Linear Models

Q. Derive the least squares solution for linear regression

Q. Explain L1 and L2 regularization. Compare Ridge and Lasso regression

Regularization is a technique used to **prevent overfitting** by **penalizing large weights** in a model.

It **adds a penalty term** to the loss function, which discourages the model from becoming too complex.

Types of Regularization:

---

1. L1 Regularization (Lasso Regression)

- Adds the **absolute value** of the weights to the cost function:

$$J(w) = \text{MSE} + \lambda \sum_{i=1}^{n} |w_i|$$

- λ: regularization parameter
- Encourages **sparsity** → many weights become **exactly zero**

Used for **feature selection** — it can remove irrelevant features entirely.

---

2. L2 Regularization (Ridge Regression)

- Adds the **square of the weights** to the cost function:

$$J(w) = \text{MSE} + \lambda \sum_{i=1}^{n} w_i^2$$

- Penalizes **large weights**, but does **not force them to zero**
- All features are retained, but with **smaller magnitudes**

Helps improve **generalization** by shrinking weights smoothly.

---

3. Elastic Net (Bonus Knowledge)

- Combines **L1 and L2** regularization:

$$J(w) = \text{MSE} + \lambda_1 \sum |w_i| + \lambda_2 \sum w_i^2$$

Useful when data is **high-dimensional and correlated**.

Comparison: Ridge vs Lasso

Feature	Ridge Regression (L2)	Lasso Regression (L1)
Effect on Weights	Shrinks weights	Can shrink some weights to **zero**
Feature Selection	No	Yes
Use Case	When **all features are useful**	When **only few features matter**
Solution	Has a closed-form	Requires optimization

Visual Analogy:

- **L2 penalty** creates **circular contours** → smooth weight reduction
- **L1 penalty** creates **diamond contours** → sharp edges lead to zero weights

---

Example Scenario:

Suppose you have 100 features, but only 10 are truly relevant:

- **Use Lasso (L1):** It will eliminate the 90 irrelevant ones.
- **Use Ridge (L2):** It will keep all 100, but shrink them down.

Q. How is linear regression used for classification

While **linear regression** is primarily used for **predicting continuous values**, it can also be **adapted for classification**, especially **binary classification**.

Idea: We use **linear regression to compute a continuous output**, then apply a **threshold** to convert it into a **class label**.

Example Scenario:

Classify whether a student passes or fails based on hours of study.

- Input: x = hours studied
- Output: y ∈ {0,1} → 0 = Fail, 1 = Pass

---

1. Linear Regression Formula:

$$\hat{y} = w_0 + w_1 x$$

Where:

- y^: predicted output (can be any real number)
- w0: bias term
- w1: weight

---

2. Thresholding for Classification:

To convert the continuous output into a class:

$$\text{Class} = \begin{cases} 1 & \text{if } \hat{y} \geq 0.5 \\ 0 & \text{if } \hat{y} < 0.5 \end{cases}$$

This makes it behave like a binary classifier.

Limitations of Using Linear Regression for Classification:

Issue	Explanation
Unbounded Output	Regression output is not limited to [0,1]
Poor Probabilistic Interpretation	Doesn't model probability correctly (e.g., >1 or <0)
Sensitive to Outliers	Large outliers can distort decision boundary
Inaccurate for Complex Boundaries	Works only for linearly separable classes

---

Better Alternative: Logistic Regression

- Uses **sigmoid function** to squash output to (0,1)
- Interpreted as **probability**

- Trained specifically for classification loss (cross-entropy)

$$P(y=1|x) = \frac{1}{1+e^{-w^T x}}$$

Q. What is the role of the cost function in regression

In **regression**, a **cost function** (also called **loss function**) measures how well the **predicted values** y^\hat{y} from a model **match the actual target values** yy.

It quantifies the **error** or **difference** between predicted and actual values and helps the model **learn by minimizing this error**.

Role of Cost Function:

The cost function:

1. **Guides the Learning Process:**
   It tells the algorithm how far off the predictions are and in **what direction to update the weights**.
2. **Objective to Minimize:**
   In training, we try to find model parameters (like weights ww) that **minimize the cost**.
3. **Controls Convergence:**
   The shape of the cost function affects how fast and well the model **converges during training**.

Most Common Cost Function in Regression: Mean Squared Error (MSE):

$$J(w) = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(y_i - (w^T x_i))^2$$

Where:

- yi: true value
- y^i: predicted value
- m: number of data points
- w: model parameters

The ½ is added for convenience when differentiating.

Graphical Insight:

Imagine a bowl-shaped curve (paraboloid) — the **lowest point** on this curve is where the **cost is minimized**, and that's the **best model**.

Q. Evaluate multiple regression models

Multiple Linear Regression is an extension of simple linear regression where the target variable y depends on **multiple independent variables** x1,x2,...,xn:

$$y = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

Each wi represents the **weight (coefficient)** for the respective feature xi.

Goal of Evaluation: The goal is to **evaluate how well the model fits the data** and how accurately it predicts **unseen data**.

Evaluation Criteria for Multiple Regression Models:

---

1. Mean Squared Error (MSE)

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

- Measures the **average squared difference** between actual and predicted values.
- Lower MSE = better fit.

---

2. Root Mean Squared Error (RMSE)

$$\text{RMSE} = \sqrt{MSE}$$

- Easier to interpret as it's in **original units of y**.
- Preferred when large errors should be heavily penalized.

---

3. Mean Absolute Error (MAE)

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

- Measures average **absolute difference**.
- Less sensitive to outliers compared to MSE.

---

4. Coefficient of Determination R² (R-squared)

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

- Explains the **proportion of variance** in y explained by the model.
- Ranges from 0 to 1:
  - R² = 1: Perfect fit
  - R² = 0: No explanatory power

---

5. Adjusted R²

$$\text{Adjusted } R^2 = 1 - \left(\frac{(1-R^2)(n-1)}{n-p-1}\right)$$

Where:

- n: number of data points
- p: number of predictors

Compensates for **adding irrelevant predictors**.

---

6. Cross-Validation (e.g., k-Fold CV)

- Splits data into k parts.
- Trains on k-1 folds, tests on the remaining fold.
- Repeats k times and takes **average RMSE/MSE**.

Provides **robust estimate** of generalization error.

Comparison Table for Evaluation Metrics:

Metric	Measures	Sensitive to Outliers?	Scaled?
MSE	Squared error	Yes	No
RMSE	Error in original units	Yes	Yes
MAE	Absolute error	Less	Yes
R2R^2	% of variance explained	No	Yes (0–1 scale)

Q. Explain curve fitting using least square method. Derive formula for different line patterns

Curve fitting is the process of finding a mathematical function (curve) that **best fits a set of data points**.

The goal is to find a function f(x) such that the difference between the predicted values f(xi) and actual values yi is minimized.

The most common method used is the **Least Squares Method**, which minimizes the **sum of squared errors** between predicted and actual values.

General Least Squares Principle

Given:

- n data points: (x1,y1),(x2,y2),…,(xn,yn)
- A model (curve) f(x) with parameters to fit

We minimize the **sum of squared errors**:

$$S = \sum_{i=1}^{n} [y_i - f(x_i)]^2$$

Choose parameters in f(x) that **minimize S**.

---

1. Linear Fit: y = a + bx

2. Quadratic Fit: y = a + bx + cx²

3. Exponential Fit: y = ae^{bx}

4. Power Law Fit: y = ax^b

Q. Explain the geometric intuition behind SVM

Support Vector Machine (SVM) is a powerful **supervised learning algorithm** used for **classification** and **regression**, especially known for handling **linearly and non-linearly separable data**.

SVM tries to find the **best boundary (hyperplane)** that **separates classes with the largest possible margin**.

Geometric Intuition of SVM (in 2D)

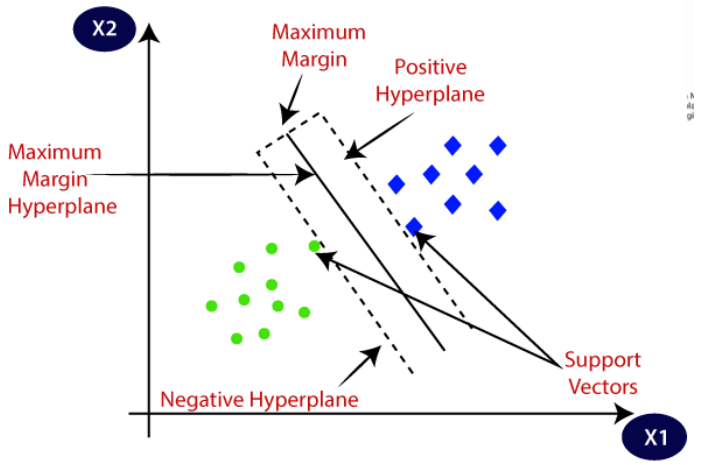
Goal: Find a **decision boundary (line or hyperplane)** that:

- **Separates the classes correctly**
- Has the **maximum margin** between the two classes

Key Terms:

- **Hyperplane:** A flat surface (line in 2D, plane in 3D) that divides the space
- **Margin:** The distance between the hyperplane and the **closest data points** from each class
- **Support Vectors:** The data points that lie **on the edge of the margin** — they "support" the hyperplane

---



What SVM Finds Geometrically:

SVM finds the **optimal separating hyperplane**:

$$w^T x + b = 0$$

Where:

- w is the **weight vector** (normal to the hyperplane)
- b is the **bias** (offset)
- x is the input vector

SVM Optimization Goal:

Maximize the margin:

$$\text{Margin} = \frac{2}{||w||}$$

So the optimization becomes:

$$\min_{w,b} \frac{1}{2}||w||^2 \quad \text{subject to: } y_i(w^T x_i + b) \geq 1$$

This ensures all data points are **on the correct side** of the margin.

SVM's Geometric Strategy:

Step	Action	Geometric Meaning
1	Draw all possible separating lines	Many ways to divide classes
2	Find the one with **largest margin**	Best generalization to new data
3	Focus only on **support vectors**	Other points do **not affect** the boundary

---

What If Data Is Not Linearly Separable?

1. Soft Margin SVM

- Allows **some misclassifications** (adds slack variables)
- Trades off between **margin size and classification errors**

2. Kernel Trick

- Maps data to **higher dimensions** where it **becomes linearly separable**
- Common kernels: Polynomial, RBF (Gaussian)

Q. What is the kernel trick? How does it help in SVM

The **kernel trick** is a method in Support Vector Machines (SVM) that allows the algorithm to **work in high-dimensional feature spaces without explicitly transforming the data**.

It lets SVM **learn non-linear boundaries** by **implicitly mapping input features into a higher-dimensional space** — where the data becomes **linearly separable**.

Why It's Called a "Trick"?

Because we avoid **computing the mapping** ϕ(x) explicitly.

Instead, we compute:

$$K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$$

This is a **kernel function** — it computes the **dot product in the higher-dimensional space** using only the **original input data**.

How It Helps SVM:

- **Without kernel trick:** SVM finds a linear hyperplane in original feature space
- **With kernel trick:** SVM finds a **non-linear decision boundary** in original space by using a **linear hyperplane in transformed space**

You get a **non-linear classifier** with **linear math**!

Why It Matters:

- In original space: classes are **not linearly separable**
- In transformed space: data becomes **linearly separable**
- With the **kernel trick**, SVM can **create curved decision boundaries** using only **linear math**

Visual Example:

Say your data looks like this in 2D:

Class 1 (o): around the origin

Class 2 (x): surrounding like a ring

Original space: No straight line can separate o from x

Feature space: Map $(x_1, x_2) \rightarrow (x_1^2 + x_2^2)$, and now a **linear plane** can separate them!

Common Kernels Used in SVM:

Kernel Type	Formula	Use Case / Notes
Linear	K(x,x′)=x^Tx′	For linearly separable data
Polynomial	K(x,x′)=(x^Tx′+c)^d	Nonlinear relationships
RBF / Gaussian	K(x,x′)=exp(−γ‖x−x′‖2)	Handles complex, wavy boundaries
Sigmoid	tanh(αxTx′+c)	Similar to neural network behavior

Advantages of Kernel Trick in SVM:

Benefit	Explanation
Enables **nonlinear classification**	Without needing neural networks
Works in **very high-dimensional spaces**	Even infinite (like RBF kernel)
Avoids **explicit computation**	Saves time and space — very efficient
Allows **flexibility**	Choose kernels suited to specific data patterns

Q. Explain the concept of hyperplane in SVM

A **hyperplane** is a flat, affine subspace that **divides the feature space into two parts**.

- In **2D**, it's a **line**
- In **3D**, it's a **plane**
- In **n-dimensional space**, it's called a **hyperplane**

In **Support Vector Machines (SVM)**, a **hyperplane is the decision boundary** that separates data points from **different classes**.

Mathematical Equation of a Hyperplane:

For a hyperplane in n-dimensional space:

$$w^T x + b = 0$$

Where:

- w = **weight vector** (normal to the hyperplane)
- x = input vector
- b = **bias (offset)**
- w^T x = dot product between w and x

In SVM:

- The SVM tries to find the **optimal hyperplane** that:
    1. **Maximally separates** the two classes
    2. Has the **maximum margin** from the closest points (support vectors)

Types of Hyperplanes in SVM:

Hyperplane Type	Equation	Meaning
Main Decision Boundary	w^T x + b = 0	Separates the classes
Margin Boundaries	w^T x + b = ±1	Define the width of the margin
Support Vectors	Points lying on margin lines	Critical to defining the hyperplane

---

Key Properties:

1. **Only support vectors matter** — other points don't affect the hyperplane
2. **The margin is**:

$$\text{Margin} = \frac{2}{||w||}$$

3. **Best hyperplane** is the one with **maximum margin**, which improves **generalization**

Q. What is the role of support vectors in SVM? Give an example

In **Support Vector Machine (SVM)**, **support vectors** are the **critical data points** that lie **exactly on the margin boundaries** — they are the **closest points to the decision boundary** from each class.

They **"support" the optimal hyperplane**, meaning:

- The **hyperplane is positioned using only these points**
- All other data points **do not affect the decision boundary**

Key Roles of Support Vectors:

Role	Explanation
1. Define the Margin	Support vectors lie on the lines: wTx+b=±1w^T x + b = \pm 1
2. Anchor the Hyperplane	The optimal separating hyperplane is **entirely determined** by them
3. Robustness	They make the model more stable — only a few important points matter
4. Sparsity	SVM solutions are **sparse** — they depend only on a few support vectors
5. Sensitivity	Moving or removing a support vector **changes the decision boundary**

Q. Explain different types of the SVM kernels

In Support Vector Machines (SVM), a **kernel function** is used to **transform data into a higher-dimensional space**, where a **linear hyperplane** can be used to separate classes — **even if they're not linearly separable in the original space**.

This technique is known as the **kernel trick**, which allows us to **compute the dot product in a higher dimension without explicitly mapping the data**.

Types of SVM Kernels:

---

1. Linear Kernel

$$K(x, x') = x^T x'$$

- **No transformation** is applied.
- Used when the data is already **linearly separable**.

Fast, simple, and works well when the number of features is high (e.g., text classification).

Use case: Spam detection, document classification

---

2. Polynomial Kernel

$$K(x, x') = (x^T x' + c)^d$$

Where:

- c = a constant (usually ≥ 0)
- d = degree of the polynomial
- Introduces **nonlinearity** into the decision boundary.
- Captures **feature interactions** of various degrees.

Higher d means more complex curves.

Use case: When data has **non-linear but polynomial relationships**

---

3. Radial Basis Function (RBF) / Gaussian Kernel

$$K(x, x') = \exp\left(-\frac{||x - x'||^2}{2\sigma^2}\right) \quad \text{or} \quad K(x, x') = \exp(-\gamma||x - x'||^2)$$

- Projects data into **infinite-dimensional space**
- **Very flexible**: Can model **complex, wavy boundaries**
- Highly effective for most **non-linear classification** tasks

γ controls the **radius of influence** of each point

Use case: Image classification, bioinformatics, handwriting recognition

---

4. Sigmoid Kernel (Like Neural Network)

$$K(x, x') = \tanh(\alpha x^T x' + c)$$

- Similar to activation function in neural networks
- Behaves like a **2-layer perceptron**

Not commonly used in practice — may not satisfy SVM kernel conditions for all values

Use case: Historical use; rarely applied now

Q. Write details notes on Quadratic Programming solution for finding maximum margin separation in support vector machine