S2 compute the gradients

→ The gradients of the cost function with respect to m and c are computed as

① gradient with respect to m

$$\frac{\partial J}{\partial m} = \frac{1}{N} \sum_{i=1}^{N} -x_i (y_i - (mx_i + c))$$

② g with respect to c

$$\frac{\partial J}{\partial c} = \frac{1}{N} \sum_{i=1}^{N} -(y_i - (mx_i + c))$$

S3 update the parameters

→ using the learning rate α, update the para

- meters m & c

$$m = m - \alpha \frac{\partial J}{\partial m}$$

$$c = c - \alpha \frac{\partial J}{\partial c}$$

S4 iterate through the above steps for a certain number of iterations or until the cost function converges to a minimum.

S5 The process stops when the cost function $J(m, c)$ doesn't change significantly b/w iterations.

(2) → linear regression model trained on 500 houses
→ model predicts house prices based on square
→ model performs poorly on new data

(a) possible reasons for poor performance

(1) overfitting :- The model performance well on training data but poorly on new data

(2) underfitting :- the model is too simple and does not capture the patterns in the data

(3) Data quality issues :- Noisy or inconsistent data, missing values, or incorrect labels.

(4) feature scaling :- poorly scaled features.

(5) outliers :- extreme values in the data can skew the model.

(6) Insufficient data :- 500 data points may not be enough to train a robust model

(b) Improve their model's performance

→ use cross-validation to check model generalization.

→ feature engineering :- add relevant features

→ use data augmentation or increase dataset size

→ feature scaling :- Normalize or standardize the square footage values to improve gradient descent efficiency

→ Handle data issues :- clean data by removing outliers, filing missing values and ensuring consistency.

→ Try diff models :- DT, RF or NN for better performance.

(3) polynomial regression eqn for two features with degree 2.

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 + w_4 x_2^2 + w_5 x_1 x_2$$

parameters to be estimated

w_0 → intercept

w_1 → coefficient for x_1

w_2 →

w_3

w_4

w_5

→ These parameters are estimated using gradient descent or the Normal equation

$$\theta = (X^T X)^{-1} X^T y$$

(4)

→ Overfitting

↳ the model learns noise & pattern specific to the training data.

Symptoms:

① very high accuracy on training but poor performance on test data

② the model might memorize data points rather than generalizing.

Example

① Suppose the dataset contains 500 houses, and the model includes highly specific features like street name, exact house age, or very high-degree polynominal terms

② The model might learn patterns that only apply to the training set, but this rule won't generalize to new houses.

result :- High training accuracy → 98%.
low test accuracy → less 60%.

Underfitting

→ The model is too simple to capture important patterns in the data.

Sym

→ poor performance on both training & test data
→ High bias (unable to capture relationship properly)

Eg

① suppose the model only considers square footage but ingores location & no. of rooms.

② It assumes house prices are strictly linear (eg:- price = 200 x sqft), even though prices aslo depend on factors like house age, proximity to school etc

result , Low accuracy on both train & test

→ How to improve the model

→ Overfitting

① Reduce Complexity

↳ reduce the degree of polynomial regression

↳ remove irrelevant features like exact student names.

② Increase Training data

③ Split data into multiple training/test subsets to ensure the model generalizes well.

→ underfitting

① add more features

↳ Include relevant variables like location, crime rate

② Use a more flexible model

↳ Try DT, RF, or NN

③ Use feature engineering

Cross- validation

① Hyperparameter tuning

↳ Hyperparameter tuning is the process of finding the best combination of hyperparameters to optimize a model's performance.

→ In polynomial regression, key hyperparameters includes

① degree of the polynomial

② Regularization strength (prevents overfitting

```
param_grid = {
    'poly-degree' : [1, 2, 3],
    'ridge-alpha' : [0.01, 0.1, 1]
}
```

→ dop (d)

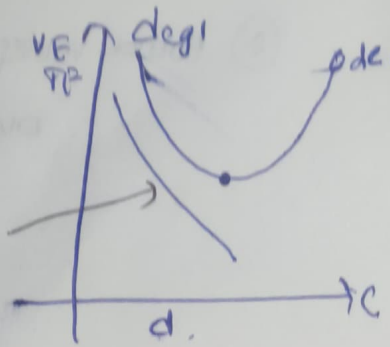
→ low dp (eg d = 1) might underfit the data

→ high dp (eg d = 10) might overfit the data

→ tuning helps find the optimal degree the balances bias & variance

Degree :- Hyperparameters (para)

Coefficient parameter.



- Regularization parameter (α)
  - → used in ridge (L2) or lasso (L1) regression
  - → helps prevent overfitting by penalizing large coefficients
  - → higher α - more regularization

→ HPT methods

1. Grid search
2. Random search
3. cross-validation

(8) HPT def

→ key parameters

1. max_depth → limits tree depth to prevent overfitting
   - a. low values → underfitting
   - b. high values → overfitting
2. min_samples_split → min samples required to split a node
   - a. higher values → prevents unnecessary splits
   - b. lower values → more splits, leading to overfitting.

③ min_samples_leaf → min samples per leaf node

① HV → simpler model, reduces overfitting

② LV → more complex tree, risk of overfitting

④ max_features → limits no. of features considered per split

Best practise :→ use sqrt(n_features) for classification, log2(n_features) for regression

⑤ criterian → split quality metric

→ gini or entropy for classification

→ mse for regression

⑥ ccp_alpha → post-pruning parameter

→ Higher alpha → more pruning

→ lower alpha → less pruning

HPT Methods

① Grid search

② Random Search

③ Bayesian Optimization

④ Cross-Validation

5) Cross-val

over-fitting

→ model learns both the patterns & noise from training data, reducing generalization ability

→ High variance, low bias

→ poor accuracy due to lack of generalization

→ Too complex (too many features)

eg:- A deep neural network memorizing training data instead of generali -zing.

underfitting

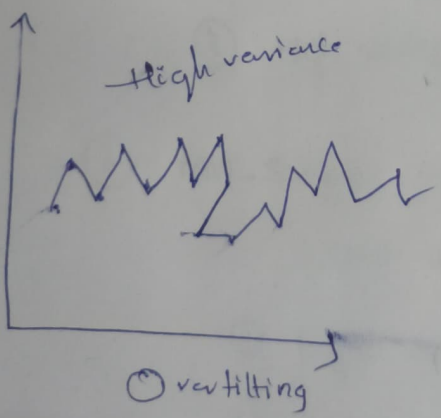
→ model is too simple to capture patterns in the da leading to poor peforama

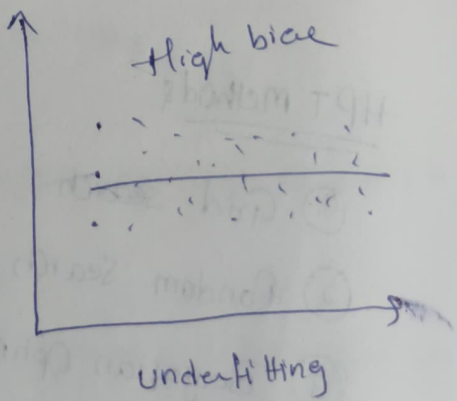
→ High bias, low variance

→ poor accuracy due to overly simplistic model

→ Too simple (insufficient features

eg:- A linear regression mod failing to fit a non-linea dataset





How cross-validation helps detect & fix overfitting & underfitting

→ CV helps checks if a model is too-complex (overfit) or too simple (underfit) by testing it on different parts of the data

detecting issues

→ overfitting :- def

→ underfitting :- def

fixing

over	under
→ use regularization	→ use a more complex model
→ reduce model complexity	→ Reduce regularization
→ use more training data or data augmentation	→ improve feature selection & preprocessing

⑥ K-fold CV

→ It is a technique used to evaluate ml models by splitting the dataset into k equal parts (folds). the model is trained on k-1 folds and tested on the remaining 1 fold, repeating this process k times with different test folds. the final performance is the average of all k-iterations.

How it helps

→ Reduce bias by ensuring every data point gets a chance in both test & training data sets

→ Helps detect over & underfitting

→ Ensures that the model is tested on diff. subsets of the data.

k-fold cross-validation

→ Splits data into k-folds & runs k-train/testing iterations

→ reduce variance by averaging results over k-runs

→ Uses more data for training, making the model better

→ Slower computation time

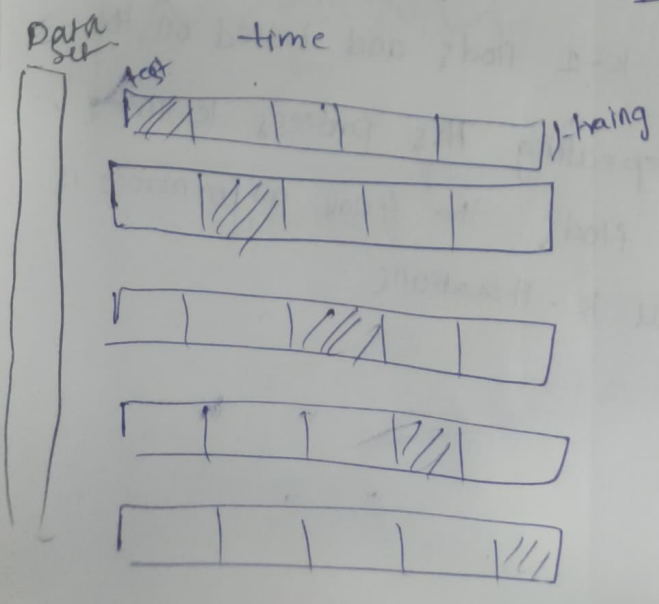
Train-test split

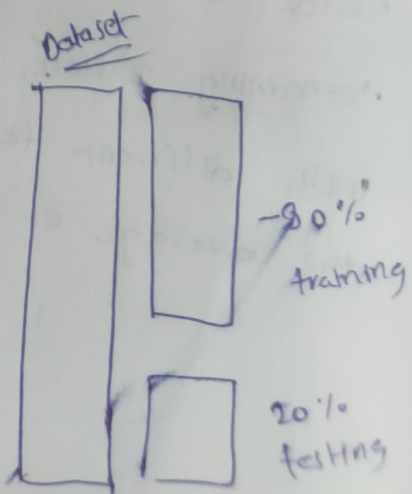
→ splits data into on training set & one test set
Eg:- (80-20) split

→ can have high variance as the performance depends on a single test set

→ less training data, as part of the dataset is reserved for testing

→ faster computation time





Why CV

→ small datasets
→ Hyperparameter tuning
→ avoiding overfitting
→ Helps detecting overfitting & underfitting
→ reduces variance in performance metrics
→ more reliable model evaluation

16 Confusion matrix

→ It is a table to evalute the performance of classification model by comparing actual & predicted values

	Predicated +ve	-ve
Actual +ve	TP	FN
A -ve	FP	TN

Significance

→ helps identify type of error
→ essential for imbalanced datasets where accuracy alone is misleading.
→ enables calculations of multiple performance metrics for better decision making.

TP - correctly predicted positive cases

TN - correctly -ve

FN - inc +ve (Type 1 error)

FN - inc -ve (Type 2 error)

ii) precision, recall & f1-score.

(1) Precision (positive predictive vaule)

↳ It measures how many of the predicted -ve cases were actually correct

$$\text{precision} = \frac{TP}{TP + FP}$$

High p → fewer fp (useful in spam detection, where false alarms should be minimized)

low p → Many incorrect +ve predictions

(2) Recall (sensitivity or TP Rate)

↳ It measures how many of the actual positive cases were correctly identified.

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

High recall → fewer false -ve (imp in medical diagnosis)

low R → Many missed +ve cases

③ F1-score (Harmonic mean of precision & Recall)

→ It balances precision & recall, making it useful when both false +ve & false -ve matter

$$F_1\text{-score} = 2 \times \frac{P \times R}{P + R}$$

High F_1 → good balance b/w P & R

low F_1 → Indicates an imbalance b/w P & R

⑬ ROC (Receiver Operating Characteristic)

→ is a graphical tool used to evaluate the performance of a binary classifier at diff threshold values.

→ It helps assess how well a model distinguishes b/w +ve & -ve classes

→ ROC curve plot

True +ve rate TPR => Recall → $\frac{TP}{TP+FN}$

False -ve rate (FPR) => $\frac{FP}{FP+TN}$

→ How to interpret the ROC-curve

① Ideal curve

↳ The curve is close to the the top-left corner

↳ meaning high TPR & low FPR

② Random model

↳ diagonal line (45°) represents random guessing (AUC = 0.5)

③ poor model

↳ below the diagonal means the classifier is worse than random.

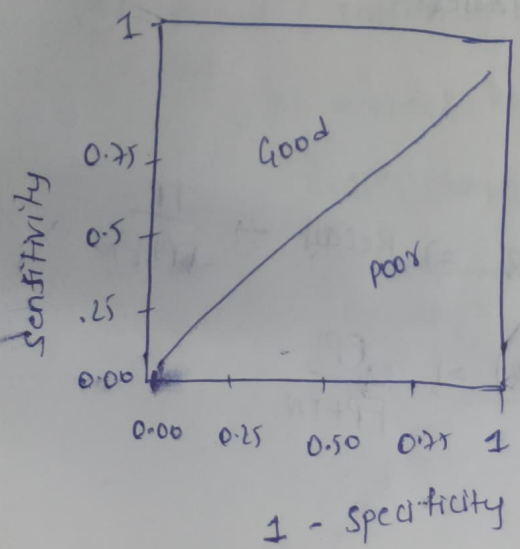
→ · AUC value — model performance

0.9 - 1 → Excellent

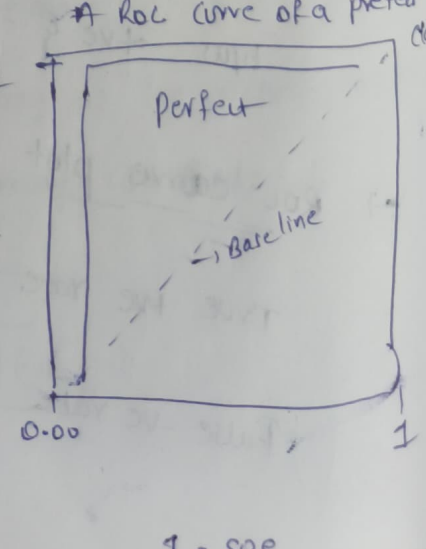
⋮

0.5 - 0.6 → very poor.

A ROC curve of a random classifier



A ROC curve of a perfect classifier



Use

→ threshold selection
→ Imbalanced data handling
→ comparing models

(14) diff b/w R^2 & MSE

MSE (Mean squared error)

→ measures the avg speed squared diff b/w the actual & predicted values

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

Lower MSE → better model fit
Higher MSE → poor model performance

→ Used for comparing models in terms of absolute error

R^2 (R^2, co-efficient of determination)

→ measures the proportion of variance in the dependent variable that is explained by the model

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

SS_{res} → Sum of squared errors

SS_{tot} → total variance in the target variable

R^2 range 0 to 1

$R^2 = 1$ → perfect fit

$R^2 = 0$ → model is no better than the mean

$R^2 < 0$ → model is worse than using the mean as a predictor

→ Used for understanding how well the model explains variance in the data

17 Mean absolute error (MAE) & MSE

MAE

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

→ measures the avg absolute diff. b/w actual & predicted values

⇒ lower MAE → better predictions

→ from sklearn.metrics import mean_absolute_error,
mean_squared_error

y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0, 2, 8]

mae = mean_absolute_error(y_true, y_pred)
mse = mean_square_error(y_true, y_pred)
print(mae, mse)

→ Without libaries

n = len(y_true)
y_true = [ ]
y_pred = [ ]

mae = sum(abs(y_true[i] - y_pred[i]) for
i in range(n))/n

mse = sum((y_true[i] - y_pred[i])**2 for
i in range(n))/n

print(mae)
print(mse)

18) Refer 13 Answer

19) accuracy = $\frac{TP+TN}{TP+TN+FP+FN}$

22) y does linear svm fail for non-linearly separable data.

A) linear svm tries to find a staright hyperplane that best separates two classes.

=) however, it fails when the data is non-linearly separable

→ No straight line can separate the classes

→ if the data is non-linearly distributed, a linearly decision boundary cannot separete the two classes effectively

eg. consider XOR data, where no straight line can separate +ve & -ve classes

→ High Misclassification rate

Since linear svm forces a straight boundary, it will misclassify many points in a complex decision space

→ poor generalization

It fails on unseen data when the actual distribution is non-linear

→ soln

→ polynomial kernal

→ Radial basis function (RBF kernal)

↳ maps data into infinite dimensions, making non-linear separation possible

→ sigmoid kernel → mimics nn

23 SVM

↳ Aim to find a decision boundary (hyperplane) that maximally separates data points belonging to diff classes.

Margin in svm

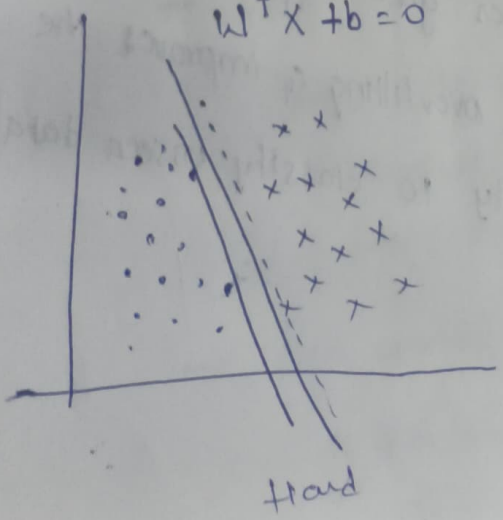
↳ Margin in svm is the distance b/w the decision boundary (hyperplane) and the closest data points from each class

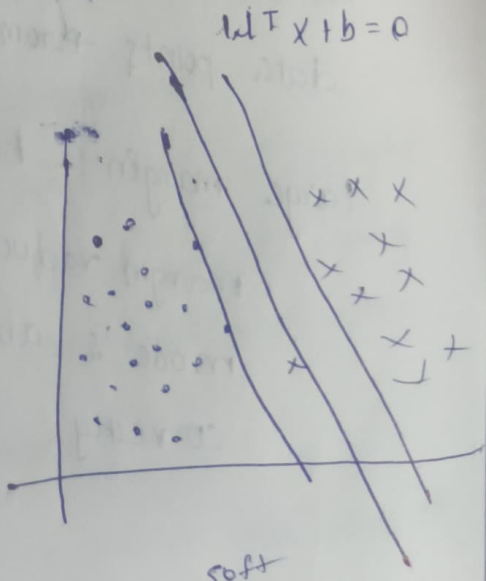
Large margin :- better generalization, a larger margin reduces overfitting & improves the model's ability to classify unseen data correctly.

Small margin : higher risk in overfitting, a small margin makes the classifier sensitive to noise

types of margin.

	Hard margin	Soft Margin
applicability	used when data is perfectly separable	used when data is not perfectly separable
Misclassification	NO	Allows some
Margin	Maximized	balances margin size
overfitting risk	high	lower
use case	when training data is clean & linearly separable	When data has some noise or overlap b/w classes





Optimization framework for hard margin

objective :- Maximize the margin

- the margin is given by $\frac{1}{\|w\|}$, so maximizing
- the margin is equivalent to minimizing $\|w\|$

-) .

given a dataset (x_i, y_i)

where $y_i \in \{-1, +1\}$

where w & b are weights & bias

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

constraints

$$y_i (w \cdot x_i + b) \geq 1, \quad \forall i$$

-) The constraint ensures that each data point lies outside or on the margin boundaries

-) The lagrange multipliers technique is used to solve this optimization problem.

24 diff kernels in SVM & their hyperparameter

→ SVM uses kernel functions to transform data into higher dimensions, allowing non-linearly separable data to be classified.

Kernal	Mathematical function	Hyperparameter	usage
linear kernal	$K(x,x') = x \cdot x'$	C (regularization parameter	used when data is linearly separable
polynomial kernal	$K(x,x') = (x \cdot x' + c)^d$	C d → degree of freedom c → coefficent	non-linearly relationship with polynomical features
Radial basis function (RBF)	$K(x,x') = e^{-\gamma \|x-x'\|^2}$	C γ (gamma)	$x \neq x'$ complex, non-linearly separable data
Sigmod kernal	$K(x,x')$ $\tanh(\alpha x \cdot x' + c)$	C, c α (scaling factor)	similar to nerval network, used when data is binary
Guassian kernal	$e^{-\frac{\|x-x'\|^2}{2\sigma^2}}$	C σ (sigma)	high-dimensinal problems

9) SVD (singular value decomposition)

i) is a matrix factorization technique that decomposes any given matrix A into three matrices

$$A = U \Sigma V^T$$

where

U => Orthogonal matrix containing the left singular vectors

Σ => diagonal matrix with singular values

V^T => orthogonal matrix (right singular vectors)

→ used in dimensionality reduction, noise reduction & data compression.

How does SVD relate to PCA

→ PCA is a technique used for dimensionality reduction. it finds the principal components of the data which are the dirn of maximum variance

① PCA is performed using SVD

→ PCA computes the eigenvectors of the co-variance matrix, but instead of calculating the co-variance matrix explicitly, it can

be obtained directly using SVD

→ the right singular vectors (V) in SVD corresponds to the principal components in PCA

→ & represents the importance of each principal component

(2) dimensionality reduction

↳ in PCA, we select the top k singular values & their corresponding singular vectors to reduce the no. of dimensions while preserving as much variance as possible.

Eg

$$A = \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix}$$

SVD :-

$$A = U \Sigma V^T$$

$U \Rightarrow AA^T$ → eigen vector

$V \Rightarrow A^TA$ → eigen values

$\Sigma \Rightarrow A^TA$ → eigen vector

Sol (i)

$$A^TA = \begin{bmatrix} 4 & 3 \\ 0 & -5 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix}$$

$$= \begin{bmatrix} 25 & -15 \\ -15 & 25 \end{bmatrix}$$

Eigenvalues A^TA

$$\det(A^TA - \lambda I) = 0$$

$$\begin{vmatrix} 25-\lambda & -15 \\ -15 & 25-\lambda \end{vmatrix} = 0$$

$$(25-\lambda)(25-\lambda) - (-15)(-15) = 0$$

$$\lambda^2 - 50\lambda + 400 = 0$$

$$\lambda_1 = 40 \quad , \quad \lambda_2 = 10$$

$$\sigma_1 = \sqrt{40}$$

$$\sigma_2 = \sqrt{10}$$

$$\Sigma = \begin{bmatrix} \sqrt{40} & 0 \\ 0 & \sqrt{10} \end{bmatrix}$$

~~S2~~ S3

~~AAT~~ $\lambda_1 = 40$

$$\begin{vmatrix} 25-40 & -15 \\ -15 & 25-40 \end{vmatrix} \begin{bmatrix} v_{11} \\ v_{12} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$v_1 = \frac{1}{\sqrt{2}}[1, -1]$$

$\lambda_2 = 10$

$$v_2 = \frac{1}{\sqrt{2}}[1, 1]$$

$$V = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

Sy for U

$$AA^T = \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix} \begin{bmatrix} 4 & 3 \\ 0 & -5 \end{bmatrix} = \begin{bmatrix} 16 & 12 \\ 12 & 34 \end{bmatrix}$$

Eigen value

$$\lambda_1 = 40 \quad \lambda_2 = 10$$

$\lambda_1 = 40$

$$\begin{bmatrix} 16-40 & 12 \\ 12 & 34-40 \end{bmatrix} \begin{bmatrix} U_{11} \\ U_{12} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$U_1 = \frac{1}{\sqrt{13}} [4, 3]$$

$\lambda_2 = 10$

$$U_2 = \frac{1}{\sqrt{13}} [-3, 4]$$

$$U = \begin{bmatrix} 4/\sqrt{13} & -3/\sqrt{13} \\ 3/\sqrt{13} & 4/\sqrt{13} \end{bmatrix}$$

Totally →

$$U = \begin{bmatrix} 4/\sqrt{13} & -3/\sqrt{13} \\ 3/\sqrt{13} & 4/\sqrt{13} \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \sqrt{40} & 0 \\ 0 & \sqrt{10} \end{bmatrix}$$

$$V = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

10) Curse of dimensionality

↳ Refers to the challenges & problems that arise when working with high-dimensional data.

→ Impact of increasing the no. of features

① increased sparsity

↳ In high-dimensional spaces, data points become more spread out, making it difficult for models to find meaningful patterns.

② Higher computational cost

↳ More features require more memory & processing power, slowing down training & inference.

③ Overfitting

↳ A larger no. of features can cause models to memorize noise instead of learning useful patterns, reducing generalization.

④ difficulty in dis-based algorithms

↳ Algo like k-NN & clustering rely on distance metrics, which become less effective in high dimensions as distance become almost uniform.

Strategies to mitigate the curse of dimensionality

① feature selection
   ↳ choose only the most relevant features

② dimensionality reduction
   ↳ use techniques like PCA or SVD to transform high to low dimensional space while retaining imp info

③ Regularization
   ↳ apply L1 or L2 regularization to prevent overfitting

④ collect more data

25

Eigen values

$\lambda_1 = 4.5$

$\lambda_2 = 2.3$

$\lambda_3 = 0.7$

Eigen vectors

$V_1 = \begin{bmatrix} 0.8 \\ 0.5 \end{bmatrix}$ $V_2 = \begin{bmatrix} -0.4 \\ 0.9 \end{bmatrix}$ $V_3 = \begin{bmatrix} 0.2 \\ -0.1 \end{bmatrix}$

dataset

$X = \begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix}$

principal component

$PC_1 = \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix}$ $PC_2 = \begin{bmatrix} -0.6 \\ 0.8 \end{bmatrix}$

→ Identify ~~the~~ PC_1 & PC_2

PC_1 corresponds to the largest eigenvalue

$\lambda_1 = 4.5$

$PC_1 = V_1 = \begin{bmatrix} 0.8 \\ 0.5 \end{bmatrix}$

$\lambda_2 = 2.3$

$PC_2 = V_2 = \begin{bmatrix} -0.4 \\ 0.9 \end{bmatrix}$

compute projection on PC_1

projection = $X \cdot PC$

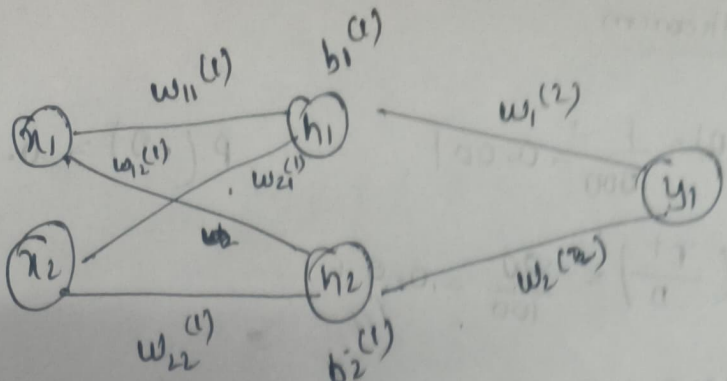
$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix} \begin{bmatrix} 0.8 \\ 0.6 \end{bmatrix} = \begin{bmatrix} 3.4 \\ 6.2 \\ 9.0 \end{bmatrix}$$

PC2

$$\begin{bmatrix} 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix} \begin{bmatrix} -0.4 \\ 0.9 \end{bmatrix} = \begin{bmatrix} 1.2 \\ 1.6 \\ 2.0 \end{bmatrix}$$

Q6

(x_1) $w_{11}^{(1)}$



$$L = \frac{1}{2}(y - \hat{y})$$

$$z_1 = x_1 (w_{11})^{(1)} + x_2 (w_{21})^{(1)} + b_1^{(1)}$$

$$z_2 = x_2 (w_{22})^{(1)} + x_1 (w_{12})^{(1)} + b_2^{(1)}$$

$$h_1 = \sigma(z_1)$$

$$h_2 = \sigma(z_2)$$

$$z_3 = w_1^{(2)} h_1 + w_2^{(2)} h_2 + b_2^{(2)}$$

$$y = \sigma(z_3)$$

find $w_{12}^{(1)}$, $w_2^{(2)}$

$$w_{12} \longrightarrow z_2 \longrightarrow h_2 \longrightarrow z_3 \longrightarrow y \longrightarrow L$$

$$\frac{\partial L}{\partial w_{12}} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial z_3} \times \frac{\partial z_3}{\partial h_2} \times \frac{\partial h_2}{\partial z_2} \times \frac{\partial z_2}{\partial w_{12}}$$

$$w_2^{(2)} \longrightarrow z_3 \longrightarrow y \longrightarrow L$$

$$\frac{\partial L}{\partial w_2^{(2)}} = \frac{\partial L}{\partial y} \times \frac{\partial y}{\partial z_3} \times \frac{\partial z_3}{\partial w_2^{(2)}}$$

$$\frac{\partial L}{\partial w_{12}} = 2(y-\hat{y}) \times \sigma'(z_3) \times (h_2 + h_1) w_2^{(2)} * \sigma'(z_2) * x_1$$

$$\frac{\partial L}{\partial w_2^2} = 2(y-\hat{y}) * \sigma'(z_3) * h_2 * \sigma'(z_1) * x'$$

20 Bayes's theorem

$P(D) = \frac{1}{1000} = 0.001$ $\quad P(\neg D) = 0.999$

$P\left(\frac{T^+}{D}\right) = \frac{90}{100} = 0.9$

$P\left(\frac{F}{\neg D}\right) = \frac{5}{100} = 0.05 \quad = \frac{5}{0.9999} \Rightarrow 0.05$

$$P\left(\frac{D}{T}\right) = \frac{P(T^+/D)\,P(D)}{P(T^+/D)\,P(D) + P(F/\neg D)\,P(\neg D)}$$

$$= \frac{(0.9)(0.001)}{(0.9)(0.0001) + (0.05)(0.999)}$$

$$= \frac{0.0009}{0.00009 + 0.049}$$

$$= \frac{0.0009}{0.05004} \qquad \frac{0.0009}{0.05085}$$

$$= 0.00179 \qquad 0.0177$$

$P(D/T) = 1.77\%$