

MACHINE LEARNING

SUPERVISED

CHAP2

CLASSIFICATION/ LOGISTIC REGRESSION

Jean Tabet

j.tabet@ul.edu.lb

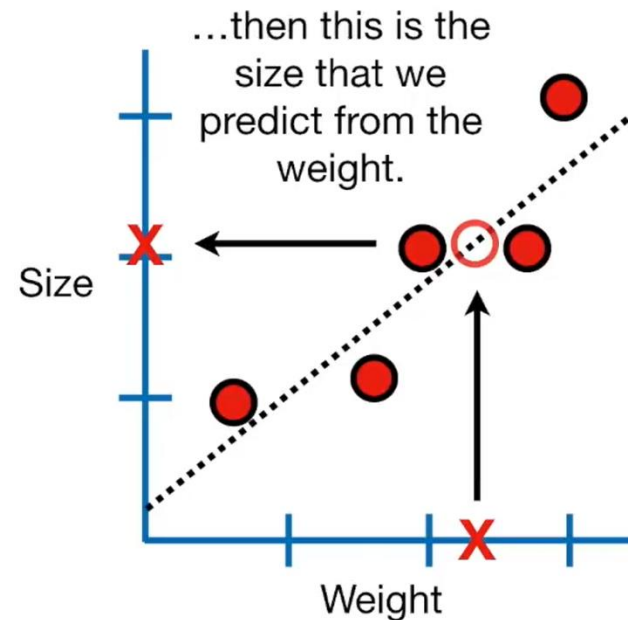
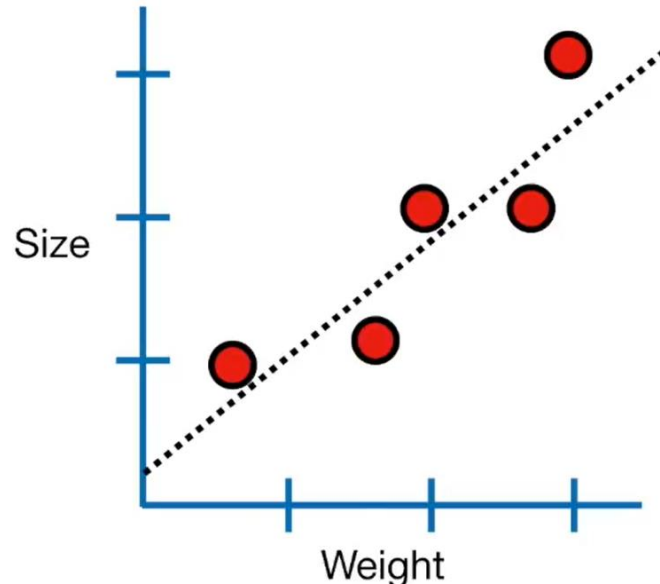
→ LOGISTIC REGRESSION

- Email: Spam / Not Spam ?
- Online Transactions: Fraudulent (Yes/No)?
- Tumor: Malignant / Benign?
- Loan Demand (Credit Risk): Safe / Risky

➔ LINEAR REGRESSION (REVIEW)

We fit the data with a line. Then, we:

- Calculate R^2 and determine if weight and size are correlated. Large values imply a large effect.
- Calculate a p-value to determine if the R^2 value is statistically significant.
- Use the line to predict size given weight.

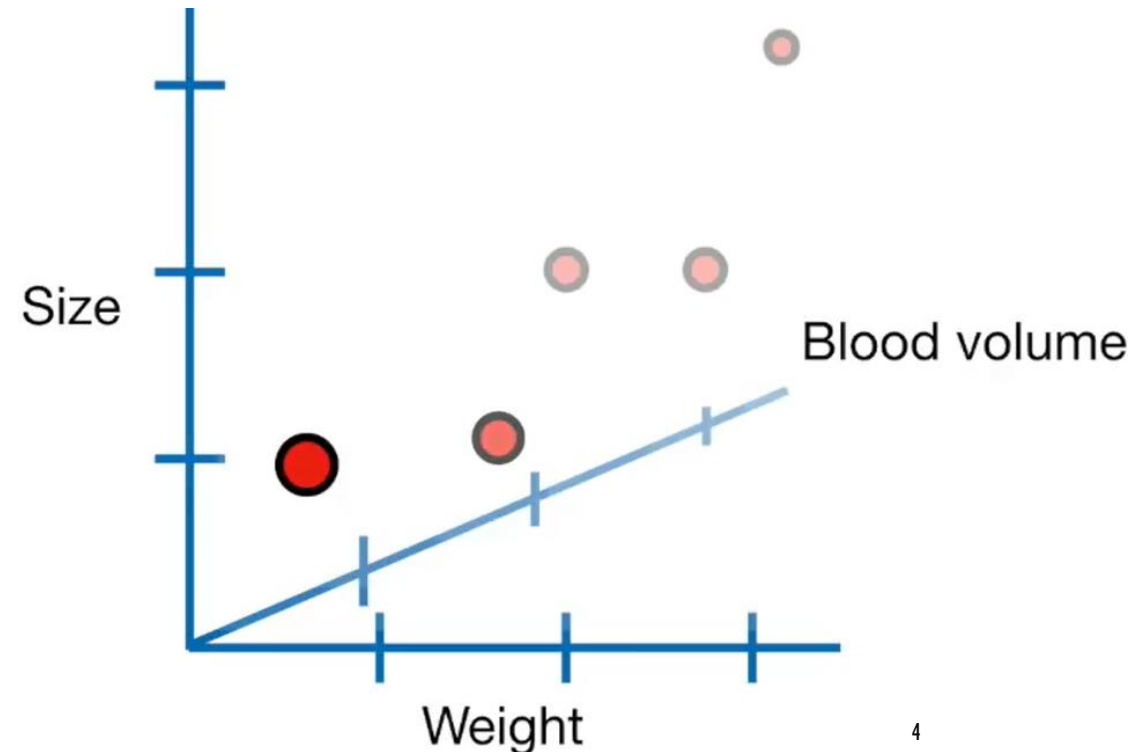


→ MULTIPLE REGRESSION (REVIEW)

With multiple regression, we can model size using weight and blood volume.

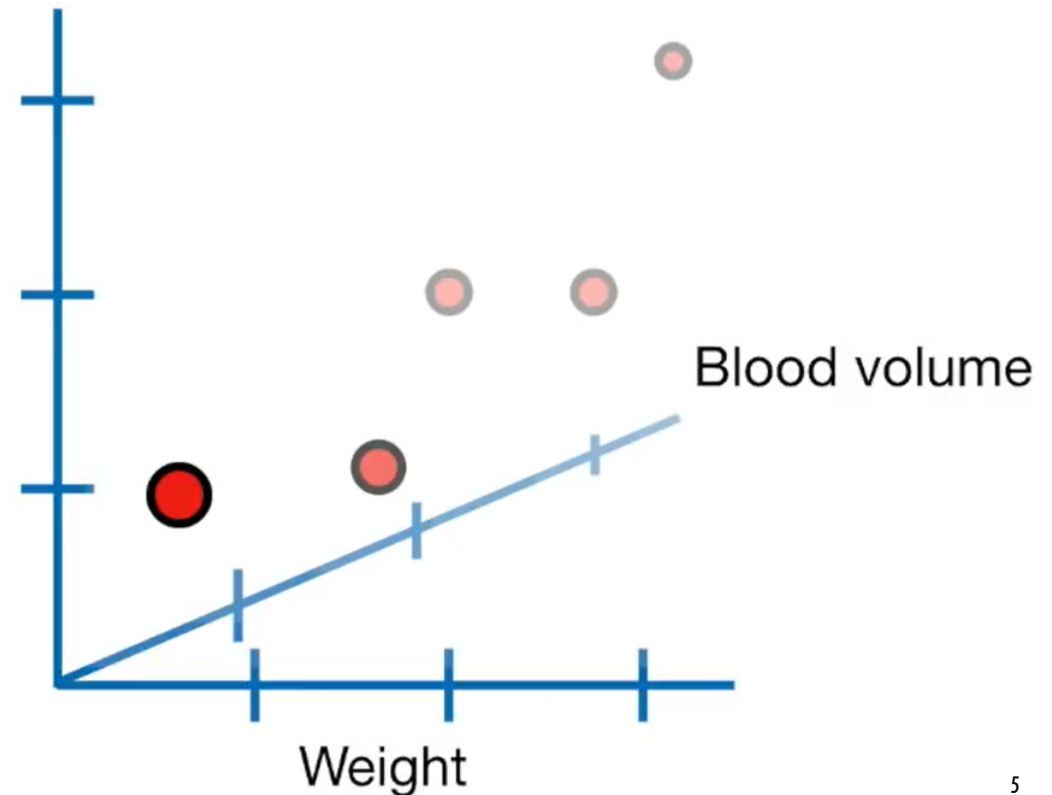
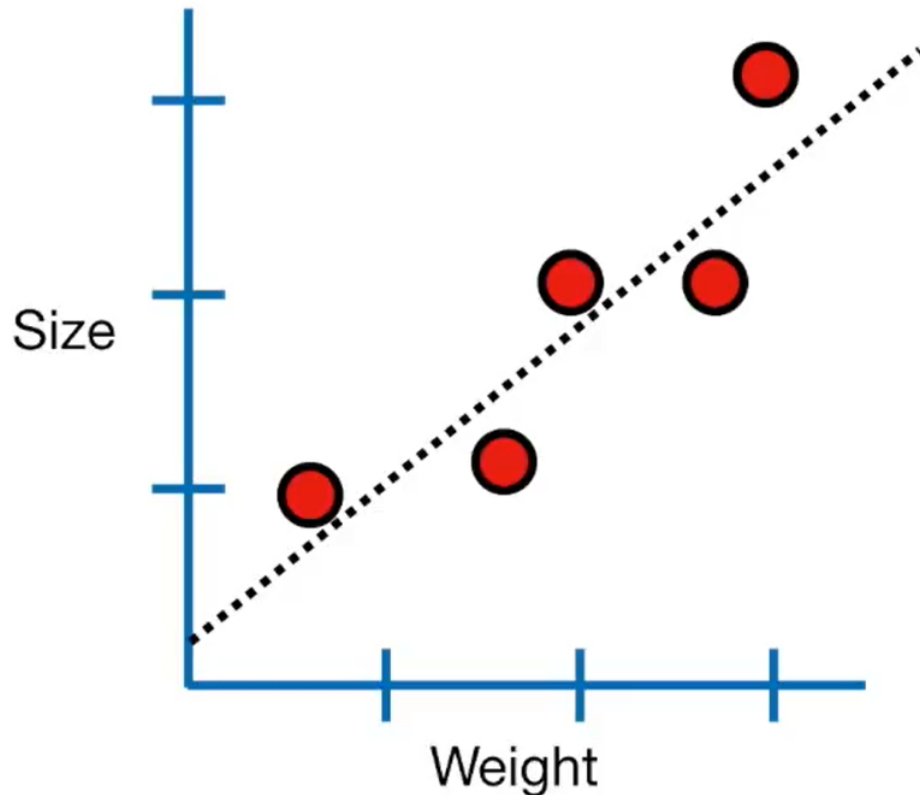
Multiple regression did the same things that normal regression did...

- Calculate R^2 , adjusted R^2 .
- Calculate the p-value.
- Predict size given weight and blood volume.

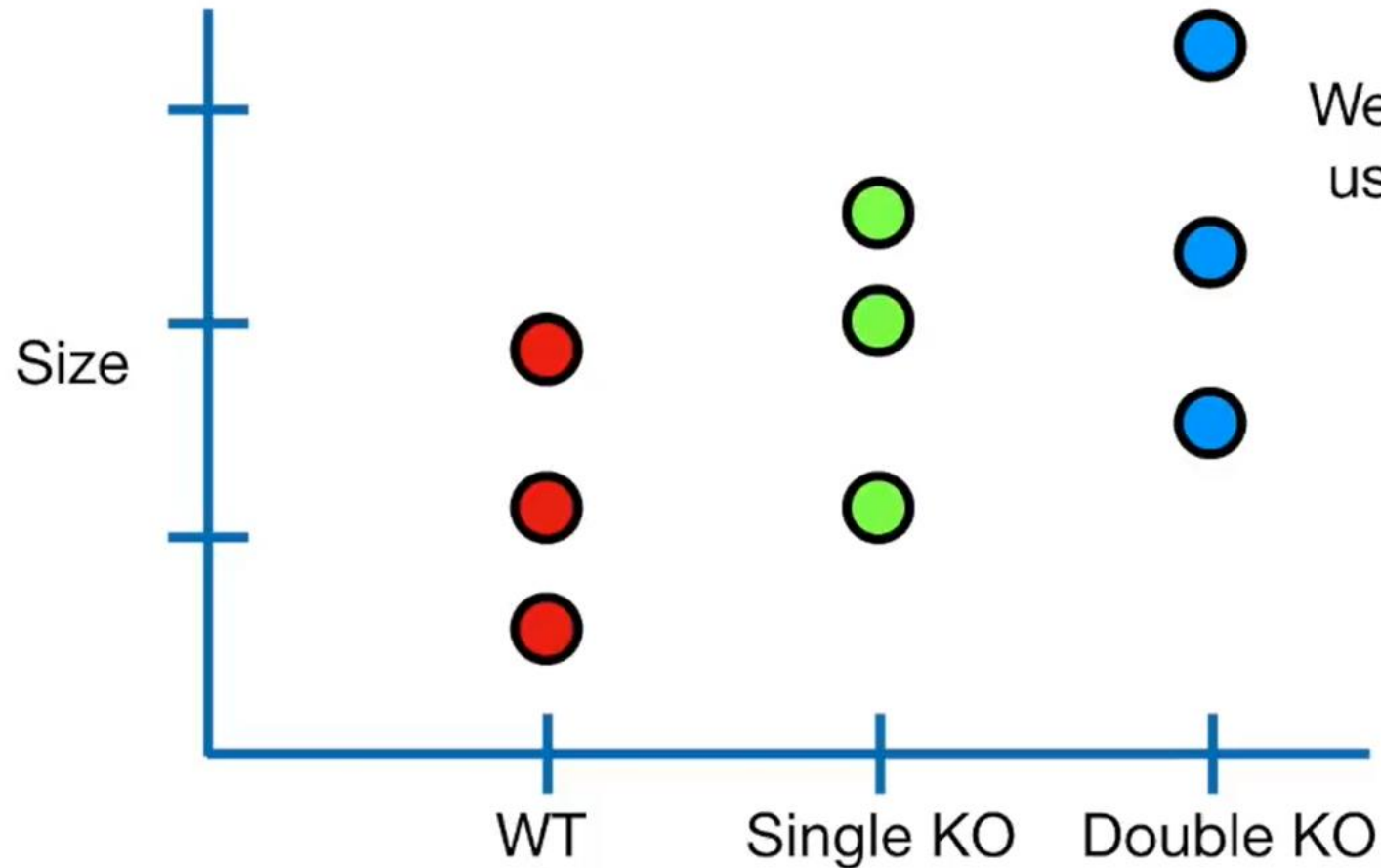


→ SIMPLE REGRESSION VS MULTIPLE REGRESSION

Comparing the simple model to the complicated one tells us if we need to measure **weight** and **blood volume** to accurately predict **size** or if we get away with just **weight**.



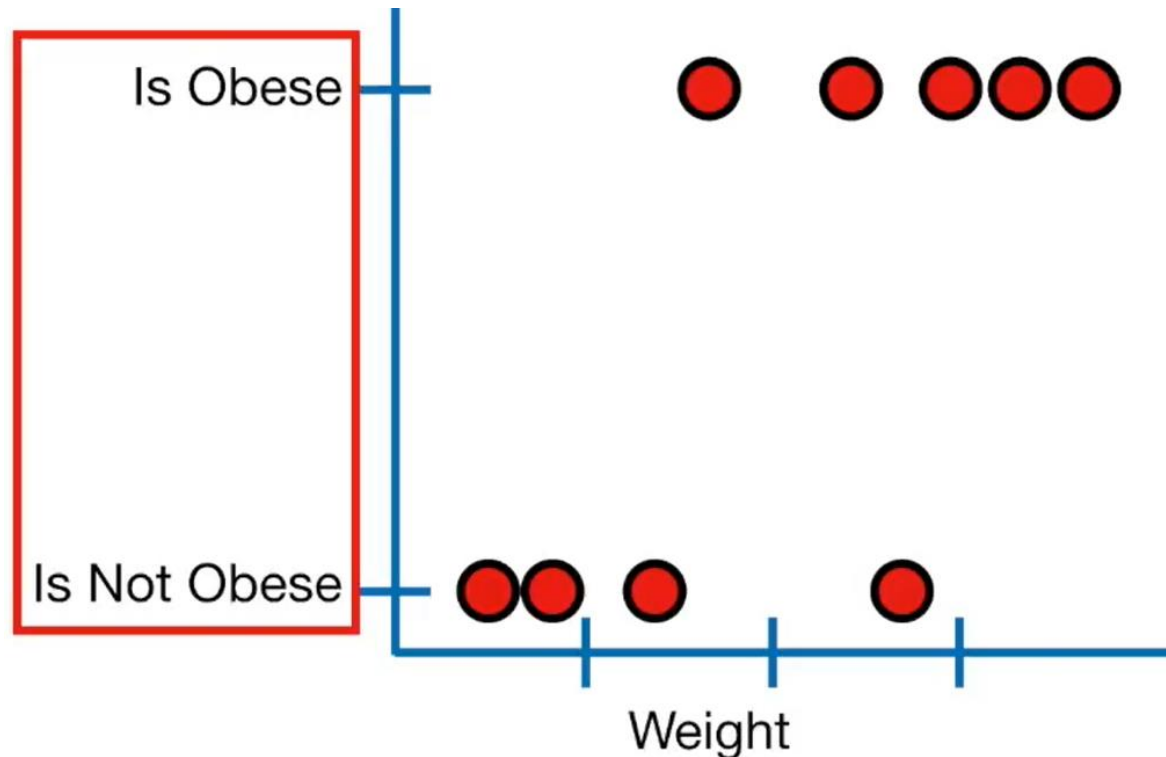
→ REGRESSION BASED ON CATEGORICAL DATA



We also talked about how we can use discrete measurements, like ***genotype***, to predict ***size***.

→ LOGISTIC REGRESSION

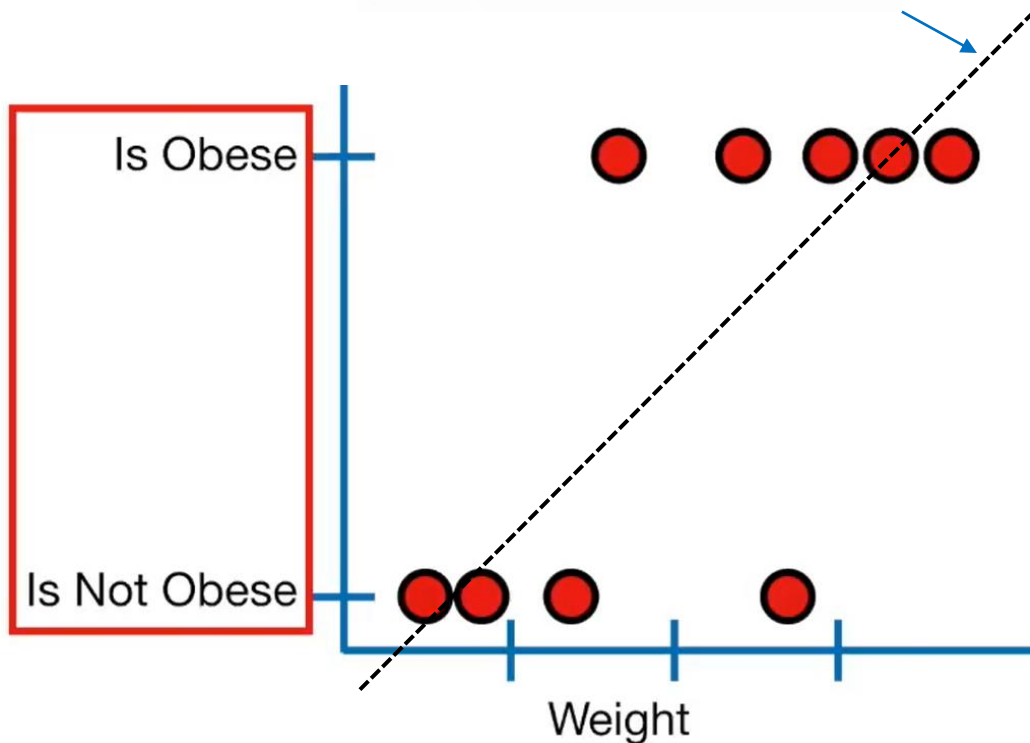
Logistic regression predicts whether something is True or False, instead of predicting something continuous like size.



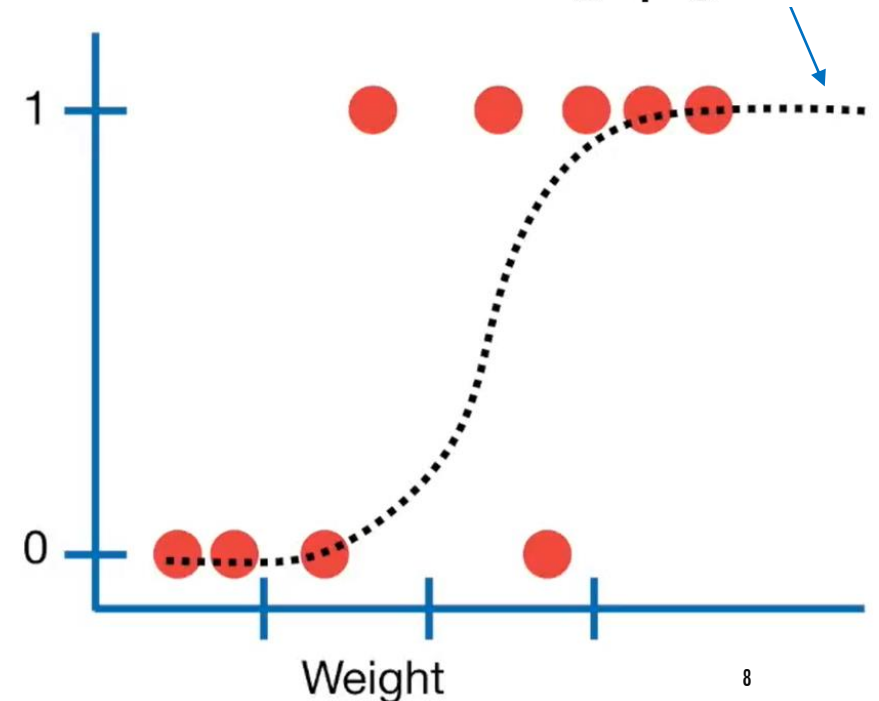
→ LOGISTIC REGRESSION

Instead of fitting a line to the data, logistic regression fits an “S” shaped “logistic function” (Squiggle).

Linear function: $y = m * x + b$

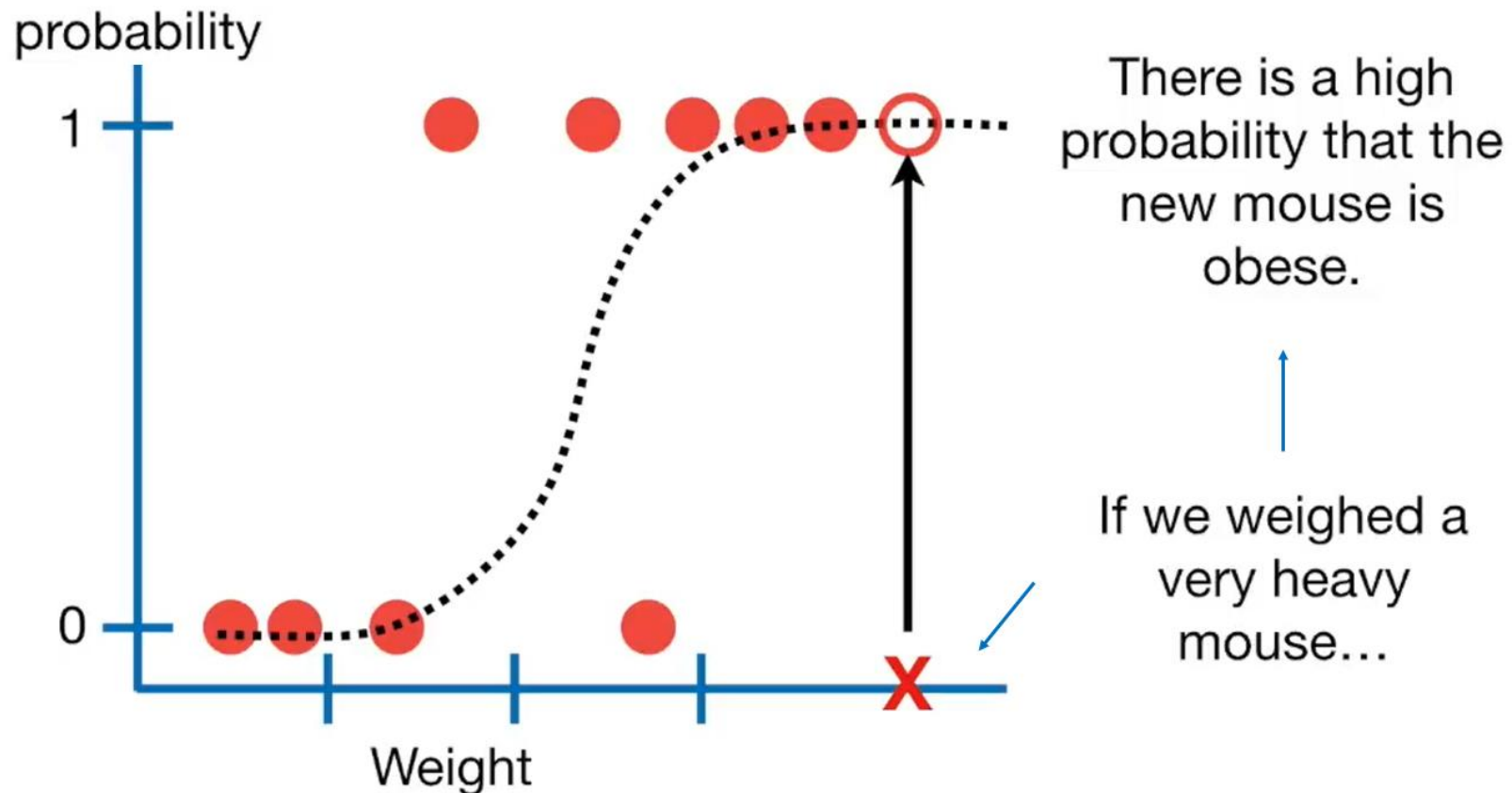


Sigmoid function: $y = \frac{1}{1 + e^{(m*x+b)}}$



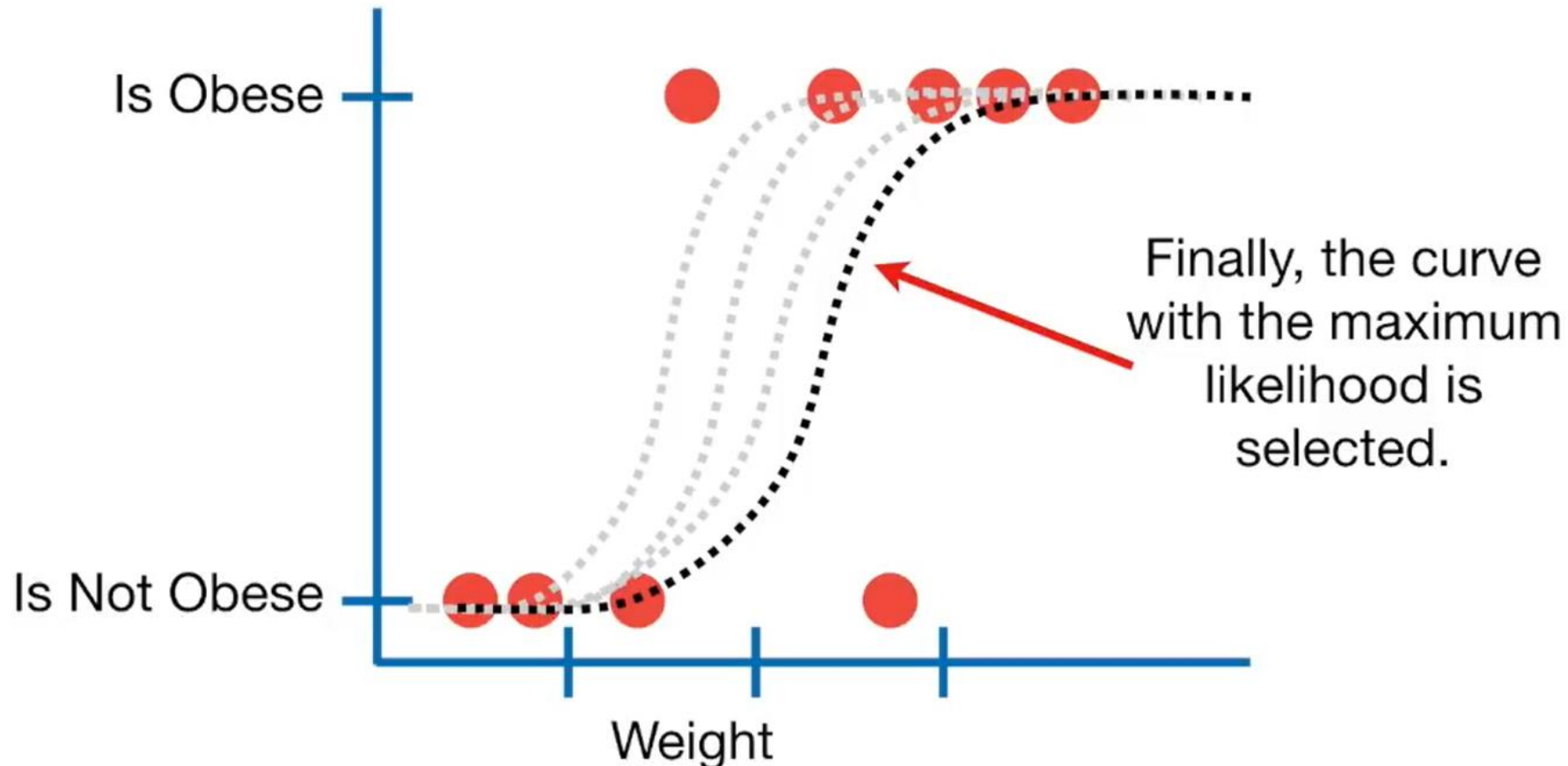
→ LOGISTIC REGRESSION

Probability of a mouse to be obese function depending on its weight



→ LOGISTIC REGRESSION

- Logistic regression doesn't have the same concept of a "residual", so it can't use least squares and it can't calculate R^2 . Instead, it uses something called "Likelihood".



→ LOGISTIC REGRESSION

- Using simple models, Logistic regression can be used to predict obesity from weight.
- In more complicated models, Logistic Regression can work with continuous data (like weight and age) and categorical data (like genotype and Gender).

Obesity is predicted by $\text{Weight} + \text{Genotype} + \text{Age} + \text{Gender}$

- To see if a variable's effect on the prediction is significantly different from 0 we use Wald's test. If not, the variable is eliminated from the model.

→ COST FUNCTION OF LOGISTIC REGRESSION

- Logistic regression uses the likelihood function instead of least squares.
- The cost function is the negative log-likelihood (Log Loss):

$$\text{Cost} = - [y \log(p) + (1 - y) \log(1 - p)]$$

- The cost function penalizes wrong predictions heavily as probabilities approach 0 or 1.
- Optimization: minimize this cost to improve model accuracy.

➔ COST FUNCTION (CROSS-ENTROPY)

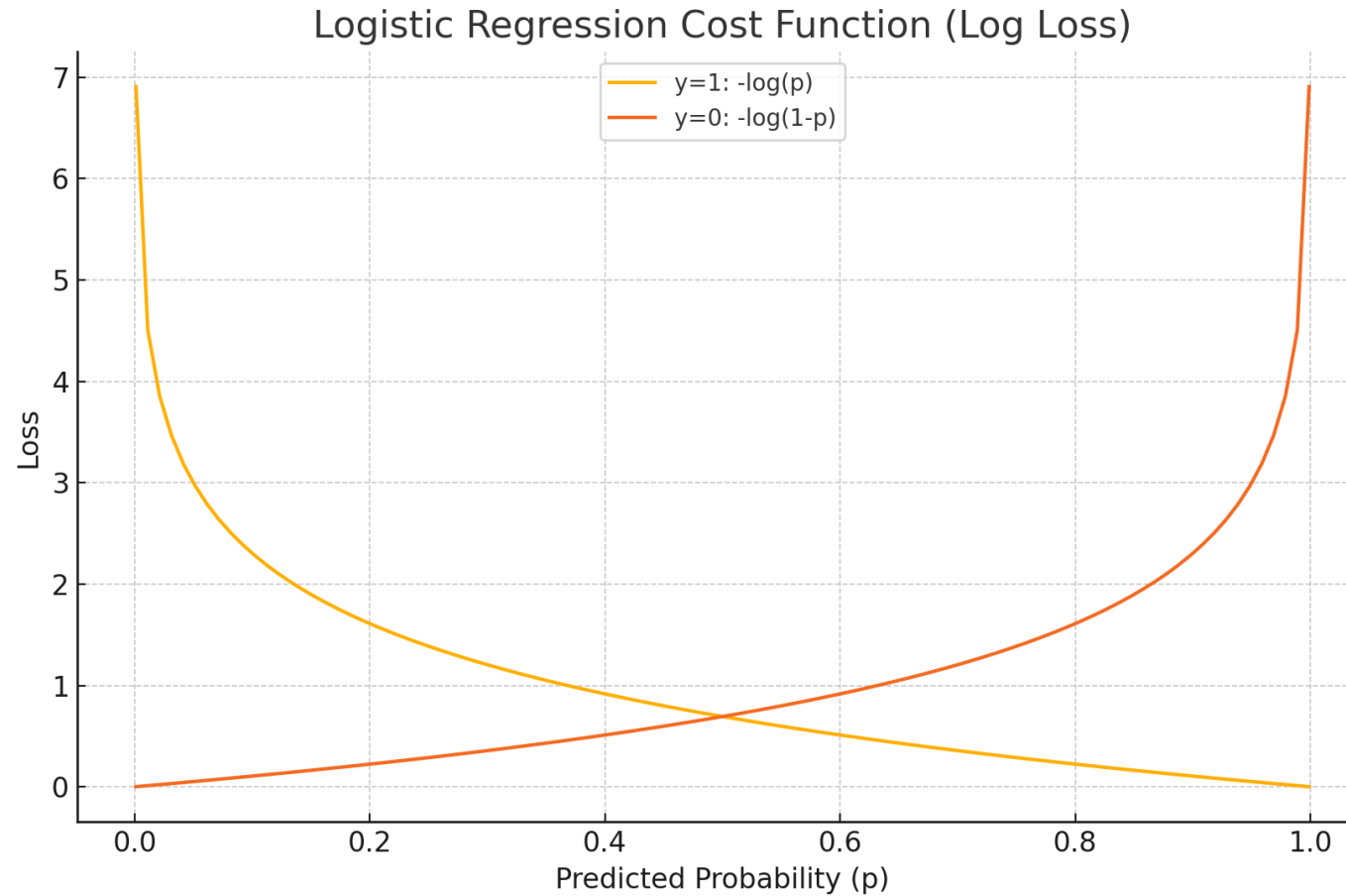
$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right]$$

- m : Total number of training examples.
- $x^{(i)}$: Input features of the i -th training example.
- $y^{(i)}$: Actual label (0 or 1) for the i -th example.
- θ : Vector of model parameters (coefficients).
- $h_{\theta}(x^{(i)})$: Predicted probability that $y = 1$, given by the **sigmoid function**:

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

- $\log(h_{\theta}(x^{(i)}))$: Penalizes wrong predictions when the true label is 1.
- $\log(1 - h_{\theta}(x^{(i)}))$: Penalizes wrong predictions when the true label is 0.
- **Negative sign**: Ensures the cost is positive, since logs of numbers between 0 and 1 are negative.

➔ COST FUNCTION (CROSS-ENTROPY)



The cost function penalizes wrong predictions heavily as probabilities approach 0 or 1.

→ REGULARIZATION IN LOGISTIC REGRESSION

- Since minimizing cost can lead to a model that fits the training data too well but performs poorly on new data → Regularization term to control model complexity.
- Regularization helps prevent overfitting by penalizing large coefficients.

"Don't trust any single feature too much."

$$J = \text{Log Loss} + \lambda \cdot \text{Penalty}$$

- Now the model must:
 - Minimize Error + Complexity
 - Find a balance between fit and simplicity
- In Scikit-Learn the parameter is C ($C = 1/\lambda$).
 - Small C → Strong regularization.
 - Large C → Weak regularization.

➔ COST FUNCTION WITH L1 REGULARIZATION

L1 Regularization (Lasso):

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{m} \sum_{j=1}^n |\theta_j|$$

- $|\theta_j|$: Absolute value of coefficients.
- **Effect:** Forces some coefficients exactly to zero → leads to **feature selection**.

L1 (Lasso)

- Removes some features entirely
- Creates simpler model
- Good when many irrelevant features exist

➔ COST FUNCTION WITH L2 REGULARIZATION

L2 Regularization (Ridge):

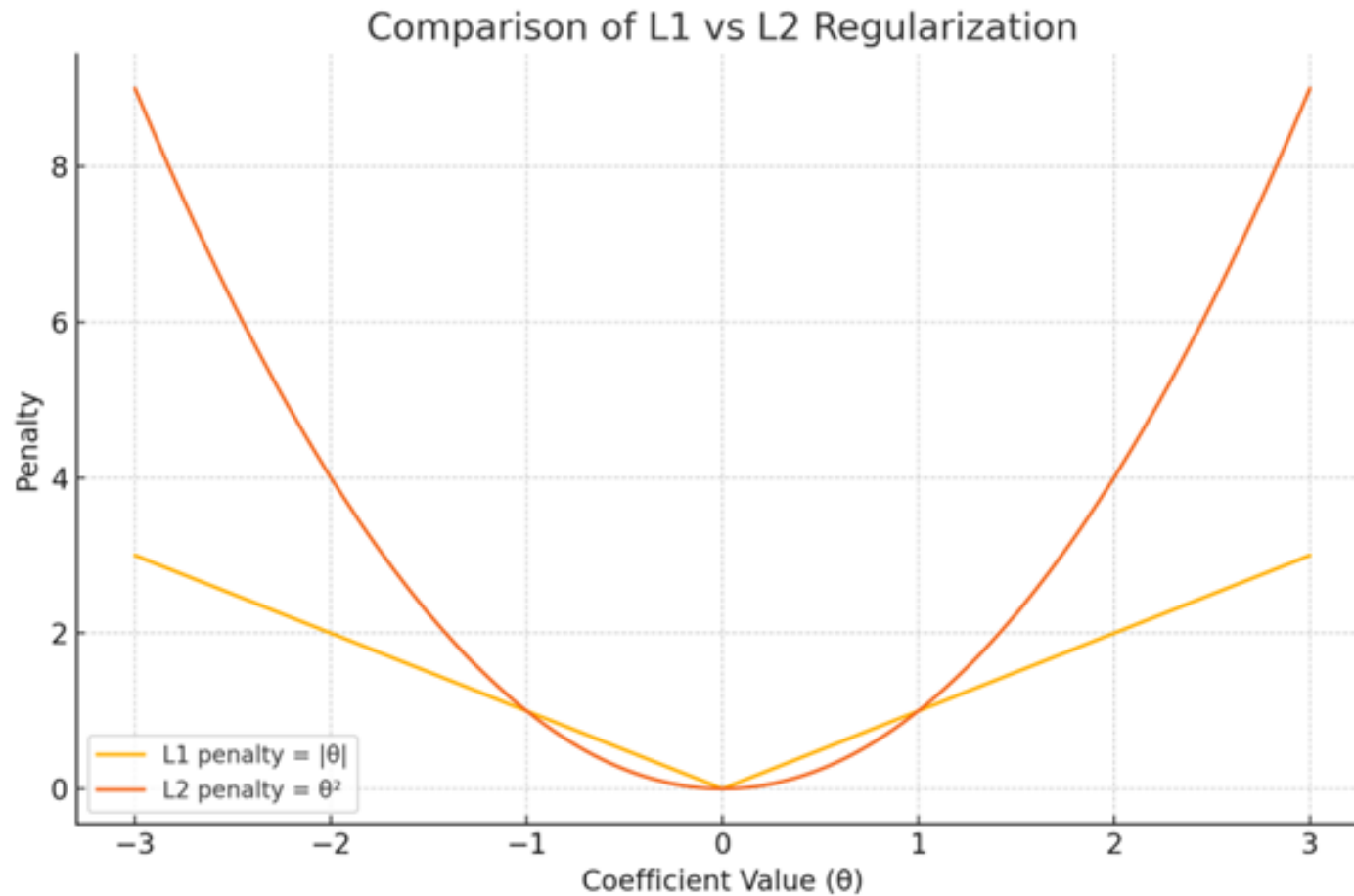
$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

- θ_j^2 : Square of each coefficient. Large values are penalized heavily.
- $\sum_{j=1}^n \theta_j^2$: Sum of squared coefficients
- **Effect:** Shrinks coefficients toward zero smoothly, discouraging complexity.

L2 (Ridge)

- Shrinks all coefficients smoothly
- Keeps all features
- Very stable

→ L1 VS L2 REGULARIZATION



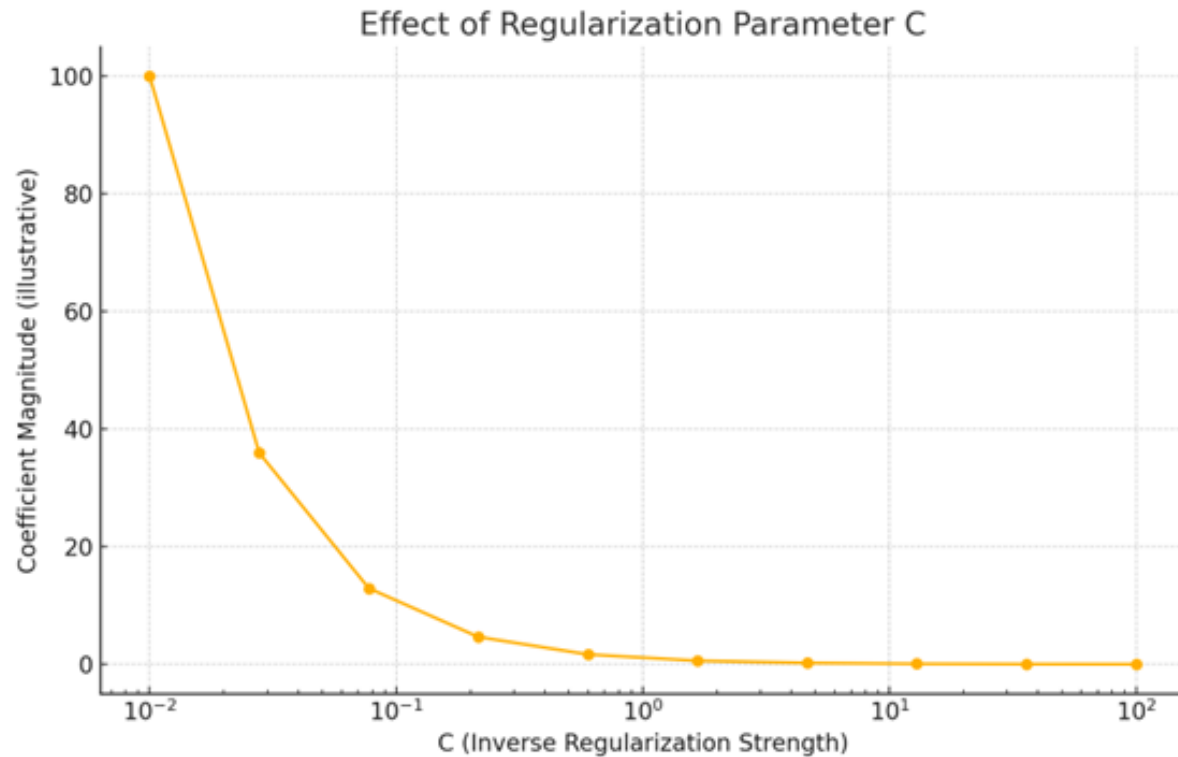
L_1 leads to sparse models
(coefficients driven to zero).

L_2 shrinks coefficients
smoothly.

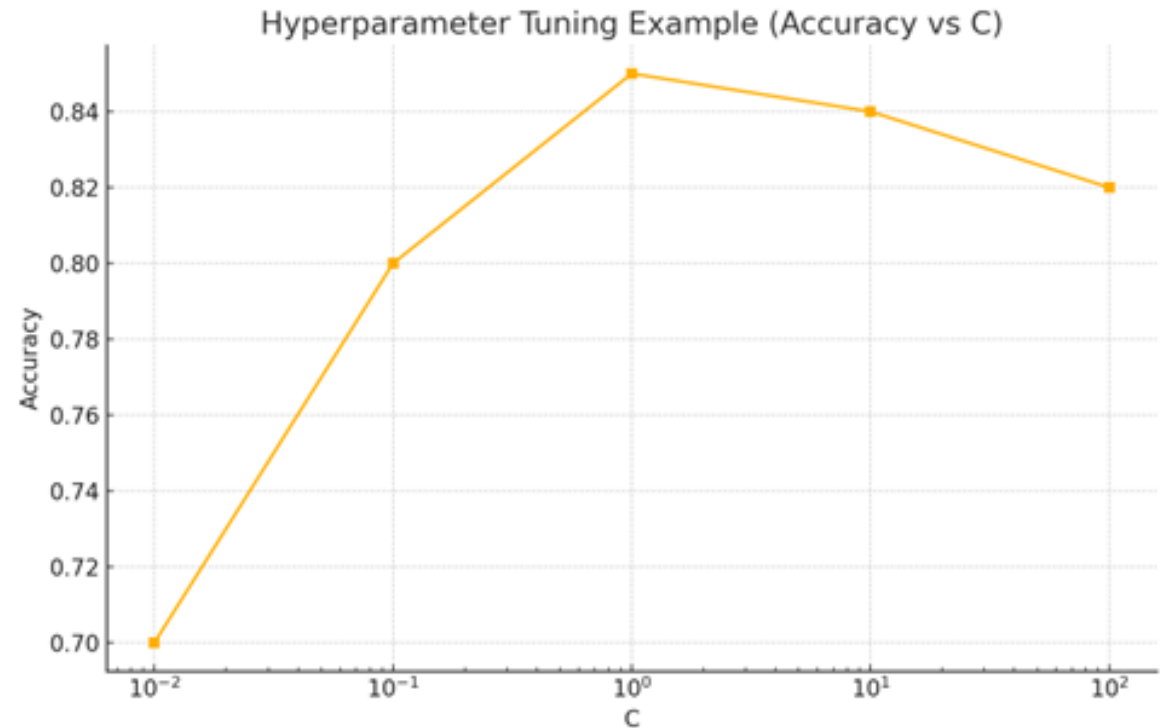
→ HYPERPARAMETER TUNING

- Important hyperparameters:
 - Penalty type (L_1 , L_2 , elasticnet, none)
 - C (regularization strength)
 - Solver (liblinear, lbfgs, saga)
 - max_iter (maximum number of iterations)
- Hyperparameter tuning methods:
 - Grid Search (try combinations of values)
 - Random Search (sample random combinations)
 - Cross-validation for performance estimation

➔ REGULARIZATION PARAMETER C



Smaller C means stronger regularization, shrinking coefficients more aggressively.



Tuning C affects model accuracy. Cross-validation helps choose the best trade-off.

→ RECAP

- Cost Function: Logistic regression minimizes Log Loss (negative log-likelihood).
- Regularization:
 - L_1 encourages sparsity (feature selection).
 - L_2 shrinks coefficients smoothly (prevents overfitting).
- Hyperparameters:
 - Penalty (L_1 , L_2 , elasticnet, none)
 - C (controls regularization strength)
 - Solver (optimization algorithm)
- Hyperparameter tuning with cross-validation improves model performance.

➔ LOGISTIC REGRESSION (PYTHON)

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)    # Splitting

model = LogisticRegression(max_iter=200)          # Initialize logistic regression model

model.fit(X_train, y_train)                        # Fit model

y_pred = model.predict(X_test)                    # Make predictions

from sklearn.metrics import accuracy_score        # Evaluate accuracy

print("Accuracy:", accuracy_score(y_test, y_pred))
```

➔ PRINTING COST FUNCTION

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss

# Train logistic regression model
model = LogisticRegression(max_iter=200)
model.fit(X_train, y_train)

# Predict probabilities
y_prob = model.predict_proba(X_test)

# Compute log loss (cost function)
loss = log_loss(y_test, y_prob)
print('Log Loss:', loss)
```

➔ HYPERPARAMETER TUNING

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
```

```
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['L1', 'L2', 'elasticnet'],
    'solver': ['liblinear', 'saga', 'lbfgs']
}
```

```
grid = GridSearchCV(LogisticRegression(max_iter=500), param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)
```

```
print('Best Parameters:', grid.best_params_)
print('Best Score:', grid.best_score_)
```