Learning From Data | Topic-Wise Notes

1) Learning Models and Algorithms

1.1 Perceptron (PLA) + Step Activation

What it is (simple definition)

A **perceptron** is the simplest **binary classifier** (two classes). It tries to separate data using a **straight line** (2D) or a **flat surface** (higher dimensions). That's why we call it a **linear classifier**.

Core idea in one line

Compute a score → convert score to a class label.

Equations + meaning

(1) Score $s = \mathbf{w}^\top \mathbf{x} + b$
Meaning:

- $\mathbf{x}$ = input features (example: height, weight, etc.)
- $\mathbf{w}$ = weights (importance of each feature)
- b = bias (shifts the decision boundary)
- s = "score" (how strongly the model leans to one side)

(2) Prediction using sign / step $\hat{y} = \text{sign}(s)$
Meaning:

- If $s \geq 0$ → predict $(+1)$
- If $s < 0$ → predict (-1) So $\hat{y} \in +1, -1$.

How PLA learns (easy steps)

1. Start with random $\mathbf{w}$ and b
2. For each training point $(\mathbf{x}, y)$:
    - compute score $s = \mathbf{w}^\top \mathbf{x} + b$
    - predict $\hat{y} = \text{sign}(s)$
3. If prediction is wrong, update:

$$\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{x}, \quad b \leftarrow b + y$$

Meaning of update:

- If a point with label y was misclassified, we "push" weights toward that point's direction so next time it's more likely to be correct.
- $y\mathbf{x}$ adds the feature vector if $y = +1$, subtracts if $y = -1$.

Why it can fail

- PLA **only converges** if data is **perfectly linearly separable** (there exists a perfect straight-line split).
- If there is **noise** (wrong labels) or overlapping classes, it may keep updating forever.

1.2 Activation Functions (Step, Sigmoid, ReLU)

What is an activation function?

It converts the raw score (s) into an output. In neural networks, activations are important because they add **non-linearity** (otherwise deep networks become just a big linear model).

Step function

$$f(s) = \begin{cases} 1, & s \geq 0 \\ 0, & s < 0 \end{cases}$$

Meaning: output becomes either 0 or 1. **Problem:** it is not smooth → hard to train using gradients.

Sigmoid

$\sigma(s) = \frac{1}{1+e^{-s}}$
Meaning: converts any real number s into a number between **0 and 1**.

- If $s = 0$, output is 0.5 (neutral probability).
- Big positive s → output close to 1
- Big negative s → output close to 0 So it's "probability-like".

ReLU

$\mathrm{ReLU}(s) = \max(0, s)$
Meaning:

- If $s < 0$ → output 0
- If $s \geq 0$ → output s **Why popular:** simple and helps gradients flow better than sigmoid in deep nets.

1.3 Pocket Algorithm (Perceptron for non-separable data)

Simple definition

Pocket algorithm is like perceptron **but smarter**: Even if data is not separable, it keeps the **best solution seen so far**.

Key idea

- Run PLA updates normally.
- Also track $\mathbf{w}_{best}$ which gives **lowest training mistakes** so far.
- At the end, output the "pocket" (best stored) weights.

Why useful: you always end with a decent classifier even when PLA never converges.

1.4 K-Nearest Neighbors (KNN)

Simple definition

KNN does not build a formula/model first. To predict for a new point, it looks at the **K closest training points** and takes a vote (classification) or average (regression).

Distance equation (Euclidean)

$$d(\mathbf{x}, \mathbf{x}_i) = \sqrt{\sum_{j=1}^{d} (x_j - x_{ij})^2}$$

Meaning:

- $\mathbf{x}$ = new (test) point
- $\mathbf{x}_i$ = a training point
- d = number of features
- The formula measures "straight-line distance" in feature space.

Choosing K (important viva)

- Small K (like 1) → very sensitive to noise → **overfitting (high variance)**
- Large K → too smooth, ignores local structure → **underfitting (high bias)**
- Pick K using **cross-validation** (try many K values and choose best validation performance)

Viva-friendly point: "Loss of KNN"

KNN does not minimize a single global loss function during training. Instead, it's a **local method** (vote from neighbors). The "learning" is basically storing data.

Practical note (often asked)

KNN is sensitive to **feature scaling**. If one feature has huge values, it dominates distance → always standardize/normalize.

1.5 Linear Regression

Simple definition

Predict a **real number** (like price, temperature) using a weighted sum of features.

Model

$\hat{y} = \mathbf{w}^\top \mathbf{x} + b$
Meaning: predicted value = weighted sum of inputs + bias.

Loss (MSE)

$E(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} (\hat{y}_n - y_n)^2$
Meaning:

- N = number of training samples
- y_n = true target
- $\hat{y}_n$ = predicted target
- Square punishes big errors more. Average gives overall error.

How we minimize it

- **Normal equation** (closed-form solution) for small/medium problems
- **Gradient Descent / SGD** for large data

1.6 Logistic Regression

Simple definition

Logistic regression is used for **classification**, but outputs a **probability**.

Model

Score: $s = \mathbf{w}^\top \mathbf{x} + b$
Meaning: raw "push" toward class 1 or class 0.

Probability: $p = P(y = 1 \mid \mathbf{x}) = \sigma(s) = \frac{1}{1+e^{-s}}$
Meaning: probability that class is 1 (between 0 and 1).

Loss (Cross-Entropy)

$E(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \Big[-y_n \log(p_n) - (1-y_n) \log(1-p_n) \Big]$
Meaning:

- If true label $y_n = 1$, loss becomes $-\log(p_n)$ → pushes p_n toward 1
- If true label $y_n = 0$, loss becomes $-\log(1-p_n)$ → pushes p_n toward 0 So it strongly punishes confident wrong predictions.

Why not MSE for classification? (common viva)

MSE works, but cross-entropy matches probability modeling better and gives cleaner gradients for classification.

1.7 Linear vs Logistic Regression (quick)

- **Output:** Linear → real value, Logistic → probability (0 to 1)
- **Task:** Linear → regression, Logistic → classification
- **Loss:** Linear → MSE, Logistic → cross-entropy

1.8 SVM (Support Vector Machine)

Simple definition

SVM finds a separating boundary that leaves the **largest gap (margin)** between classes. Bigger margin → more confidence → often better generalization.

Hard-margin SVM (when separable)

Constraint: $y_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1$
Meaning: every point must be correctly classified and not too close to boundary.

Objective: $\min \frac{1}{2}|\mathbf{w}|^2$
Meaning: SVM tries to make $|\mathbf{w}|$ small because that **increases margin**.

Support vectors

The points closest to the boundary are **support vectors**. They “support” the boundary — if you move them, boundary changes.

Soft-margin SVM (when not separable)

$\min \frac{1}{2}|\mathbf{w}|^2 + C\sum_{n=1}^{N}\xi_n$ subject to $y_n(\mathbf{w}^\top \mathbf{x}_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0$

Meaning:

- ξ_n = how much point breaks the margin rule (violation)
- C = tradeoff:
    - large C → punish mistakes more → less violations, possible overfit
    - small C → allow mistakes → smoother boundary, possible underfit

Kernels (for nonlinear data)

Kernel trick allows SVM to act like it’s working in a higher-dimensional space without explicitly computing it.

RBF kernel: $K(\mathbf{x}, \mathbf{z}) = \exp(-\gamma|\mathbf{x} - \mathbf{z}|^2)$
Meaning: similarity is high if points are close. γ controls how “local” it is.

Polynomial kernel: $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z} + c)^p$
Meaning: creates curved boundaries by combining features in polynomial ways.

1.9 Neural Networks (NN)

Simple definition

Neural network = many neurons arranged in layers. Each neuron does: weighted sum → activation.

Neuron equation

$a = \phi(\mathbf{w}^\top \mathbf{x} + b)$

Meaning:

- $\mathbf{w}^\top \mathbf{x} + b$ = score
- $\phi(\cdot)$ = activation (ReLU, sigmoid, etc.)
- a = neuron output

Where learning happens

Learning means adjusting **weights and biases** so loss becomes small.

1.10 RNN vs LSTM

RNN (simple)

Used for sequence data (text, time series). It keeps a “memory” called hidden state.

$\mathbf{h}_t = \tanh(W_h \mathbf{h}_{t-1} + W_x \mathbf{x}_t + \mathbf{b})$

Meaning:

- $\mathbf{x}_t$ = input at time t
- $\mathbf{h}_t$ = hidden state (memory) at time t
- $W_h, W_x, \mathbf{b}$ = learned parameters
- $\tanh$ squeezes output to $(-1, 1)$

Problem: vanishing gradients → hard to learn long-term dependencies.

LSTM (simple)

LSTM adds a **memory cell** + **gates** (control what to remember/forget). That is why LSTM handles long sequences better.

1.11 Attention (and matrices)

Simple definition

Attention learns **which parts of the input are important** for the current prediction.

Attention formula

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V$$

Meaning (piece by piece):

- Q (queries): “what I am looking for”
- K (keys): “what I have”
- V (values): “the information to take”
- $QK^\top$: similarity scores between query and keys
- divide by $\sqrt{d_k}$: prevents numbers from becoming too large (stabilizes training)
- softmax: converts scores into **weights that sum to 1**
- multiply by V: makes a weighted average of information

How Q, K, V are created

$$Q = XW_Q, \quad K = XW_K, \quad V = XW_V$$

Meaning:

- X = input embeddings
- W_Q, W_K, W_V are learned matrices that convert inputs into queries/keys/values.

1.12 Transformer (GPT) + preprocessing

What is GPT based on?

GPT uses **Transformer decoder-only** architecture.

Why Transformer works well

- Attention can connect far tokens directly (long-range dependency)
- Training can be parallel (faster than RNNs)

Preprocessing steps (simple)

1. Tokenize text (break into tokens)
2. Pad/truncate to fixed length
3. Create attention mask (ignore padding; also enforce “no future token” for decoder)
4. Convert tokens to embeddings + add positional information

1.13 GD vs SGD

Gradient Descent (GD)

$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w})$
Meaning:

- η = learning rate (step size)
- $\nabla E(\mathbf{w})$ = direction that increases loss the most
- subtracting moves weights in direction that **reduces loss**
- Uses full dataset per step → stable but slow for huge data

Stochastic Gradient Descent (SGD)

$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E_n(\mathbf{w})$

Meaning:

- Uses one sample (or mini-batch) for update
- Faster, noisier updates
- Noise can help escape bad local patterns and can improve generalization

---

2) Model Evaluation and Performance

2.1 PAC Learning (simple meaning)

PAC says: with high probability, the true error is small.

- With probability $(1 - \delta)$, error $\leq \epsilon$

Meaning:

- ϵ = acceptable error (tolerance)
- δ = risk of being wrong about the guarantee Small δ means higher confidence.

2.2 Overfitting vs Underfitting

- **Overfitting:** training error low, test error high (memorizes noise)
- **Underfitting:** training error high, test error high (model too simple)

Fixes:

- Overfitting → regularization, more data, simpler model, early stopping
- Underfitting → richer features, bigger model, train longer, reduce regularization

---

2.3 Bias–Variance (core equation)

$$\mathbb{E}[(y-\hat{f}(\mathbf{x}))^2] = \mathrm{Bias}^2 + \mathrm{Var} + \sigma^2$$

Meaning:

- Left side: expected prediction error
- Bias: error from being too simple / wrong assumptions
- Variance: error from being too sensitive to training data
- σ^2: noise in data (cannot be removed fully)

Tradeoff: More complex model usually ↓ bias but ↑ variance.

2.4 (E_{in}) and (E_{out}) (classification error)

$E = \frac{1}{N}\sum_{n=1}^{N}\mathbb{I}(\hat{y}_n \neq y_n)$
Meaning:

- $\mathbb{I}(\hat{y}_n \neq y_n)$ is 1 if wrong, 0 if correct
- average wrong rate = error

Percentage: $E($
Meaning: convert fraction to percent.

2.5 Cross Validation

K-fold CV (simple steps)

1. Split data into K parts (folds)
2. Use K−1 folds for training, 1 fold for validation
3. Repeat K times (each fold becomes validation once)
4. Average validation performance
5. Pick the model/hyperparameters with best average

LOOCV

LOOCV = leave-one-out CV (K = N). Accurate but can be slow.

2.6 Generalization Bound (VC style)

$$E_{out}(h) \le E_{in}(h) + \sqrt{\frac{8}{N} \ln\left(\frac{4m_{\mathcal{H}}(2N)}{\delta}\right)}$$

Meaning (very important):

- $E_{out}(h)$: true/test error of hypothesis h
- $E_{in}(h)$: training error
- second term: **penalty** due to limited data and complexity
- N: number of training samples (more data makes bound tighter)
- $m_{\mathcal{H}}(2N)$: growth function (how complex hypothesis set is)
- δ: confidence risk (smaller δ means higher confidence but larger penalty)

3) Statistical Learning Theory

3.1 VC Dimension

VC dimension (d_{VC}) = maximum number of points that a hypothesis set can **shatter**.

Shatter meaning: For those points, the model can fit **every possible labeling**.

If $d_{VC} = \infty$, theoretical generalization guarantees become weak.

3.2 Growth Function

$m_{\mathcal{H}}(N)$ = maximum number of different labelings possible on N points.

If a model shatters N points: $m_{\mathcal{H}}(N) = 2^N$
Meaning: all possible labelings exist.

3.3 Dichotomy and Break Point

- Dichotomy: one specific labeling of points
- Break point (k): first number where model cannot realize all labelings: $m_{\mathcal{H}}(k) < 2^k$

3.4 Sauer's Lemma

$m_{\mathcal{H}}(N) \leq \sum_{i=0}^{d_{VC}} \binom{N}{i} \quad (N > d_{VC})$
Meaning: If VC dimension is finite, growth function becomes polynomial-ish instead of exponential → better generalization guarantees.

3.5 Complexity vs (E_{in}, E_{out})

- Complexity ↑ usually makes E_{in} ↓ (fits training better)
- But too much complexity → overfitting → E_{out} ↑ This creates the famous **U-shape** for test error vs complexity.

---

4) Optimization and Learning in Neural Networks

4.1 Learning Rate (η)

η controls step size.

- too big → overshoot, diverge
- too small → very slow learning

4.2 Backpropagation (core equation + meaning)

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w}$$

Meaning:

- E = loss
- w = a weight
- z = pre-activation value (before activation)
- a = activation output This is the chain rule: how changing weight changes z, changes a, changes loss.

4 steps

1. Forward pass (compute predictions)
2. Compute output error (how wrong we are)
3. Backward pass (compute gradients layer by layer)
4. Update weights: $w \leftarrow w - \eta \frac{\partial E}{\partial w}$
   Meaning: move weight in direction that reduces loss.

4.3 Brute force gradient vs Backprop

Brute force approximation: $\frac{\partial E}{\partial w} \approx \frac{E(w+\epsilon)-E(w)}{\epsilon}$
Meaning: measure how loss changes by tiny change ϵ.

Why bad: extremely slow for many weights and can be numerically unstable.

4.4 Regularization (L1 / L2)

L2

$E_{reg} = E + \lambda|\mathbf{w}|^2$
Meaning: adds penalty for large weights → keeps model smoother.

L1

$E_{reg} = E + \lambda|\mathbf{w}|_1$
Meaning: adds penalty based on sum of absolute values → encourages many weights to become exactly 0 (**sparsity**).

---

5) Data Preprocessing

Standardization vs Normalization

Standardization: $x' = \frac{x-\mu}{\sigma}$
Meaning: subtract mean μ, divide by std dev σ → makes feature have mean 0 and spread 1.

Normalization (min-max): $x' = \frac{x-x_{\min}}{x_{\max}-x_{\min}}$
Meaning: scales feature into [0,1].

Viva point: Distance-based models (KNN, SVM with RBF) usually need scaling.

---

6) Probability and Bounds

6.1 Hoeffding's Inequality

$\Pr(|E_{in} - E_{out}| > \epsilon) \le 2e^{-2\epsilon^2 N}$
Meaning: Probability that training error differs from true error by more than ϵ becomes very small when:

- N increases (more data)
- ϵ is not too tiny

---

7) Practical / Application (simple decision rules)

- To minimize E_{in}: stronger model, better optimization, more features
- For evaluation (cats vs dogs): confusion matrix + precision/recall/F1 on test set
- Choose model for unseen data: lowest validation/CV error (not lowest training error)
- Time series: LSTM/Transformers (depending on setting)
- GPT: Transformer decoder + self-attention
- If live E_{out} increases: drift checks, monitoring, retrain, recalibrate, feature update

8) Extra Viva Topics You MUST Know

8.1 Train / Validation / Test split (very commonly asked)

- Train: learn parameters
- Validation: choose hyperparameters (K, C, λ, etc.)
- Test: final unbiased evaluation

Viva warning: never tune on test set.

8.2 Data Leakage

Leakage = model indirectly sees information from the future/test set during training. Examples:

- normalizing using mean/std computed on full dataset instead of train only
- feature includes target information (like “final grade” used to predict “pass/fail”) Leakage makes results look amazing but fails in real life.

8.3 Calibration and Thresholds (for logistic regression)

- Logistic regression outputs probability p.
- But final class often uses threshold (like 0.5):

    - if $p \geq 0.5$ predict 1 else 0 Changing threshold trades precision vs recall.

8.4 Confusion matrix metrics

- TP: predicted positive, actually positive
- FP: predicted positive, actually negative
- TN: predicted negative, actually negative
- FN: predicted negative, actually positive

Precision: $\text{Precision} = \frac{TP}{TP+FP}$
Meaning: of predicted positives, how many were correct?

Recall: $\text{Recall} = \frac{TP}{TP+FN}$
Meaning: of actual positives, how many did we catch?

F1: $F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
Meaning: balances precision and recall.

Q&A (Deeper + Common Traps)

A) Perceptron / Pocket / Activation

1. **Q:** When does perceptron guarantee convergence? **A:** Only when data is linearly separable (a perfect linear boundary exists).
2. **Q:** What does the bias (b) do geometrically? **A:** It shifts the decision boundary; without b, boundary must pass through origin.
3. **Q:** Why is step activation bad for gradient-based training? **A:** Step is not smooth; gradient is zero or undefined, so learning by gradients fails.
4. **Q:** Why does pocket help in non-separable case? **A:** It stores the best weights seen so far, so it returns a good classifier even if updates never stop.
5. **Q:** What happens if you multiply $\mathbf{w}$ and b by 2 in perceptron? **A:** Sign doesn't change, so predictions stay same (scale doesn't matter for sign-based decisions).

B) KNN

6. **Q:** Why does KNN need scaling? **A:** Distance depends on feature magnitude; large-scale feature dominates, giving wrong “nearest”.
7. **Q:** What is the time cost of KNN prediction? **A:** Slow at prediction because it compares test point to many training points.
8. **Q:** What’s the difference between parametric and non-parametric model? **A:** Parametric learns fixed number of parameters (like $\mathbf{w}$); non-parametric (KNN) grows with data.

C) Linear Regression / Logistic Regression

9. **Q:** Why is MSE convex for linear regression? **A:** The loss is a quadratic function of $\mathbf{w}$, which forms a single bowl shape.
10. **Q:** Why cross-entropy is preferred for logistic regression? **A:** It matches probability modeling and strongly penalizes confident wrong predictions; gives better gradients.
11. **Q:** How do you turn logistic probability into a class label? **A:** Use threshold (often 0.5), but can be changed based on precision/recall needs.
12. **Q:** What is an “odds” interpretation in logistic regression (if asked)? **A:** Logistic regression models log-odds as linear in features (probability mapped through logit becomes linear).

D) SVM

13. **Q:** Why does minimizing $|\mathbf{w}|^2$ maximize margin? **A:** Margin is inversely proportional to $|\mathbf{w}|$; smaller norm → larger margin.
14. **Q:** What does C control in soft-margin SVM? **A:** Tradeoff between wide margin and fewer training violations.
15. **Q:** What does γ control in RBF kernel? **A:** How local similarity is. Large γ → very local, can overfit; small γ → smoother.
16. **Q:** Why are only support vectors important? **A:** They are the closest points; the optimal boundary depends mainly on them.

E) Neural Nets / Backprop / Optimization

17. **Q:** Why deep networks without activation are useless? **A:** Composition of linear layers is still linear → no extra power.
18. **Q:** Why does sigmoid cause vanishing gradients? **A:** For large $|s|$, sigmoid saturates near 0 or 1, gradients become tiny.
19. **Q:** Why SGD can generalize better than full GD sometimes? **A:** Noise in updates can prevent overfitting and help find flatter minima.
20. **Q:** What does L1 vs L2 do differently? **A:** L1 creates sparsity (feature selection), L2 smoothly shrinks weights.

F) RNN / LSTM / Attention / Transformer

21. **Q:** What exactly vanishes in "vanishing gradients" for RNN? **A:** Gradients passed back through many time steps become extremely small, so early tokens don't learn.
22. **Q:** What do LSTM gates do conceptually? **A:** Decide what to forget, what to store, and what to output.
23. **Q:** Why does attention use softmax? **A:** To convert scores into non-negative weights that sum to 1 (like "importance percentages").
24. **Q:** Why divide by $\sqrt{d_k}$ in attention? **A:** Prevent dot products from getting too large when dimension is big; stabilizes softmax and training.
25. **Q:** Why are Transformers parallelizable but RNNs not? **A:** Transformer processes all tokens together using attention; RNN depends on previous hidden state step by step.

G) Evaluation / Overfitting / Bounds / VC

26. **Q:** Why is "test set" sacred? **A:** It must represent unseen data; using it for tuning leaks information.
27. **Q:** What is the difference between model selection and model assessment? **A:** Selection = choose hyperparameters using validation; assessment = final performance using test.
28. **Q:** What does Hoeffding inequality tell in one sentence? **A:** With enough samples, training error is close to true error with high probability.
29. **Q:** What does VC dimension measure? **A:** Capacity/complexity: how many points can be fit in all possible ways.
30. **Q:** What happens to generalization bound when N increases? **A:** The complexity penalty term decreases → better guarantee.

H) Practical / Deployment

31. **Q:** If live performance drops, what is the first suspect? **A:** Data drift (input distribution changes), concept drift (label relationship changes), or leakage during evaluation.
32. **Q:** What is calibration? **A:** Whether predicted probability matches real frequency (e.g., among predictions of 0.8, about 80% are truly positive).

Prepared by MD. SHAHNAWAZ HOSSAN