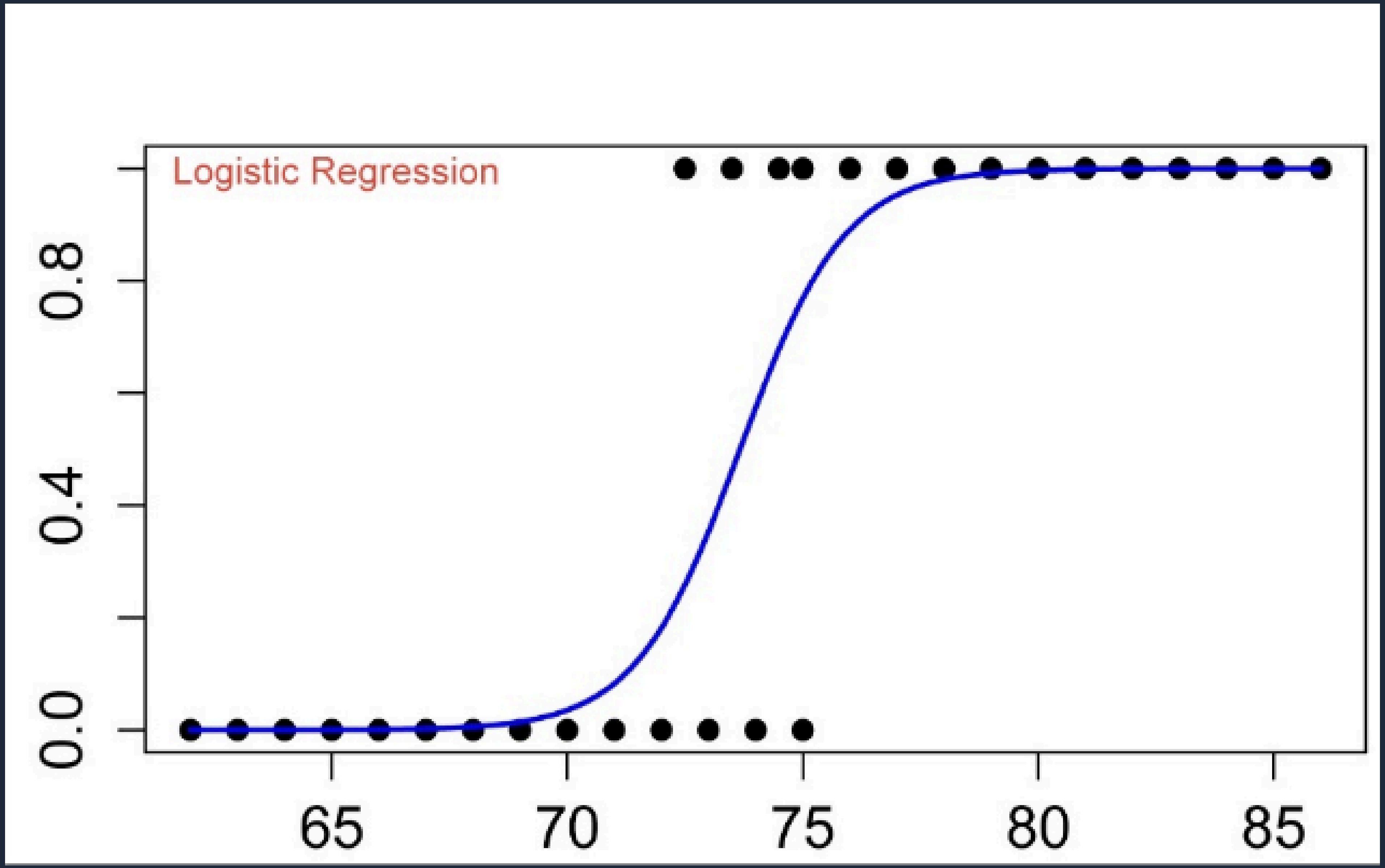
Why not use Linear Regression? Because linear lines go to infinity. Probabilities must stay between 0 and 1.

Solution: Wrap the linear equation in a **Sigmoid Function**.

```
from  sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
# Returns 0 or 1 preds = clf.predict(X_test)
```



The Sigmoid Curve



The "S" curve forces any input number into a safe probability range (0 to 1). We typically set a threshold at 0.5.

K-Nearest Neighbors (KNN)

"Birds of a feather flock together"

KNN assumes similar things exist in close proximity. To classify a new point, it looks at the 'K' nearest points.

Note: Very slow on large datasets because it has to calculate distance to *every* point.

```
from sklearn.neighbors import KNeighborsClassifier
# K=3 means look at 3 closest neighbors
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
```

Context: The Impact of 'K'

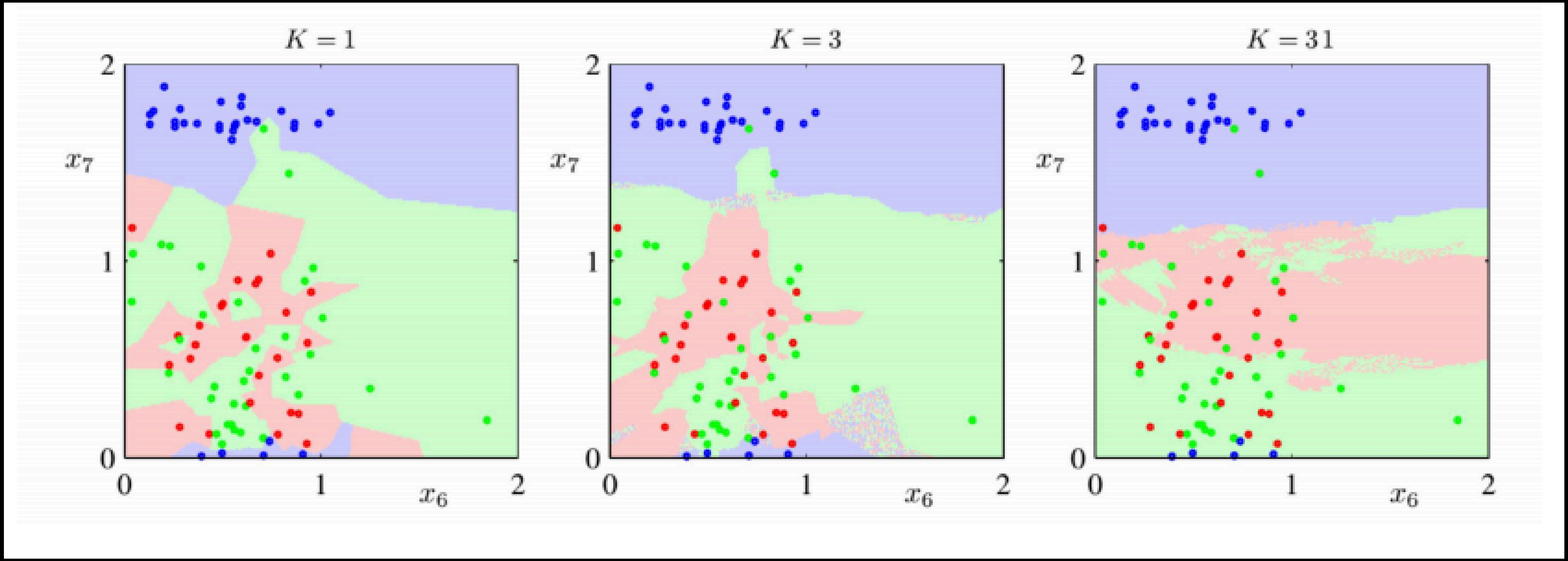
Low K (e.g., K=1)

Overfitting. The boundary is jagged and reacts to every single noise point. If your neighbor is an outlier, you get misclassified.

High K (e.g., K=100)

Underfitting. The boundary becomes too smooth. It ignores local details and just votes for the majority class of the whole area.

KNN Decision Boundaries



Notice how small K creates islands of complex boundaries, while large K creates smooth, simple lines.

Decision Trees

20 Questions

The model asks a series of Yes/No questions to split the data.

Goal: Create "Pure" leaves (groups where everyone belongs to the same class).

```
from sklearn.tree import DecisionTreeClassifier
# max_depth limits how many questions it can ask
dt = DecisionTreeClassifier(max_depth=5)
dt.fit(X_train, y_train)
```

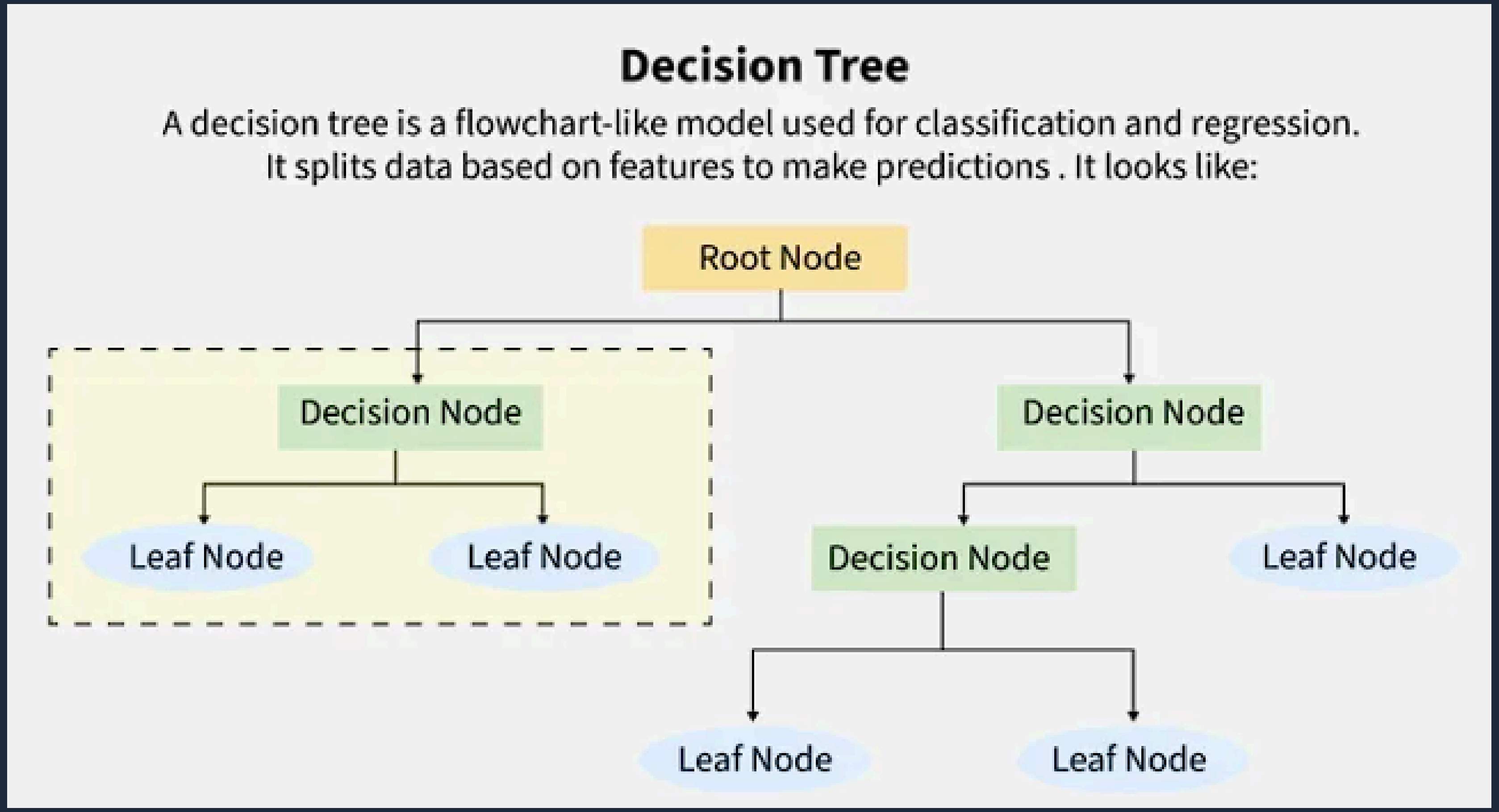
Context: How it Splits

Entropy / Gini Impurity

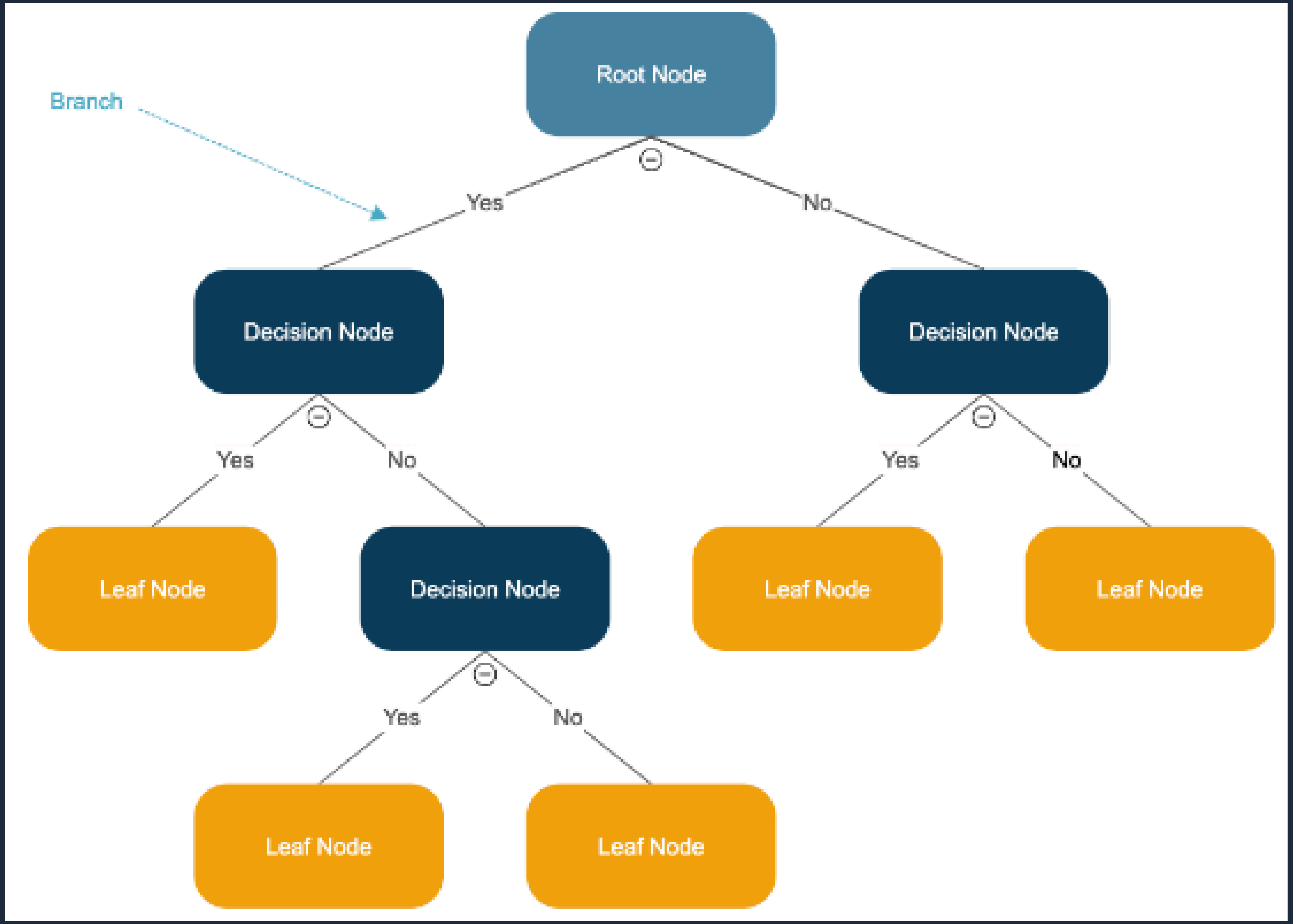
The tree calculates a score to measure "Messiness".

- **High Entropy:** 50% Cats, 50% Dogs (Messy).
- **Low Entropy:** 100% Cats (Pure).

The algorithm greedily chooses the question (e.g., "Is Weight > 10kg?") that reduces entropy the most.


Decision Tree
A decision tree is a flowchart-like model used for classification and regression.
It splits data based on features to make predictions . It looks like:
Root Node
Decision Node
Decision Node
Leaf Node
Leaf Node
Decision Node
Leaf Node
Leaf Node
Leaf Node

Tree Visualization



Interpretable AI: Unlike neural networks, you can simply follow the path to understand why the model made a decision.

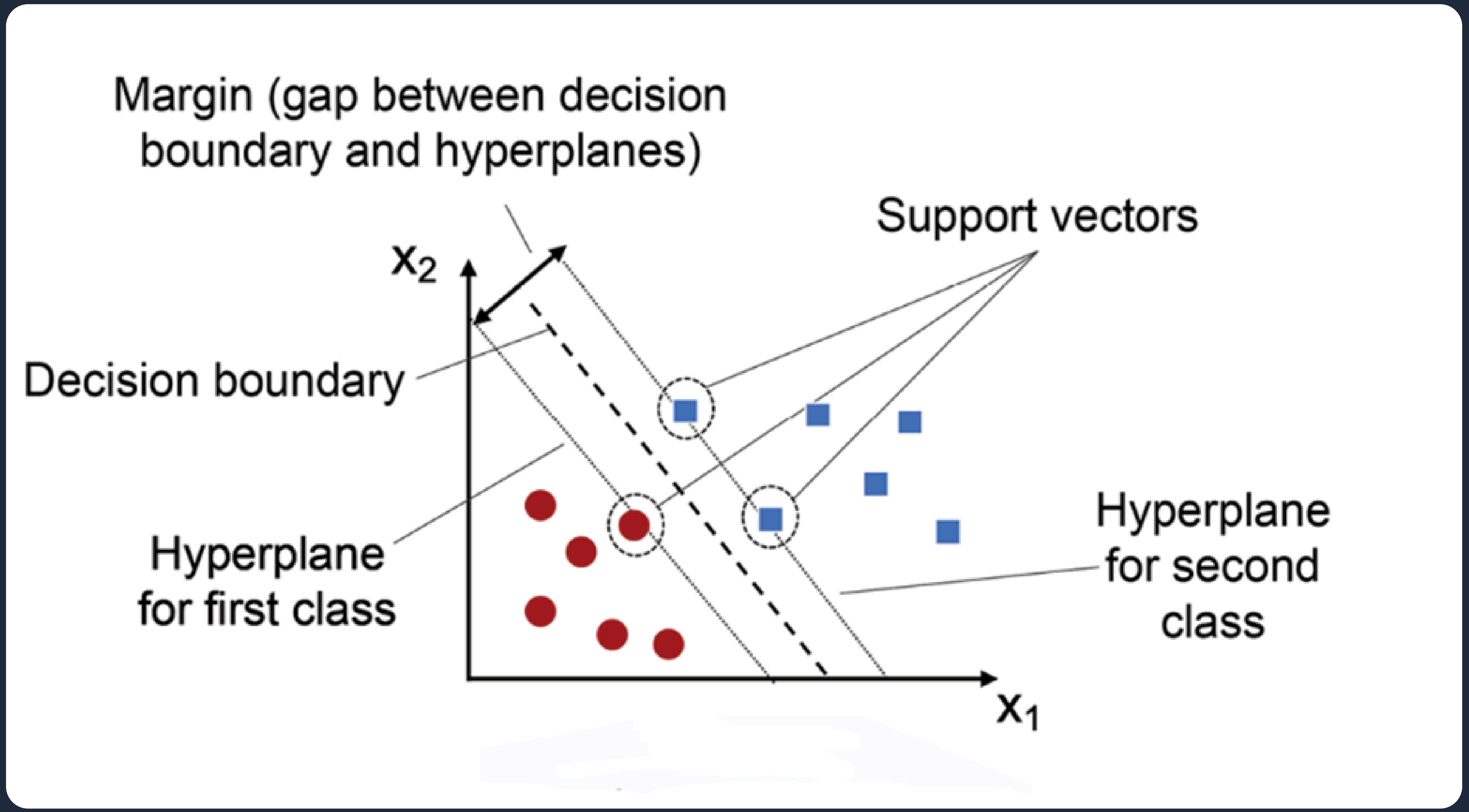
Support Vector Machines (SVM)

The Widest Street

SVM doesn't just find *a* line that separates classes; it finds the *best* line.

Margin: The distance between the boundary and the closest points (Support Vectors). SVM maximizes this margin.

```python
from sklearn.svm import SVC
# kernel='linear' for straight lines
svm = SVC(kernel='linear')
svm.fit(X_train, y_train)
```


Margin (gap between decision boundary and hyperplanes)
Support vectors
X_2
Decision boundary
Hyperplane for first class
Hyperplane for second class
X_1

Context: The Kernel Trick

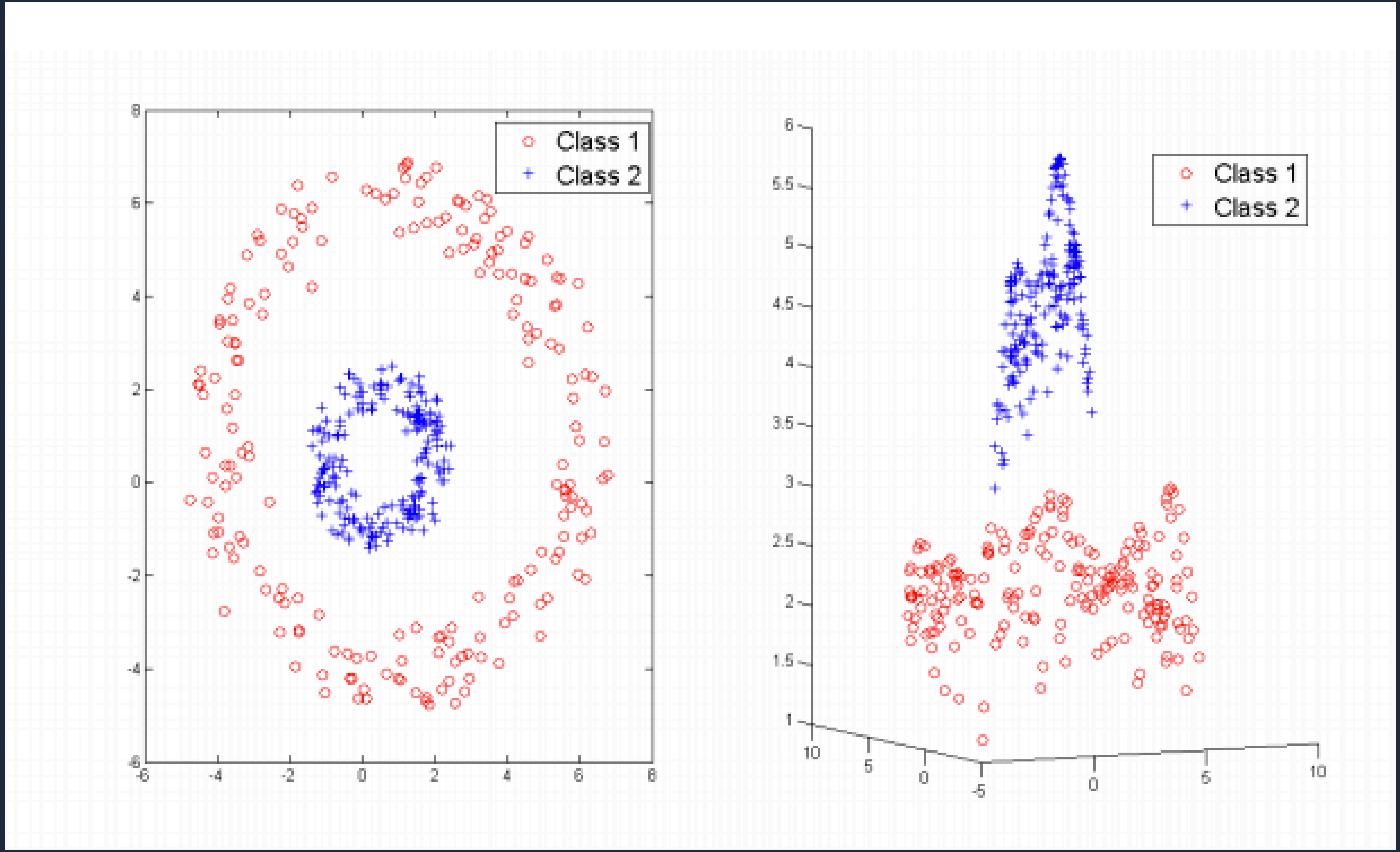
What if data isn't separable?

Imagine red dots in the center and blue dots surrounding them. A straight line can't split them.

Solution: Project data into 3D! In 3D, you might be able to slide a flat sheet (hyperplane) between them. When you project back to 2D, the boundary looks like a circle.

```
# 'rbf' is the default kernel for non-linear data svm =
SVC(kernel='rbf')
```

Visualizing 2D to 3D Projection



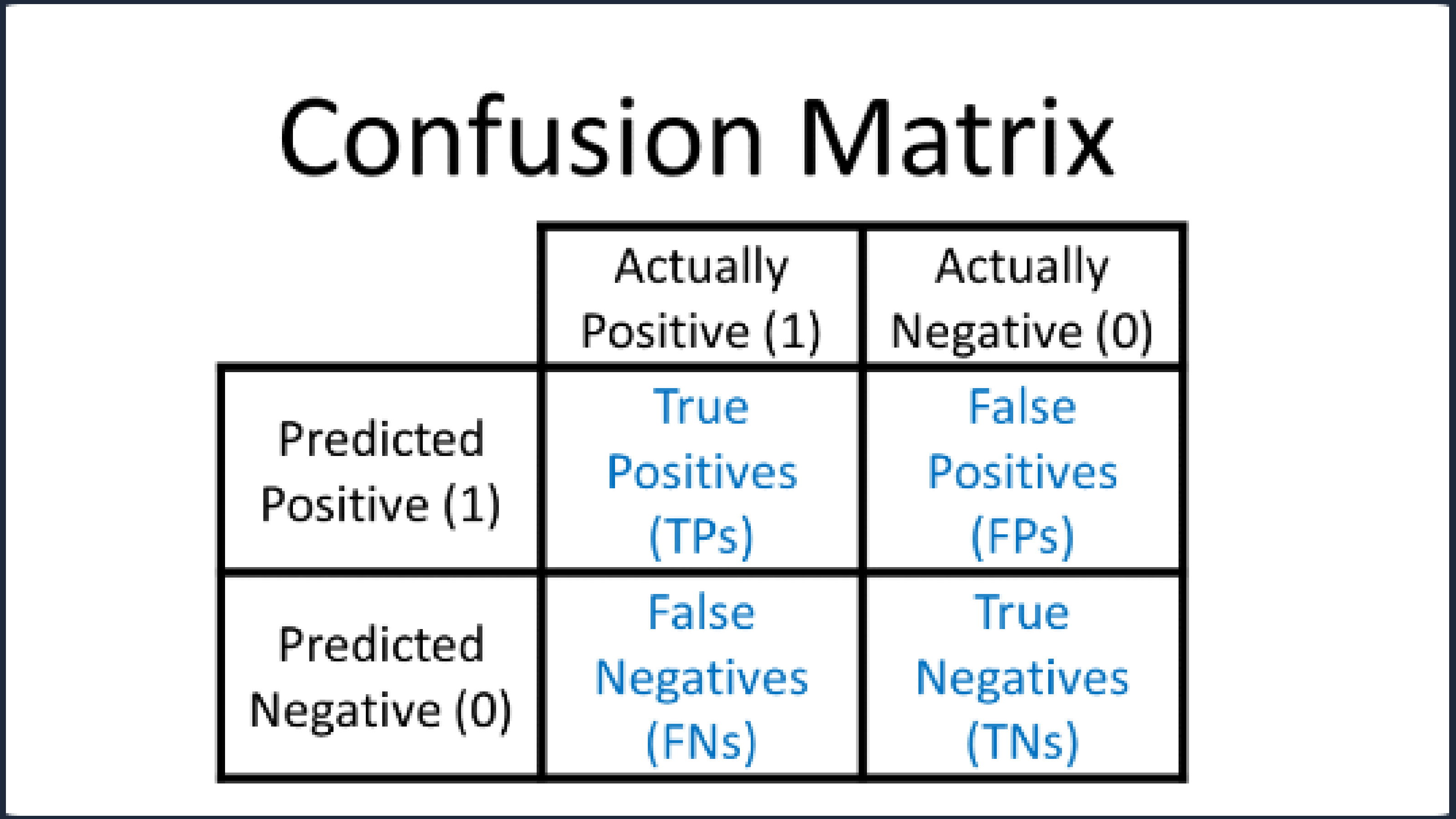
By lifting the data into higher dimensions, complex non-linear problems become simple linear ones.

Evaluation: Confusion Matrix

The 4 Outcomes

Accuracy hides details. We need to know *how* we are wrong.

- **True Positive (TP):** Correctly spotted Fire.
- **True Negative (TN):** Correctly ignored Safe.
- **False Positive (FP):** False Alarm (Type I).
- **False Negative (FN):** Missed Fire (Type II - Dangerous!).



Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Context: Precision vs Recall

Precision

"Of all the alarms we rang, how many were real fires?"

Optimize this to avoid spamming users (False Positives).

$$P = \frac{TP}{(TP + FP)}$$

Recall

"Of all the real fires, how many did we detect?"

Optimize this for cancer detection (False Negatives are fatal).

$$R = \frac{TP}{(TP + FN)}$$

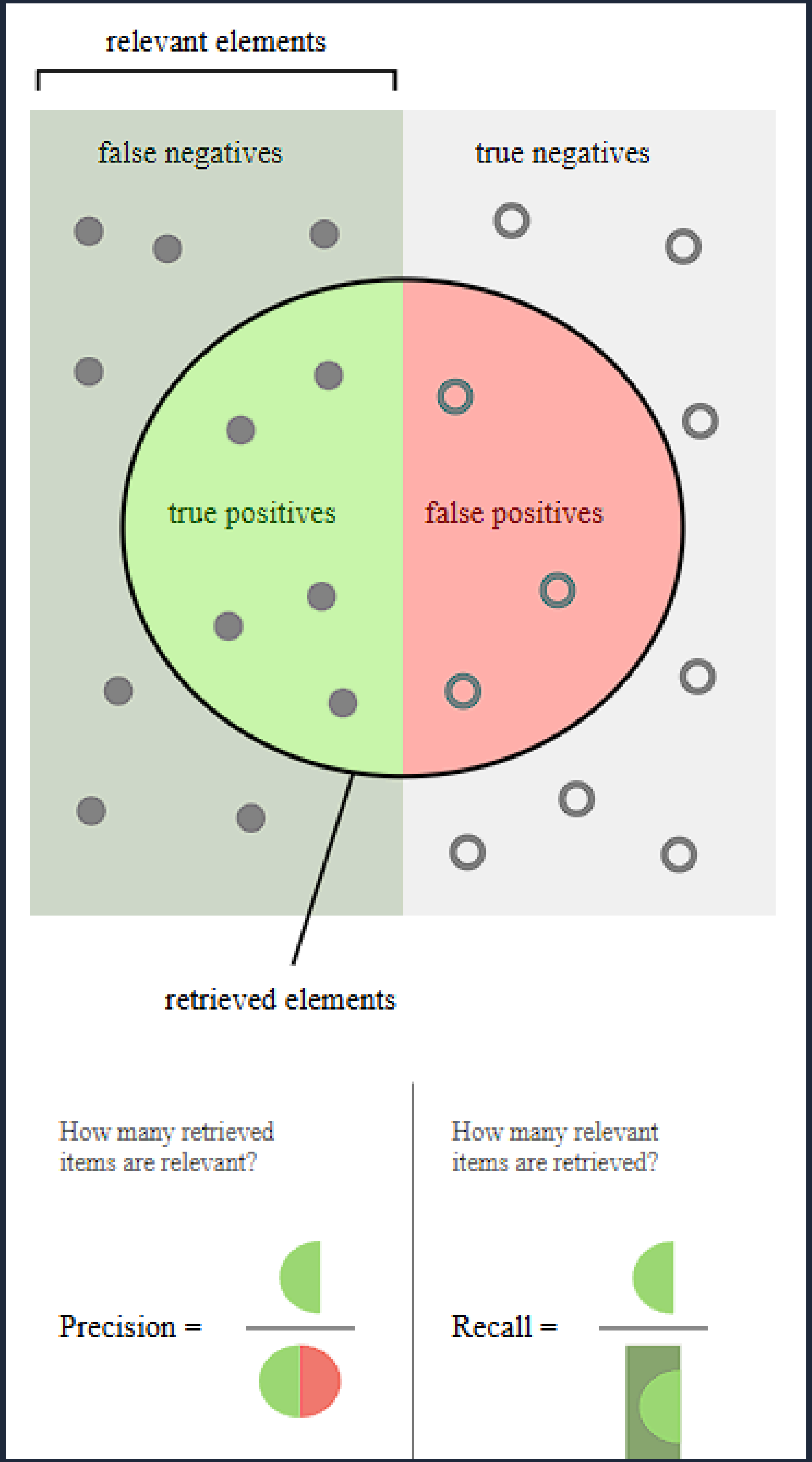
The F1-Score

Balancing the Two

You can't have perfect Precision and Recall simultaneously. As you catch more cases (Recall ↑), you inevitably make more false alarms (Precision ↓).

F1-Score: The Harmonic Mean of Precision and Recall. Use this metric when classes are imbalanced.

$$F1 = \frac{2*P*R}{(P+R)}$$



Evaluation Code

```python
from sklearn.metrics import classification_report, confusion_matrix
print(confusion_matrix(y_test, preds)) # Prints Precision, Recall, F1 for each class
print(classification_report(y_test, preds))
```

Masterclass Summary

Regression

- Goal: Continuous #
- Alg: Linear, Polynomial
- Metric: MSE, R2
- Key: Gradient Descent

Classification

- Goal: Category Label
- Alg: LogReg, KNN, SVM, Tree
- Metric: F1, Confusion Matrix
- Key: Decision Boundary

Standard Library Workflow

Whether using Scikit-Learn, PyTorch, or TensorFlow, the pattern remains consistent.

1. Instantiate

Select your model class.

```
model =
LogisticRegression()
```

2. Fit

Train on labeled data.

```
model.fit(X_train, y_train)
```

3. Predict

Infer on new data.

```
preds =
model.predict(X_test)
```

4. Evaluate

Compare with ground truth.

```
accuracy_score(y_test,
preds)
```

Questions?

Thank you for attending.