ISHAAN JAIN

Q1. A bank developed a loan approval prediction model. The confusion matrix on test data is given below:

	Predicted Approved	Predicted Rejected
Actual Approved	70 (TP)	30 (FN)
Actual Rejected	10 (FP)	90 (TN)

Tasks:

1. Calculate Accuracy, Precision, Recall, f1-score

Total Samples = TP + FN + FP + TN = 200

$$\text{Accuracy} = \frac{TP + TN}{\text{Total}} = \frac{70+90}{200} = \frac{160}{200} = 0.8 = 80\%$$

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{70}{70+10} = \frac{70}{80} = 0.875 = 87.5\%$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{70}{70+30} = \frac{70}{100} = 0.7 = 70\%$$

$$f1\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{(\text{Precision} + \text{Recall})} = \frac{2 \times 0.875 \times 0.7}{(0.875 + 0.7)} = 0.7778 = 77.78\%$$

2. Which Metric Matters most for bank loans?

- Precision measures how many predicted approvals are truly safe. A low precision means many false approvals, ie loans are granted to risky customers, leading to financial loss.
- Recall measures how many actual good applicants are captured (ie avoid wrongly rejected good customers)
  Low recall → Missed customers and lost business.

For a bank, the usual emphasis is on precision (minimizing false approvals) because an approved-but-defaulting loan is expensive. However, business strategy matters:

- If the company's priority is credit risk minimization, prioritize precision
- If the company wants to grow customer base / be inclusive, it may accept a lower precision to increase recall (capture more good customers), but that increases risk.

3. How to improve Recall.

To raise recall (catch more actual good applicants) and reduce FN (but expect FP to raise).

Methods →

(1) lower decision threshold for classifying "Approved". This directly increases recall
(2) Class weighting training: penalize FN less so model favors catching positives.
(3) Resampling: oversample minority positives with SMOTE/ADASYN or undersample negatives.
(4) Switch or ensemble methods: Better capture patterns
(5) Tune hyperparameters to favor recall
(6) Use more informative features to improve model seperability.

Q2. You trained a Linear Regression Model to predict house prices. The regression equation is

Price = 50000 + 2000 x (size in sq. mt) + 10000 x (No. of bedrooms)

for a house with 1500 (sq mt) and 3 (bedrooms)

1. Calculate the predicted price.

= 50 000 + 2000 x 1500 + 10000 x 3 =
50000 + 3000000 + 30000 = 3080000

ISHAAN JAIN

2. If the actual market price is 9000000 calculate the Residual Error.

Residual = Actual - Predicted = 9000000 - 3080000
= 5920000

We can see that the model underestimates the house price by 5920000, which is a significant error.

3. Discuss whether Linear Regression is the best model here, or if another ML algorithm might perform better.

No, the very large residual suggests missing important information or non-linearity.

REASONS

(1) Outliers: Expensive properties or luxury homes may act as outliers. Tree based model handle these outliers in a better way.

(2) Interactions: Bedroom x location, size x age interactions may matter.

(3) Heteroscedasticity: Variance of errors often grow with price. Check residuals vs predictions.

(4) feature set: The model use only size and bedrooms, but ignores features like location, lot size, age, condition, nearby amenities, crime rates, school district, balcony/garbage, floor. Missing these can cause errors.

(5) Linearity Assumption between predictors and price, where it shows non-linear behaviour (diminishing returns for very large size, multiplicative effect for location).

ALTERNATIVE MODELS

(1) Gradient Boosted Trees (XGBoost, LightGBM, CatBoost) to handle non-linearity, outliers, missing values.

(2) Random forest - robust, can capture non-linearity
(3) Neural Networks - If very large data and complex feature representations (images, textual descriptions)
(4) Regularized Linear Model (Ridge, Lasso) - If many correlated features exist, these help avoid overfitting.

Q3. You design a simple feedforward neural network with the following architecture:

- Input layer: 2 inputs X_1, X_2
- Hidden layer: 2 neurons with activation function as sigmoid.
- output layer: 1 neuron (sigmoid activation)

This parameters are -

- Weights from inputs to hidden layer

$w_{1,1} = 0.4, w_{1,2} = 0.6, w_{2,1} = 0.5, w_{2,2} = 0.9$

- hidden to output layer weight

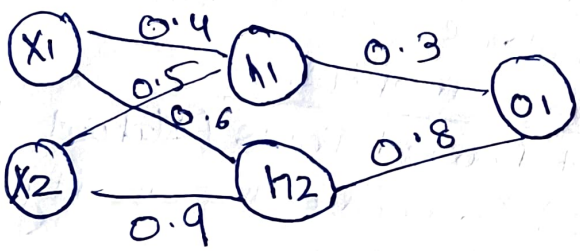
$v_1 = 0.3 \quad v_2 = 0.8$

- Bias for hidden neurons $b_h = 0.1$, bias for output neuron $b_o = 0.2$
- A single training example:

$X_1 = 1, X_2 = 2$, target $y = 1$

- learning rate = 0.1

Tasks
1. Perform forward propagation, compute hidden layer output and final output.



net $h_1 = x_1 w_{1,1} + x_2 w_{2,1} + b_h$
$= 1(0.4) + 2(0.5) + 0.1 = 1.5$

Activation $= \sigma(1.5) = \frac{1}{1+e^{-1.5}} = 0.8175$

net $h_2 = x_1 w_{1,2} + x_2 w_{2,2} + b_h$
$= 1(0.6) + 2(0.9) + 0.1 = 2.5$

Activation $= \sigma(2.5) = 0.9241$

Output neuron

net $o_2 = v_1 h_1 + v_2 h_2 + b_o = 0.3(0.8175) +$
$0.8(0.924) + 0.2 = 1.18$

final o/p $= \sigma(1.18) = \frac{1}{1+e^{-1.18}} = 0.765$

2. Calculate mean squared error (MSE)

$MSE = (1 - 0.765)^2 = 0.05486$

NN loss $L = \frac{1}{2}(y - \hat{y})^2 = 0.5 \times 0.05486$
$= 0.02743$

3. Using backpropagation, compute gradients of loss w.r.t all weights and biases, and compute the weighted updates. Show final updated weights after one iteration.

Calculating gradient using chain rule:

Error $= 1 - 0.765 = 0.235$

$\Delta w_{ij} = \eta \delta_j o_i$

$\delta_j = o_j (1-o_j)(t_j - o_j)$ (if j is output)

$\delta_j = o_j (1-o_j) \sum_k \delta_k w_{kj}$ (if j is input)

For output unit ⇒

$\delta_1 = y(1-y)(y_{target} - y)$

$= 0.765 (0.235)(0.235) = 0.04224$

For hidden unit ⇒

$\delta_{h1} = y_3 (1-y_3) w_{h101} \delta_{01}$

$= 0.8175 (0.1825) (0.3 * 0.04224)$

$= 0.00189$

$\delta_{h2} = y_4 (1-y_4) w_{h201} \delta_{01}$

$= 0.9241 (0.0759) (0.8) (0.04224)$

$= 0.00237$

New weights ⇒ $\Delta w_{ji} = \eta \delta_j o_i$

$\Delta W_{h201} = \eta \delta_{01} y_4 = 0.1 * 0.04224 * 0.9241$

$= 0.003903$

New $W_{h201} = 0.003903 + 0.8 =$

0.803903

$\Delta W_{h101} = \eta \delta_{01} y_3 = 0.1 \times 0.04224 * 0.8175$

$= 0.00345$

New $W_{h201} = 0.00345 + 0.30$

$= 0.30345$

ISHAAN JAIN

Q4. Explain the role of regularization techniques in supervised and deep learning models. In your answer discuss:

1. Why regularization is needed in machine learning models.
   - To prevent overfitting: models (especially high-capacity deep nets) can fit noise in training data, yielding poor generalization to new data!
   - Regularization constraints model complexity (weights, architecture, training duration), encouraging solutions that generalize better.

2. Difference between L1, L2, and Elastic Net regularization (with proper equations)

L2 regularization (weight decay): add $\lambda \frac{1}{2} \sum_i w_i^2$

$$L_{L2}(\theta) = L(\theta) + \lambda \frac{1}{2} \sum_i w_i^2$$

$L(\theta)$ is the original loss (MSE, cross-entropy)

- encourages small weights, penalizes large weights
- Tends to produce distributed small weights, not exact zeros.

L1 regularization: add $\lambda \sum_i |w_i|$

$$L_{L1}(\theta) = L(\theta) + \lambda \sum_i |w_i|$$

- encourages sparsity (many weights driven exactly to zero).
- Gradient is sign-based (subgradient at zero). Useful for feature selection.

Elastic Net: combines L1 and L2:

$$L_{EN}(\theta) = L(\theta) + \alpha\lambda \sum_i |w_i| + (1-\alpha)\frac{\lambda}{2} \sum_i w_i^2$$

where $0 \leq \alpha \leq 1$ controls mixture. Elastic Net inherits sparsity (L1) and stability (L2).

3. How dropout and early stopping act as regularization Methods in neural networks.

- Dropout – during training, randomly zero out (drop) a subset of neurons (or connections) each mini-batch with probability p. This forces the network to not rely on any single unit and effectively trains an ensemble of subnetworks. At interface, scale weights (or keep all units and multiply activations) so expected activations match. Dropout reduces co-adaptation and improves generalization.
- Early Stopping: Monitor validation loss; stop training once validation loss stops improving (or starts increasing). This prevents overfitting that occurs with too many optimization steps. It's a simple and very effective regularizer – training time becomes a hyperparameter.

Other typical regularizers & techniques –

- Batch normalization (stabilize training)
- Data Augmentation (increase training diversity)
- Weight Sharing/architecture choices (Conv nets) reduce parameter count.

4. Real life applications where regularization helps Improve Model generalization

- Medical Imaging – prevent overfitting when labelled examples are scarce. (eg disease detection from scans).
- finance / risk modelling – L1 helps select a small subset of predictive features for interpretable models.

3. How dropout and early stopping act as regularization Methods in neural networks.

- Dropout – during training, randomly zero out (drop) a subset of neurons (or connections) each mini-batch with probability p. This forces the network to not rely on any single unit and effectively trains an ensemble of subnetworks. At interface, scale weights (or keep all units and multiply activations) so expected activations match. Dropout reduces co-adaptation and improves generalization.
- Early Stopping: Monitor validation loss; stop training once validation loss stops improving (or starts increasing). This prevents overfitting that occurs with too many optimization steps. It's a simple and very effective regularizer – training time becomes a hyperparameter.

Other typical regularizers & techniques –

- Batch normalization (stabilize training)
- Data Augmentation (increase training diversity)
- Weight Sharing / architecture choices (Conv nets) reduce parameter count.

4. Real life applications where regularization helps improve model generalization.

- Medical Imaging – prevent overfitting when labelled examples are scarce. (eg disease detection from scans).
- finance / risk modelling – L1 helps select a small subset of predictive features for interpretable models.

- NLP with limited data: dropout and L2 improve generalization for sequence models.
- Recommendation systems: regularizations prevents memorization of sparse user-item interactions.
- Autonomous driving: prevents overfitting to training scenarios; improves robustness to unseen conditions.
- Any small-data regime: regularization is crucial whenever model capacity >> data.