

UNIT -1

BY: Dr. Gaurav Srivastav

Outline

- Types of Machine Learning: Supervised, Unsupervised, Reinforcement.
- Linear Regression: Hypothesis Function, Weights and Features, Finding Best Fit, Gradient Descent -Batch, Stochastic & Minibatch Algorithm, Learning Rate.

Learning

Learning, in a general sense, is the process of acquiring new understanding, knowledge, behaviors, skills, values, attitudes, and preferences. It's about changing, growing, and developing through experience, study, or being taught.

Machine Learning

- The term "machine learning" was coined by Arthur Samuel in 1959. His widely cited definition is: "Field of study that gives computers the ability to learn without being explicitly programmed."
- A more formal and technical definition was provided by Tom Mitchell in his 1997 book, *Machine Learning*. He defines it as: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ."

Supervised Machine Learning

Definition:

In supervised ML, we provide:

- **Inputs (features)** \rightarrow measurements, characteristics, etc.
- **Outputs (labels)** \rightarrow correct answers for each input.

The model learns the mapping $f: X \rightarrow y$ from labeled data.

Example

In the Iris Dataset

- **Features (X):**
 - Sepal length (cm)
 - Sepal width (cm)
 - Petal length (cm)
 - Petal width (cm)
- **Labels (y):**
 - 0 = *Iris-setosa*
 - 1 = *Iris-versicolor*
 - 2 = *Iris-virginica*

iris setosa



petal sepal

iris versicolor



petal sepal

iris virginica



petal sepal

This is a **classification problem** (predict a discrete label).

Example (Contd...)

2. Steps for a Supervised ML Pipeline

1. **Load dataset**
2. **Split** into training and test sets
3. **Choose a model** (e.g., Logistic Regression, Decision Tree, etc.)
4. **Train** on training data
5. **Predict** on test data
6. **Evaluate performance**

Understanding Unsupervised ML

- **No labels provided** → model must find patterns or groupings on its own.
- Goal: Discover hidden structure in the data.
- Common tasks: **Clustering** (grouping similar items), **Dimensionality Reduction** (feature compression).

Example

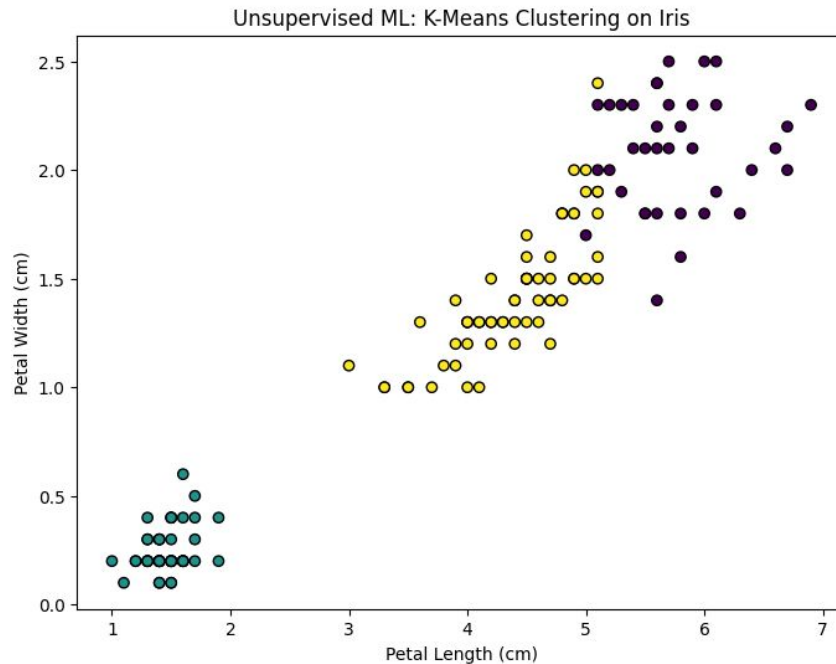
Dataset : Iris

Expected Outcome

- The scatter plot shows **3 colors** → each represents a discovered cluster.
- The algorithm grouped data points based purely on **feature similarity**, without knowing their actual species.
- Cluster centers are the mean feature values for each cluster.

Key Take-ways

- The algorithm doesn't know *Setosa*, *Versicolor*, or *Virginica*.
- It groups them into clusters **based only on patterns** in sepal/petal size.
- Sometimes clusters align well with actual classes; sometimes they don't (overlap possible).



Reinforcement Learning

Understanding Reinforcement Learning

- **Agent** interacts with an **Environment**.
- At each step:
 1. Agent observes **state**.
 2. Chooses an **action**.
 3. Environment returns a **reward** and **next state**.
- Goal: Learn a **policy** (strategy) to maximize cumulative reward.

Example

<http://www.cswiercz.info/assets/img/cart-pole.gif>

Task: Balance a pole on a moving cart.

- **State:** position, velocity, pole angle, pole velocity.
- **Actions:** move cart left or right.
- **Reward:** +1 for every step pole stays balanced.

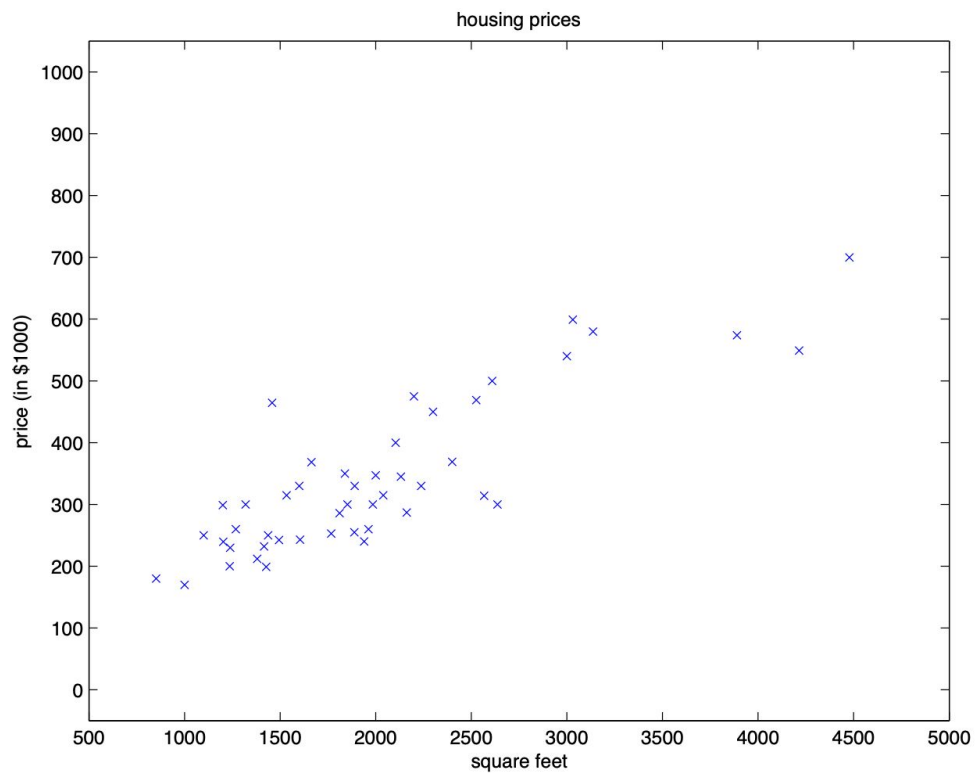
Regression

Regression is a type of supervised machine learning used to model and predict a **continuous numerical output** based on one or more input variables (features).

Key Idea

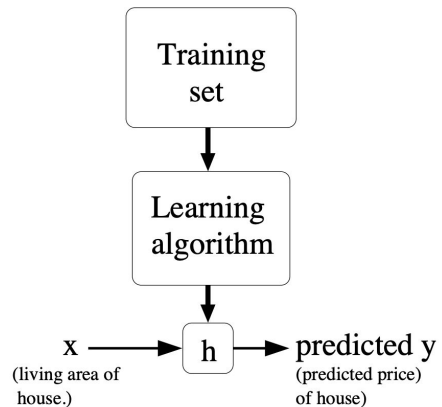
- Find a relationship between **independent variables** (inputs X) and a **dependent variable** (output y).
- The model learns a mathematical function $f(X)$ that best fits the data.

| Living area (feet ²) | Price (1000\$) |
|----------------------------------|----------------|
| 2104 | 400 |
| 1600 | 330 |
| 2400 | 369 |
| 1416 | 232 |
| 3000 | 540 |
| ⋮ | ⋮ |



Hypothesis

When the target variable that we're trying to predict is continuous, such as in our housing example, we call the learning problem a regression problem. When y can take on only a small number of discrete values (such as if, given the living area, we wanted to predict if a dwelling is a house or an apartment, say), we call it a classification problem.



Linear Regression

Linear Regression is a fundamental supervised machine learning algorithm used for predictive analysis. It aims to model the relationship between a dependent variable and one or more independent variables by fitting a linear equation to the observed data.

Linear Regression : $h_{\theta}(x) = \theta_0 + \theta_1 x$

Features (x): Input variables (e.g., house size, number of rooms).

Weights (θ): Parameters that represent the influence of each feature on the output.

| Living area (feet ²) | #bedrooms | Price (1000\$s) |
|----------------------------------|-----------|-----------------|
| 2104 | 3 | 400 |
| 1600 | 3 | 330 |
| 2400 | 3 | 369 |
| 1416 | 2 | 232 |
| 3000 | 4 | 540 |
| \vdots | \vdots | \vdots |

Linear Regression

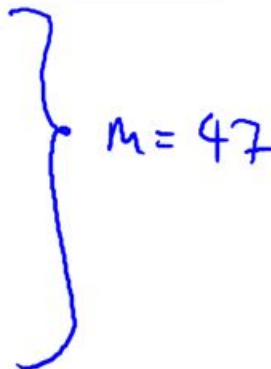
Here, the θ_i 's are the parameters (also called weights) parameterizing the space of linear functions mapping from X to Y . When there is no risk of confusion, we will drop the θ subscript in $h_{\theta}(x)$, and write it more simply as $h(x)$. To simplify our notation, we also introduce the convention of letting $x_0 = 1$ (this is the intercept term), so that

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x,$$

where on the right-hand side above we are viewing θ and x both as vectors, and here n is the number of input variables (not counting x_0).

Training Set

| Size in feet ² (x) | Price (\$) in 1000's (y) |
|-------------------------------|--------------------------|
| 2104 | 460 |
| 1416 | 232 |
| 1534 | 315 |
| 852 | 178 |
| ... | ... |

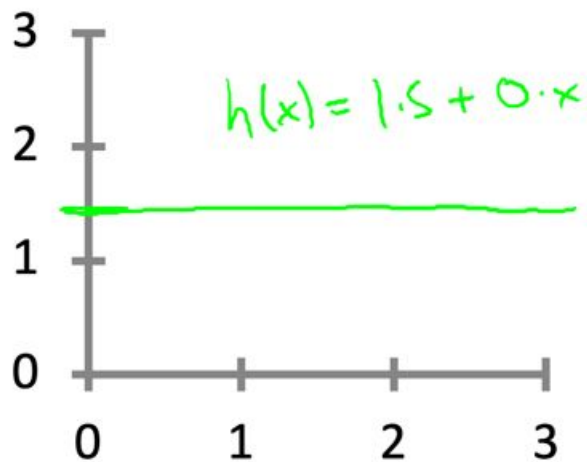
 $m = 47$

Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

θ_i 's: Parameters

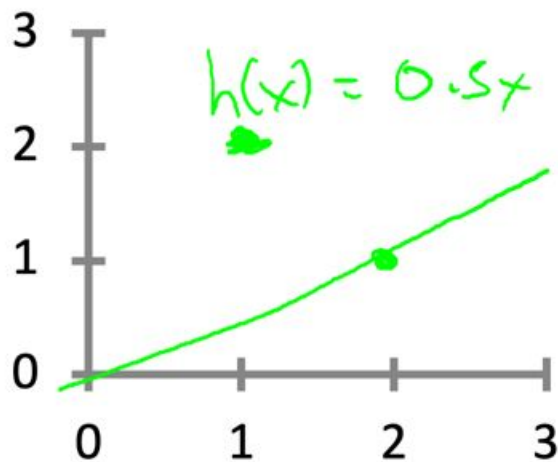
How to choose θ_i 's ?

$$\underline{h_{\theta}(x)} = \theta_0 + \theta_1 x$$



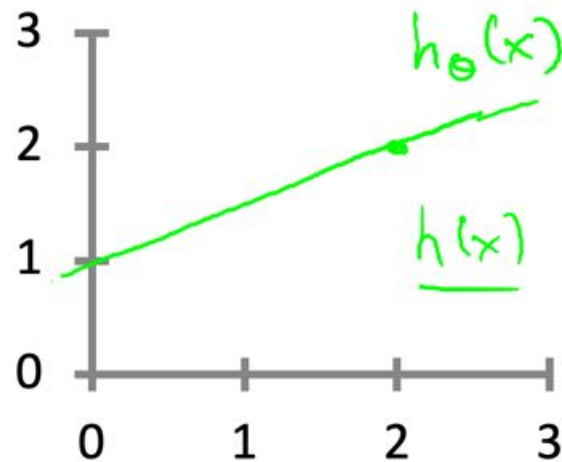
$$\rightarrow \theta_0 = 1.5$$

$$\rightarrow \theta_1 = 0$$



$$\rightarrow \theta_0 = 0$$

$$\rightarrow \theta_1 = 0.5$$



$$\rightarrow \theta_0 = 1$$

$$\rightarrow \theta_1 = 0.5$$

Cost Function: MSE

Now, given a training set, how do we pick, or learn, the parameters θ ? One reasonable method seems to be to make $h(x)$ close to y , at least for the training examples we have. To formalize this, we will define a function that measures, for each value of the θ 's, how close the $h(x^{(i)})$'s are to the corresponding $y^{(i)}$'s. We define the cost function:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2.$$

Cost Function

If you've seen linear regression before, you may recognize this as the familiar least-squares cost function that gives rise to the ordinary least squares regression model. Whether or not you have seen it previously, let's keep going, and we'll eventually show this to be a special case of a much broader family of algorithms.

Gradient Descent

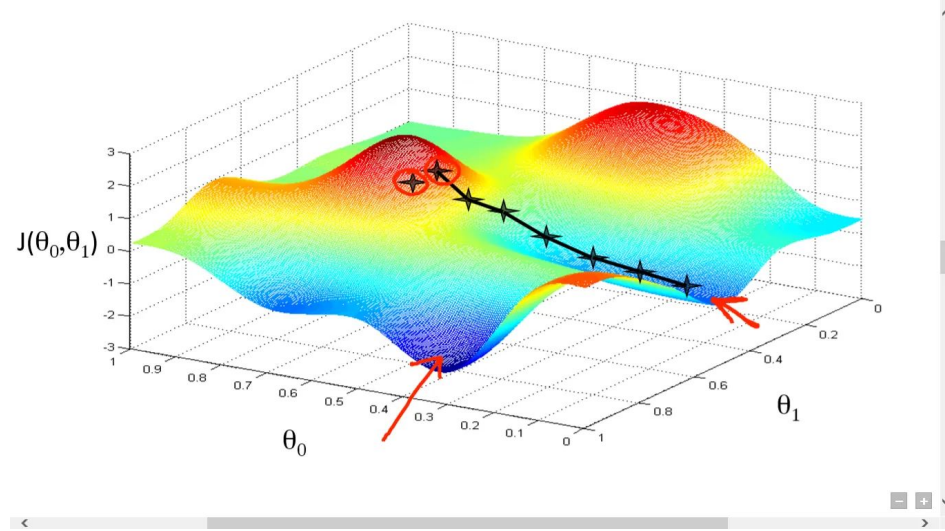
We want to choose θ so as to minimize $J(\theta)$. To do so, let's use a search algorithm that starts with some “initial guess” for θ , and that repeatedly changes θ to make $J(\theta)$ smaller, until hopefully we converge to a value of θ that minimizes $J(\theta)$.

Specifically, let's consider the gradient descent algorithm, which starts with some initial θ , and repeatedly performs the update:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$$

Gradient Descent

This update is simultaneously performed for all values of $j = 0, \dots, n$.) Here, α is called the learning rate. This is a very natural algorithm that repeatedly takes a step in the direction of steepest decrease of J . In order to implement this algorithm, we have to work out what is the partial derivative term on the right hand side. Let's first work it out for the case of if we have only one training example (x, y) , so that we can neglect the sum in the definition of J . We have:



$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^n \theta_i x_i - y \right) \\
&= (h_\theta(x) - y) x_j
\end{aligned}$$

Types of Gradient Descent

Batch Gradient Descent

- Uses **all training examples** to update weights each step.
- Stable but slow for large datasets.

Stochastic Gradient Descent (SGD)

- Updates weights after **each training example**.
- Faster, but noisy updates (fluctuating).

Mini-Batch Gradient Descent

- Updates after a **small batch** of examples (e.g., 32, 64).
- Balance between speed & stability → most popular in practice.

What is Polynomial Regression?

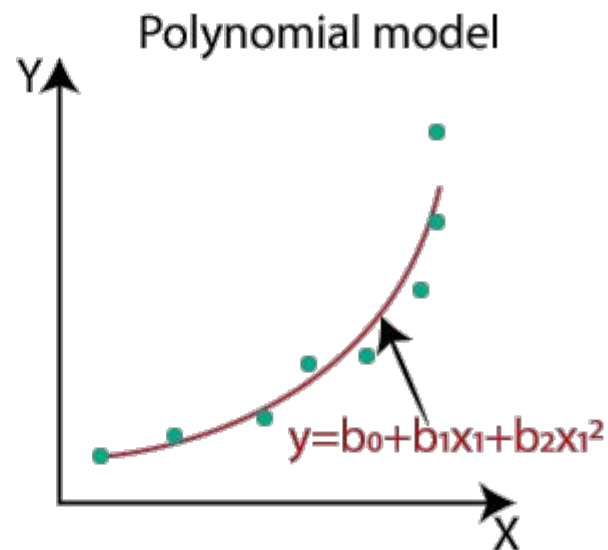
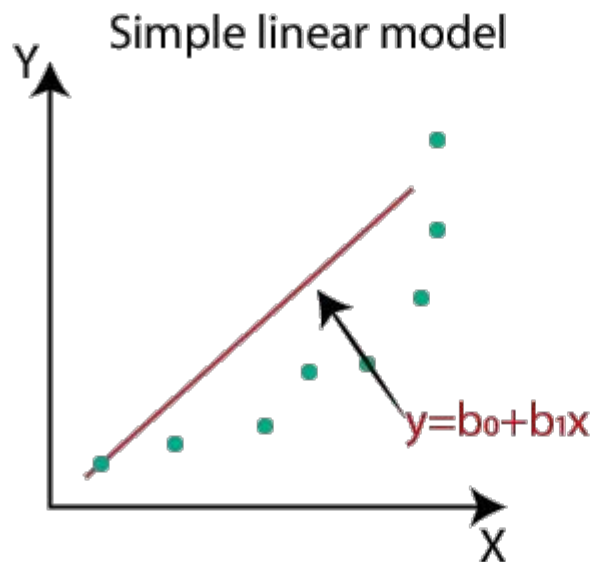
- **Polynomial Regression** is a form of regression analysis in which the relationship between the independent variables and dependent variables are modeled in the n th degree polynomial.
- **Polynomial Regression** models are usually fit with the method of least squares. *The least square method minimizes the variance of the coefficients, under the [Gauss Markov Theorem](#).*
- Polynomial Regression is a special case of Linear Regression where we fit the **polynomial equation** on the data with a curvilinear relationship between the dependent and independent variables.

Types of Polynomials

Linear ————— $ax + b = 0$

Quadratic ————— $ax^2 + bx + c = 0$

Cubic ————— $ax^3 + bx^2 + cx + d = 0$



Bias, Variance, and Error

$$E[(y - \hat{y})^2] = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

Bias means the average difference between the predicted value (expected prediction of the model) and the actual/true value. Bias can be **positive** (model consistently predicts higher) or **negative** (model consistently predicts lower). If we just take **Bias**, then positive and negative biases might cancel each other out, making the error look artificially small.

To avoid this, we square the bias:

1. Squaring makes it always positive, so underestimation and overestimation both count as error.
2. It reflects the true *magnitude* of systematic error, not just its direction.

Example

Suppose the actual value = 10

- Model A always predicts 8 \rightarrow Bias = $(8 - 10) = -2$
- Model B always predicts 12 \rightarrow Bias = $(12 - 10) = +2$

If we just average Bias:

Average Bias = $(-2 + 2) / 2 = 0$

This suggests the model is perfect, which is misleading.

If we use **Bias²**:

- Model A $\rightarrow (-2)^2 = 4$
- Model B $\rightarrow (+2)^2 = 4$

Now both errors are correctly captured.

Variance

- **Definition:** Variance measures how much the model's predictions change if we train it on *different datasets* drawn from the same population.
- High variance means the model is very sensitive to training data. A small change in training data → very different predictions.
- This usually happens in **overfitting**, where the model learns noise along with the signal.

Intuition: Variance = instability in predictions.

- Suppose you train a polynomial regression model of degree 10.
- On one dataset, it gives predictions close to actual values.
- On another dataset, predictions are very different (because it learned the noise).
- This wide fluctuation = **High Variance**.

Irreducible Error (i.e., Noise)

- **Definition:** This is the part of the error that **no model can ever remove**, because it comes from the inherent randomness or noise in the data.
- Even the perfect model cannot explain it.
- Causes:
 - Measurement errors (faulty sensors, human entry errors).
 - Randomness in the process (e.g., predicting a person's future weight—there's always natural variation).

Intuition: Irreducible Error = background noise you can't get rid of.

$$\text{Total Error} = \underbrace{\text{Bias}^2}_{\text{wrong assumptions}} + \underbrace{\text{Variance}}_{\text{over-sensitivity}} + \underbrace{\text{Irreducible Error}}_{\text{pure noise}}$$

Example

- **Bias²:** Using a linear model when the real relationship is curved.
- **Variance:** Using a very complex model that fits training houses perfectly but fails on new ones.
- **Irreducible Error:** House price depends on sudden political decisions, personal buyer emotions, or unexpected market shifts → no model can predict this 100%.

Bias Variance Tradeoff

In machine learning, the bias-variance tradeoff describes how model performance and generalization are affected by bias and variance.

What are Bias and Variance?

- Bias: When a model is too simple and fails to learn the patterns in data well, it has high bias. High bias leads to underfitting; the model performs poorly on both training and new data.
- Variance: When a model is too complex and learns the training data—including noise—too well, it has high variance. High variance leads to overfitting; the model performs well on training data but poorly on new (unseen) data.

Bias-Variance Tradeoff

The tradeoff means finding a balance so that the model is not too simple (high bias, underfitting) and not too complex (high variance, overfitting).

- High Bias: Simple model, high error due to missed patterns (underfitting)
- High Variance: Complex model, high error on new data (overfitting)
- Tradeoff: Find the “sweet spot” in model complexity so both bias and variance are low.

How to Control Bias-Variance Tradeoff?

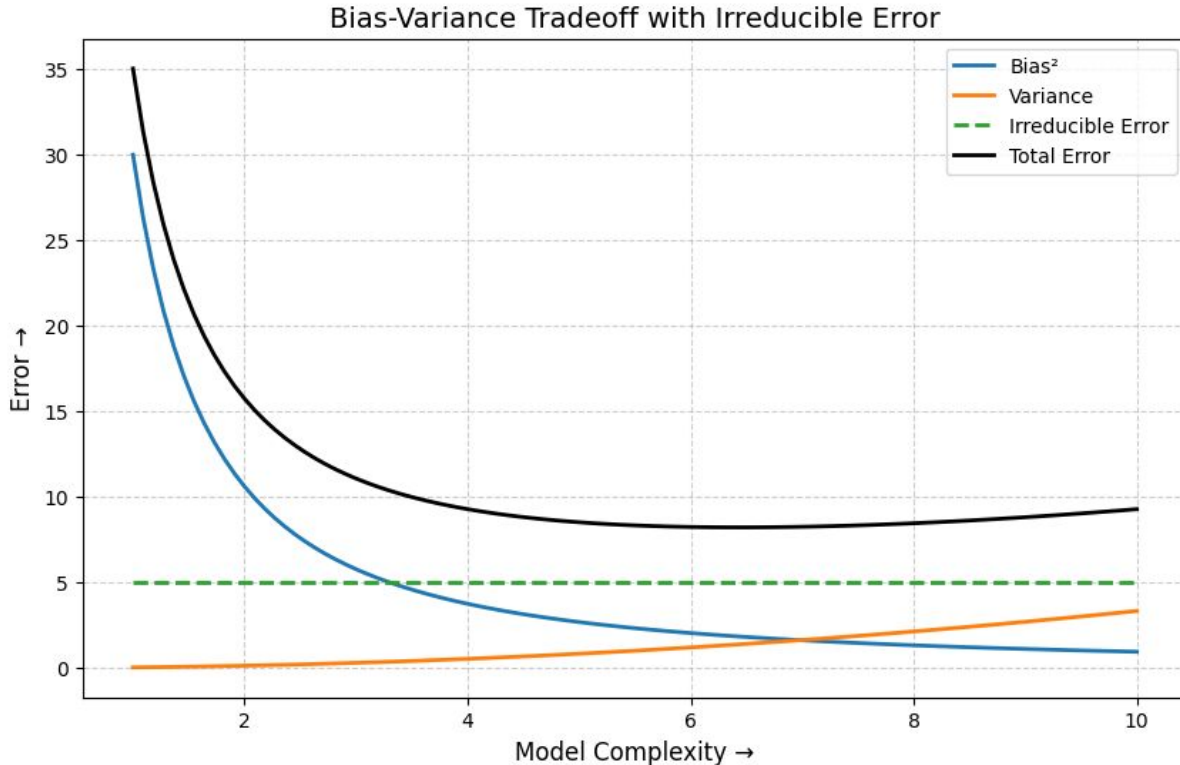
- Choose and train models carefully
- Use hyperparameter tuning
- Apply regularization techniques (like Dropout, Batch Normalization)
- Use cross-validation
- Collect and use more training data

Overfitting and Underfitting

- Underfitting: Like a student who studied too little and cannot answer any exam questions.
- Overfitting: Like a student who memorized last year's questions only and cannot answer new questions in the exam.

Bias Variance Tradeoff

- **Bias²** decreases as model complexity increases (simple models underfit).
- **Variance** increases with complexity (complex models overfit).
- **Irreducible Error** stays constant (noise in data, cannot be reduced).
- **Total Error** is the sum of all three, showing the balance point where error is minimum.



Mean Squared Error (MSE)

Definition: Average of the squared differences between actual values and predicted values.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Penalizes **large errors more** because of squaring.

Root Mean Squared Error (RMSE)

The square root of the Mean Squared Error.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

Easier to interpret than MSE because it's in the same units as the original data. RMSE penalizes bigger errors, but is commonly reported because it gives a direct sense of prediction error average.

Mean Absolute Error (MAE)

The average of the absolute differences between prediction and actual values.

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

R² Score

Measures how much variance in the dependent variable is explained by the model.

$$R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

- R² ranges between $-\infty$ to 1.
 - R²=1: Perfect model
 - R²=0: Model is no better than the mean
 - R²<0: Model is worse than predicting the mean

Example

Suppose we have **3 data points**:

$$y = [3, 5, 7], \quad \hat{y} = [2.5, 5.5, 6.0]$$

Step 1: Errors

$$(y - \hat{y}) = [0.5, -0.5, 1.0]$$

Step 2: MSE

$$MSE = \frac{(0.5^2 + (-0.5)^2 + 1.0^2)}{3} = \frac{0.25 + 0.25 + 1}{3} = \frac{1.5}{3} = 0.5$$

Step 3: RMSE

$$RMSE = \sqrt{0.5} \approx 0.707$$

Step 4: MAE

$$MAE = \frac{(|0.5| + |-0.5| + |1.0|)}{3} = \frac{2.0}{3} \approx 0.667$$

Example

Step 5: R² Score

- Mean of y :

$$\bar{y} = \frac{3 + 5 + 7}{3} = 5$$

- Total variance (denominator):

$$\sum (y_i - \bar{y})^2 = (3 - 5)^2 + (5 - 5)^2 + (7 - 5)^2 = 4 + 0 + 4 = 8$$

- Residual variance (numerator):

$$\sum (y_i - \hat{y}_i)^2 = 0.25 + 0.25 + 1 = 1.5$$

- R²:

$$R^2 = 1 - \frac{1.5}{8} = 1 - 0.1875 = 0.8125$$

Logistic Regression

Logistic Regression is a classification algorithm used to predict the probability that a given input x belongs to a particular category (usually binary: 0 or 1)

Classification

Email: Spam / Not Spam?

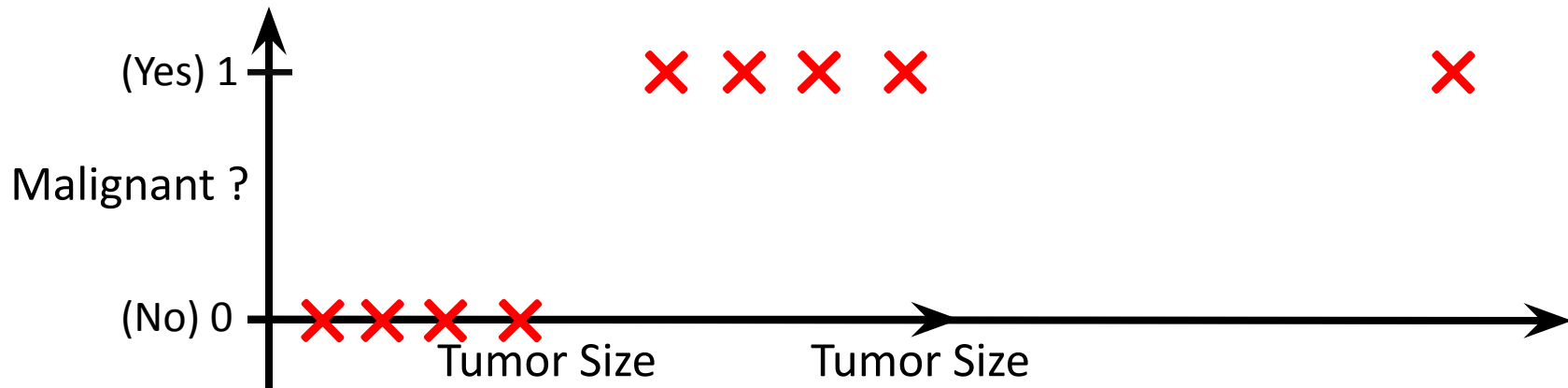
Online Transactions: Fraudulent (Yes / No)?

Tumor: Malignant / Benign ?

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

0: “Negative Class” (e.g., benign tumor)

1: “Positive Class” (e.g., malignant tumor)



Threshold classifier output $h_{\theta}(x)$ at 0.5:

If $h_{\theta}(x) \geq 0.5$, predict "y = 1"

If $h_{\theta}(x) < 0.5$, predict "y = 0"

Classification: $y = 0$ or 1

$h_{\theta}(x)$ can be > 1 or < 0

Logistic Regression: $0 \leq h_{\theta}(x) \leq 1$

Logistic Regression

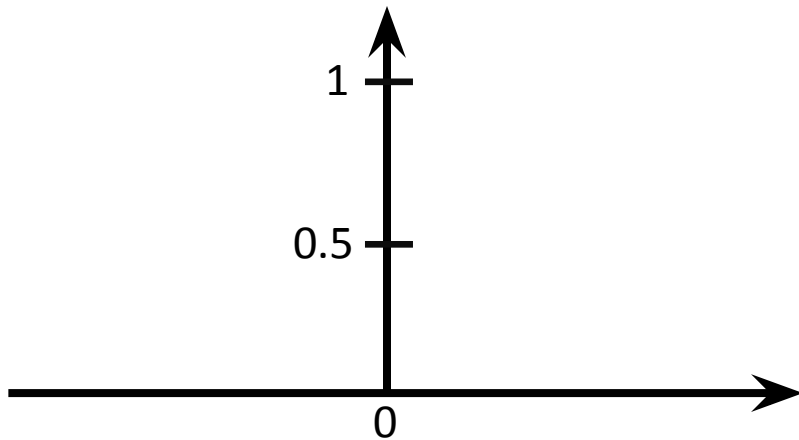
Hypothesis
Representation

Logistic Regression Model

Want $0 \leq h_{\theta}(x) \leq 1$

$$h_{\theta}(x) = \theta^T x$$

Sigmoid function
Logistic function



Interpretation of Hypothesis Output

$h_{\theta}(x)$ = estimated probability that $y = 1$ on input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$
 $h_{\theta}(x) = 0.7$

Tell patient that 70% chance of tumor being malignant

“probability that $y = 1$, given x ,
parameterized by θ ”

$$P(y = 0|x; \theta) + P(y = 1|x; \theta) = 1$$
$$P(y = 0|x; \theta) = 1 - P(y = 1|x; \theta)$$

Logistic Regression

Decision boundary

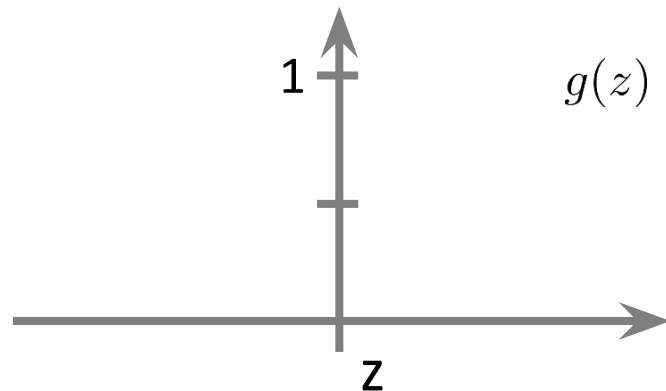
Logistic regression

$$h_{\theta}(x) = g(\theta^T x)$$

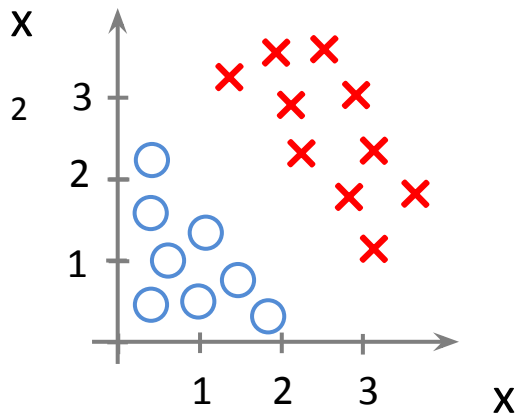
$$g(z) = \frac{1}{1+e^{-z}}$$

Suppose predict “ $y = 1$ ” if $h_{\theta}(x) \geq 0.5$

predict “ $y = 0$ ” if $h_{\theta}(x) < 0.5$



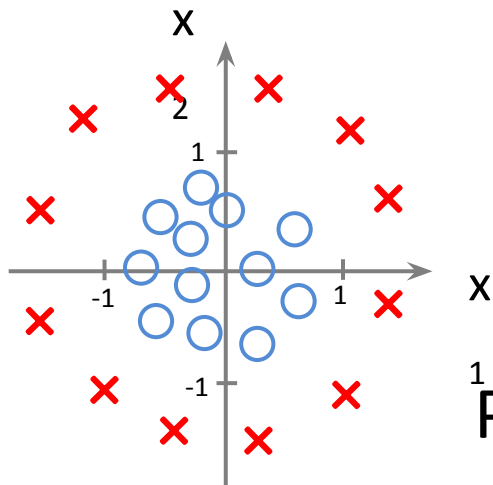
Decision Boundary



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

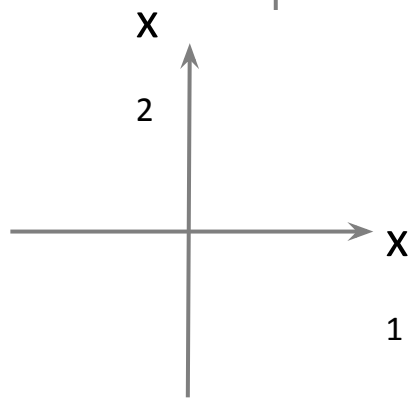
Predict “ $y = 1$ ” if $-3 + x_1 + x_2 \geq 0$

Non-linear decision boundaries



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

Predict “ $y = 1$ ” if $-1 + x_1^2 + x_2^2 \geq 0$



$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$$

Logistic Regression

Cost function

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

m examples $x \in \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$

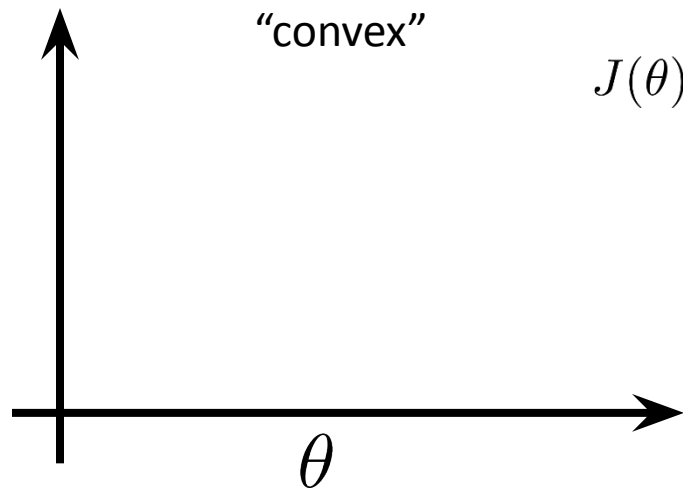
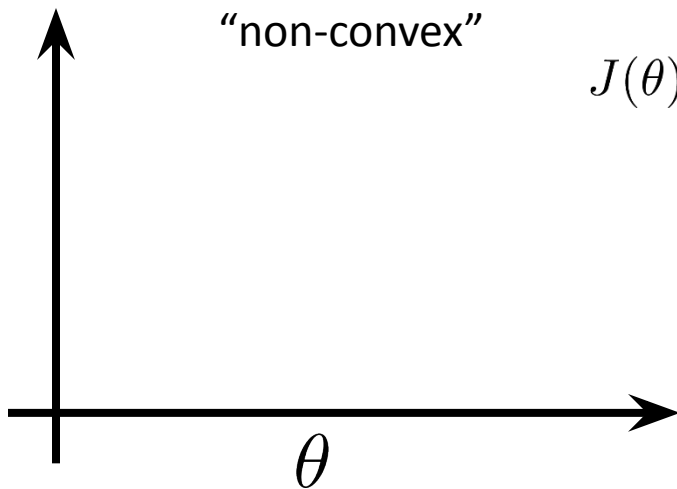
$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

How to choose parameters θ ?

Cost function

Linear regression: $J(\theta) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (h_{\theta}(x^{(i)}) - y^{(i)})^2$

$$\text{Cost}(h_{\theta}(x^{(i)}), y^{(i)}) = \frac{1}{2} (h_{\theta}(x^{(i)}) - y^{(i)})^2$$



Why Not Use MSE for Classification?

For classification (especially binary), we want $y \in \{0,1\}$. If you use MSE, the outputs of $h\theta(x)$ from a linear model aren't guaranteed to be bounded between 0 and 1. It can also result in non-convex cost landscapes for the sigmoid function, complicating optimization.

Logistic regression cost function

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

$$\text{Cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always

For all m training examples, the overall **cost function (J)** is defined as the average:

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

| Student | Exam Score x | Pass (Label, y) |
|---------|----------------|--------------------|
| 1 | 40 | 0 |
| 2 | 60 | 1 |

$$\theta_0 = -8, \theta_1 = 0.15$$

Step 1: Hypothesis Function

$$h_{\theta}(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$

Step 2: Calculate Predicted Probabilities

- Student 1:

$$h_{\theta}(40) = \frac{1}{1 + e^{-(-8 + 0.15 \times 40)}} = \frac{1}{1 + e^2} \approx \frac{1}{1 + 7.389} = 0.119$$

- Student 2:

$$h_{\theta}(60) = \frac{1}{1 + e^{-(-8 + 0.15 \times 60)}} = \frac{1}{1 + e^1} \approx \frac{1}{1 + 2.718} = 0.269$$

Step 3: Cost Function for Each Example

Cost function formula:

$$\text{Cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)), & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)), & \text{if } y = 0 \end{cases}$$

Step 3: Cost Function for Each Example

Cost function formula:

$$\text{Cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)), & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)), & \text{if } y = 0 \end{cases}$$

- Student 1 (Actual Label $y=0$):

$$\text{Cost}_1 = -\log(1 - 0.119) = -\log(0.881) = 0.127$$

- Student 2 (Actual Label $y=1$):

$$\text{Cost}_2 = -\log(0.269) = 1.313$$

Step 4: Overall Cost (Average)

$$J(\theta) = \frac{1}{2}(0.127 + 1.313) = 0.72$$

Intuition

- If the true label is $y=1$ the cost is low only when $h_{\theta}(x)$ is close to 1.
- If the true label is $y=0$, the cost is low only when $h_{\theta}(x)$ is close to 0.
- The function penalizes model predictions that are confidently wrong (pushed towards infinity) much more than less confident mistakes.

Logistic regression cost function

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

To fit parameters θ :

$$\min_{\theta} J(\theta)$$

To make a prediction given new x :

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

Gradient Descent

Want $\min_{\theta} J(\theta)$:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

(simultaneously update all θ_j)

Gradient Descent

Want $\min_{\theta} J(\theta)$:

Repeat {

$$\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

} (simultaneously update all θ_j)

Algorithm looks identical to linear regression!

Logistic
Regression

Advanced
optimization

Optimization algorithm

Cost function $J(\theta)$. Want $\min_{\theta} J(\theta)$.

Given θ , we have code that can compute

$$\begin{aligned} & - J(\theta) \\ & - \frac{\partial}{\partial \theta_j} J(\theta) \quad (\text{for } j = 0, 1, \dots, n) \end{aligned}$$

Gradient descent:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

Optimization algorithm

Given θ , we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$ (for $j = 0, 1, \dots, n$)

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

- No need to manually pick α
- Often faster than gradient descent.

Disadvantages:

- More complex

Logistic Regression

Multi-class classification:
One-vs-all

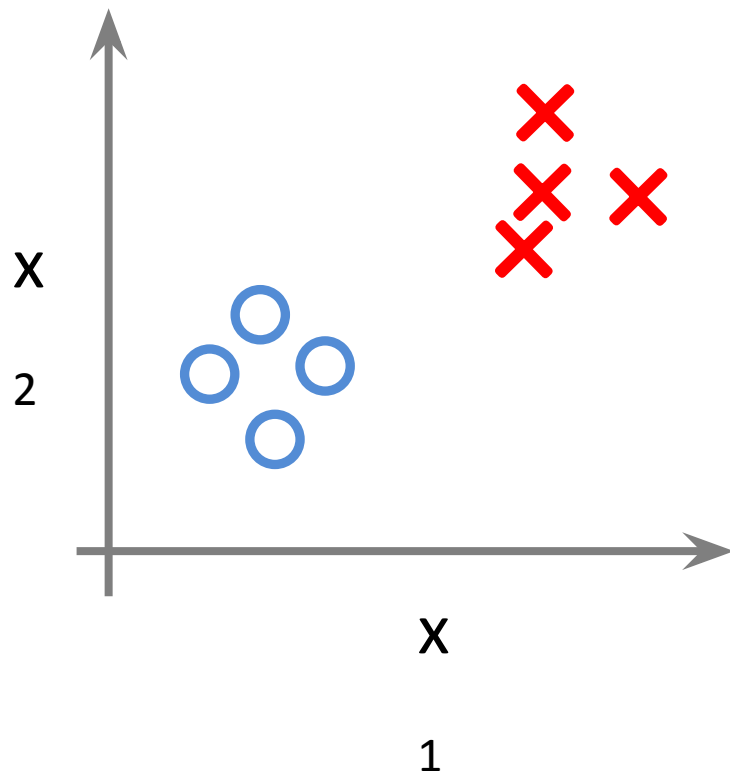
Multiclass classification

Email foldering/tagging: Work, Friends, Family, Hobby

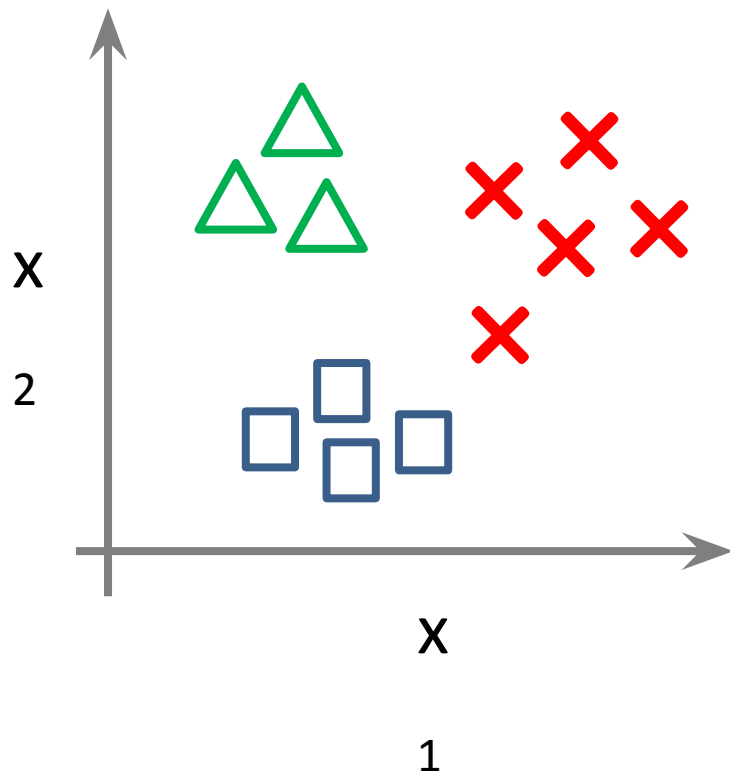
Medical diagrams: Not ill, Cold, Flu

Weather: Sunny, Cloudy, Rain, Snow

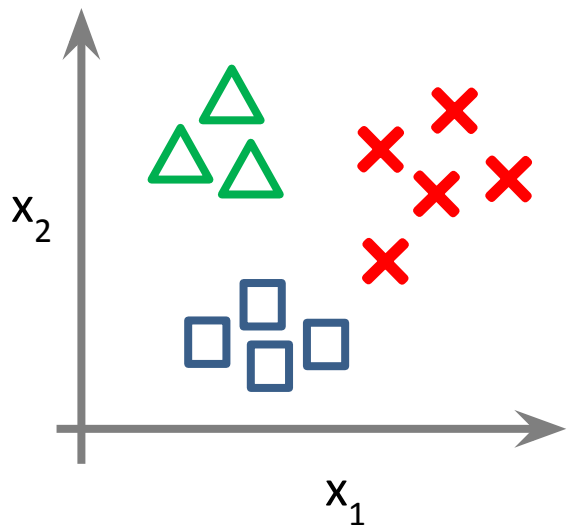
Binary classification:




Multi-class classification:



One-vs-all (one-vs-rest):

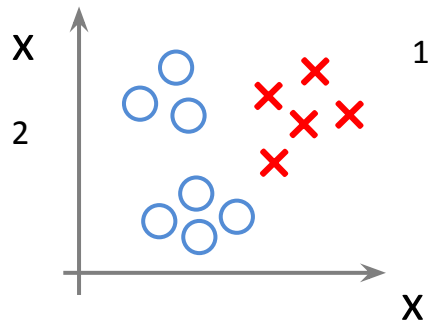
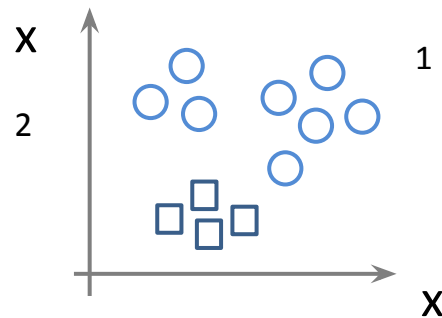
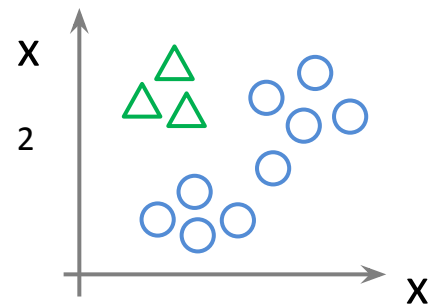


Class 1: 

Class 2: 

Class 3: 

$$h_{\theta}^{(i)}(x) = P(y = i|x; \theta) \quad (i = 1, 2, 3)$$



One-vs-all

Train a logistic regression classifier $h_{\theta}^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x , to make a prediction, pick the class i that maximizes

$$\max_i h_{\theta}^{(i)}(x)$$

Confusion Matrix

We assume **binary classification**:

- Class **1** → **Positive**
(Yes/True)
- Class **0** → **Negative**
(No/False)

| | | Predicted Class | |
|--------------|---|-----------------|-----------------|
| | | 1 | 0 |
| Actual Class | 1 | TP 11 | FN 10 |
| | 0 | FP 01 | TN 00 |

| Actual / Predicted | Predicted = 1 (Positive) | Predicted = 0 (Negative) |
|-----------------------|---|---|
| Actual = 1 (Positive) | TP (True Positive) Correctly predicted positive “11” | FN (False Negative) Missed positive “10” |
| Actual = 0 (Negative) | FP (False Positive) Wrongly predicted positive “01” | TN (True Negative) Correctly predicted negative “00” |

Key Metrics

The confusion matrix is used to calculate several performance metrics:

- **Accuracy:** The proportion of total predictions that were correct. It's the most intuitive metric but can be misleading with imbalanced datasets.

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

- **Precision (Positive Predictive Value):** The proportion of positive predictions that were actually correct. It answers the question: "Of all the positive predictions, how many were actually positive?"

$$Precision = \frac{TP}{TP + FP}$$

- **Recall (Sensitivity or True Positive Rate):** The proportion of actual positive cases that were correctly identified by the model. It answers the question: "Of all the actual positive cases, how many did the model find?"

$$Recall = \frac{TP}{TP + FN}$$

- **Specificity (True Negative Rate):** The proportion of actual negative cases that were correctly identified. It's the opposite of recall, focusing on the negative class.

$$Specificity = \frac{TN}{TN + FP}$$

- **F-beta Score:** The weighted harmonic mean of precision and recall. It's useful when you need to balance precision and recall. The beta value determines the weight given to recall.

F1 ($\beta=1$) → balances precision & recall equally.

F0.5 → favors precision.

F2 → favors recall.

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

Errors in Confusion Matrix

Two kinds of misclassifications:

1. False Positive (FP, “01”)

- Predicted **Positive (1)** but actually **Negative (0)**.
- Example: Model predicts “Patient has disease” but patient is healthy.
- Error type: **Type I Error** (False Alarm).

2. False Negative (FN, “10”)

- Predicted **Negative (0)** but actually **Positive (1)**.
- Example: Model predicts “Patient is healthy” but patient actually has disease.
- Error type: **Type II Error** (Missed Detection).

$$\text{Errors} = \text{FP} + \text{FN}$$

Connection to Metrics

- **High FP** \rightarrow lowers **precision**.
- **High FN** \rightarrow lowers **recall/sensitivity**.
- **Both FP + FN** \rightarrow lower **accuracy**.

Numerical Example

Suppose we test **100 samples** with the following results:

- **TP = 40** (Predicted positive, actually positive)
- **FP = 10** (Predicted positive, actually negative)
- **FN = 20** (Predicted negative, actually positive)
- **TN = 30** (Predicted negative, actually negative)

Step 1: Accuracy

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} = \frac{40 + 30}{100} = \frac{70}{100} = 0.70$$

Step 2: Precision

$$Precision = \frac{TP}{TP + FP} = \frac{40}{40 + 10} = \frac{40}{50} = 0.80$$

Step 3: Recall (Sensitivity / TPR)

$$Recall = \frac{TP}{TP + FN} = \frac{40}{40 + 20} = \frac{40}{60} = 0.67$$

Step 4: Specificity (TNR)

$$Specificity = \frac{TN}{TN + FP} = \frac{30}{30 + 10} = \frac{30}{40} = 0.75$$

Step 5: F1 Score

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} = 2 \cdot \frac{0.8 \cdot 0.67}{0.8 + 0.67} = 2 \cdot \frac{0.536}{1.47} \approx 0.73$$

Step 6: F2 Score (Recall-oriented)

$$F2 = (1 + 2^2) \cdot \frac{P \cdot R}{(2^2 \cdot P) + R} = 5 \cdot \frac{0.8 \cdot 0.67}{(4 \cdot 0.8) + 0.67} = 5 \cdot \frac{0.536}{3.87} = 2.68/3.87 \approx 0.69$$

Thank You