Foundations of Machine Learning – Tutorial

MVS Sri Harsha

IIT Bombay

November 7, 2025

Outline

1. Linear Regression
2. Bias-Variance Tradeoff
3. Decision Trees
4. SVM
5. Clustering
6. Neural Networks
7. Regularization
8. Attention Mechanisms
9. End

Linear Regression – Problem 1

Consider a linear model of the form

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{i=1}^{D} w_i x_i$$

together with a sum-of-squares error function of the form

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2.$$

Now suppose that Gaussian noise ϵ_i with zero mean and variance σ^2 is added independently to each of the input variables x_i.
By making use of the properties

$$\mathbb{E}[\epsilon_i] = 0, \quad \text{and} \quad \mathbb{E}[\epsilon_i \epsilon_j] = \delta_{ij}\sigma^2,$$

show that minimizing $E_D(\mathbf{w})$ averaged over the noise distribution is equivalent to minimizing the sum-of-squares error for noise-free input variables, with the addition of a **weight-decay regularization term**, in which the bias parameter w_0 is omitted from the regularizer.

Linear Regression – Problem 1 (Derivation) — Part 1

Let the observed (noisy) inputs be $\tilde{\mathbf{x}}_n = \mathbf{x}_n + \boldsymbol{\epsilon}_n$ with $\boldsymbol{\epsilon}_n \sim \mathcal{N}(\mathbf{0}, \sigma^2 I)$, independent across n and features, and model $y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{i=1}^{D} w_i x_i$. Define $a_n := y(\mathbf{x}_n, \mathbf{w}) - t_n$. Then the noisy empirical loss is

$$E_D(\mathbf{w}; \tilde{X}) = \tfrac{1}{2} \sum_{n=1}^{N} \Big(a_n + \sum_{i=1}^{D} w_i \, \epsilon_{ni} \Big)^2 .$$

Taking expectation over the noise and using $\mathbb{E}\, \epsilon_{ni} = 0$, $\mathbb{E}\left(\epsilon_{ni}\epsilon_{nj}\right) = \delta_{ij}\sigma^2$:

$$\begin{aligned}
\mathbb{E}_\epsilon \, E_D(\mathbf{w}; \tilde{X}) &= \tfrac{1}{2} \sum_{n=1}^{N} \Big(a_n^2 + 2a_n \underbrace{\mathbb{E} \sum_{i} w_i \epsilon_{ni}}_{=0} + \mathbb{E}\Big(\sum_{i} w_i \epsilon_{ni} \Big)^2 \Big) \\
&= \tfrac{1}{2} \sum_{n=1}^{N} a_n^2 + \tfrac{1}{2} \sum_{n=1}^{N} \sum_{i=1}^{D} w_i^2 \, \mathbb{E}\, \epsilon_{ni}^2 \\
&= \tfrac{1}{2} \sum_{n=1}^{N} \big(y(\mathbf{x}_n, \mathbf{w}) - t_n \big)^2 + \frac{N\sigma^2}{2} \, \|\mathbf{w}_{1:D}\|_2^2 .
\end{aligned}$$

Linear Regression – Problem 1 (Derivation) — Part 2

Thus, minimizing the noise-averaged loss equals minimizing the noiseless sum-of-squares with an L2 penalty on $w_1, \ldots, w_D$ but not on the bias w_0.

If per-feature noise variances are unequal, $\mathrm{Var}(\epsilon_{ni}) = \sigma_i^2$, the penalty becomes diagonal Tikhonov: $\frac{N}{2} \sum_i \sigma_i^2 w_i^2$.

Intuition (plain words)

- Random input noise punishes large weights because it adds extra squared error proportional to their size.
- The bias weight does not multiply noise, so it is not penalized.
- Noisier features (bigger variance) get stronger penalties: trust them less.

Linear Regression – Problem 1: Simple

Idea

Adding small random jitters to each input feature makes models with huge weights look unstable: the noise wiggles predictions a lot if a weight is large. Averaging the squared error over the jitters shows an extra term that grows with the square of the weights. So training with noisy inputs is like training without noise plus a "keep weights small" penalty (except the bias, which never multiplies noise).

- Big weights amplify tiny input noise into big output error.
- Penalizing only non-bias weights matches how the noise enters the model.
- Feature with bigger noise variance gets a stronger shrink (trust it less).
- Moral: input noise naturally encourages simpler (smaller norm) solutions.

Linear Regression – Problem 2

Problem (clear statement): We have a dataset with design matrix $X \in \mathbb{R}^{N \times D}$ and responses $\mathbf{y} \in \mathbb{R}^N$ where the feature dimension D is much larger than the number of samples N $(D \gg N)$. Explain why ordinary least squares has infinitely many solutions in this setting, derive the minimum-Euclidean-norm solution, and state how regularization (ridge) resolves non-uniqueness and improves stability.

Linear Regression – Problem 2: Solution — Part 1

Setup: $X \in \mathbb{R}^{N\times D}$, $\mathbf{y} \in \mathbb{R}^N$, with $D \gg N$ (underdetermined). Consider OLS: $\min_{\mathbf{w}} \frac{1}{2}\|X\mathbf{w} - \mathbf{y}\|_2^2$.

- Normal equations: $X^\top X\,\mathbf{w} = X^\top \mathbf{y}$. The solution set is affine: $\mathbf{w} = \mathbf{w}_0 + \mathcal{N}(X)$.
- Minimum-norm OLS is unique and equals

$$\hat{\mathbf{w}}_{\min} = X^\top (XX^\top)^{-1}\mathbf{y} = X^+\mathbf{y},$$

the Moore–Penrose pseudoinverse. Via SVD $X = U\Sigma V^\top$ with rank r: $X^+ = V\Sigma^+ U^\top$, so $\hat{\mathbf{w}}_{\min} = V\Sigma^+ U^\top \mathbf{y}$.

Linear Regression – Problem 2: Solution — Part 2

- Ridge (Tikhonov):

$$\hat{\mathbf{w}}_{\text{ridge}} = (X^\top X + \lambda I)^{-1} X^\top \mathbf{y} = X^\top (X X^\top + \lambda I)^{-1} \mathbf{y}.$$

  The two forms are equivalent (Woodbury). As $\lambda \downarrow 0$, $\hat{\mathbf{w}}_{\text{ridge}} \to \hat{\mathbf{w}}_{\min}$ along the minimum-norm path.

- Dual/kernel (efficient when $D \gg N$): $\hat{\mathbf{w}}_{\text{ridge}} = X^\top \boldsymbol{\alpha}$ with $\boldsymbol{\alpha} = (K + \lambda I)^{-1} \mathbf{y}$, $K = X X^\top$.
- Practical: standardize features; do not penalize intercept w_0. Choose λ by CV; df$= \operatorname{tr}\left(K(K + \lambda I)^{-1}\right)$.

Intuition (plain words)

When $D \gg N$, infinitely many fits exist; pick the smallest-norm one. Ridge shrinks weights, gives uniqueness, and improves stability; the dual works in $N \times N$.

Linear Regression – Problem 2: Simple

Idea

When you have way more features than data points, countless weight vectors fit the training data perfectly. Choosing the one with the smallest overall size (minimum norm) is a simple tie-breaker that tends to generalize better. Ridge adds a soft push toward smaller weights, making the solution unique and less sensitive to tiny changes in data.

- Underdetermined: infinite perfect fits; minimum-norm picks the least "wiggly".
- Ridge: adds gentle shrink so equations become stably invertible.
- Dual view: operate in sample space (size N) instead of huge feature space.
- Practical: scale features, tune λ, skip penalizing the intercept.

Linear Regression – Problem 3

Problem (clear statement): Suppose one or more regressors in X are correlated with the noise term (endogeneity). Explain why OLS becomes biased/inconsistent. Provide a method to obtain consistent estimates (instrumental variables / 2SLS), state the assumptions instruments must satisfy, and give the 2SLS estimator.

Linear Regression – Problem 3: Endogeneity (Setup)

Model: $\mathbf{y} = X\mathbf{w} + \boldsymbol{\varepsilon}$ with $\mathbb{E}[\boldsymbol{\varepsilon} \mid X] \neq 0$ (e.g., omitted variables, simultaneity) makes OLS biased/inconsistent:

$$\hat{\mathbf{w}}_{\mathsf{OLS}} = \mathbf{w} + (X^\top X)^{-1} X^\top \boldsymbol{\varepsilon}, \quad \text{plim } \hat{\mathbf{w}}_{\mathsf{OLS}} = \mathbf{w} + Q_{xx}^{-1} q_{x\varepsilon},$$

with $Q_{xx} = \text{plim } \frac{1}{N} X^\top X$ and $q_{x\varepsilon} = \text{plim } \frac{1}{N} X^\top \boldsymbol{\varepsilon}$.

Key idea: use instruments Z correlated with X but exogenous to ε to purge the bias.

Linear Regression – Problem 3: IV/2SLS (Fix)

Instruments $Z \in \mathbb{R}^{N\times m}$ must satisfy:

- Relevance: $\mathrm{rank}(Z^\top X) = p$ (for p endogenous regressors).
- Exclusion: $\mathbb{E}[\boldsymbol{\varepsilon} \mid Z] = 0$.

Projection: $P_Z = Z(Z^\top Z)^{-1}Z^\top$, $M_Z = I - P_Z$; first stage $\hat{X} = P_Z X$. Estimators:

$$\hat{\mathbf{w}}_{\text{2SLS}} = (X^\top P_Z X)^{-1} X^\top P_Z \mathbf{y}, \qquad ext(exact - id)\ \hat{\mathbf{w}}_{\text{IV}} = (Z^\top X)^{-1} Z^\top \mathbf{y}.$$

Properties:

- Consistent under relevance + exclusion; $P_Z X$ spans valid variation in X.
- Diagnostics: first-stage F-stat > 10; Sargan/Hansen test if $m > p$. Weak instruments inflate variance/bias.

Intuition

OLS fails when a regressor co-moves with the error; IV uses only the instrument-induced component of X to estimate $\mathbf{w}$, removing the bias source.

Linear Regression – Problem 3: Simple

Idea

If a predictor moves together with the errors (confounded), OLS mixes up "signal" and "noise" and shifts the weight. An instrument is a helper variable that nudges the predictor but is itself unrelated to the errors. We first predict the bad predictor using instruments (clean part), then regress the target on that cleaned version.

- Endogeneity = predictor contaminated by hidden factors also affecting y.
- Instrument: relevant (linked to predictor), exogenous (independent of error).
- Two stages: (1) regress predictor on instruments, (2) regress y on predicted values.
- Weak instruments act like a blurry lens: low power and remaining bias.

Bias-Variance – Problem 1

Problem (clear statement): For a supervised regression problem with $y = f(\mathbf{x}) + \epsilon$, show the bias–variance decomposition for squared loss. Explain the three terms (bias^2, variance, irreducible noise) in plain words and write the formula for expected test MSE.

Bias-Variance – Problem 1: Decomposition — Part 1

Assume $y = f(\mathbf{x}) + \epsilon$ with $\mathbb{E}[\epsilon \mid \mathbf{x}] = 0$, $\mathrm{Var}(\epsilon \mid \mathbf{x}) = \sigma^2$. Let $\hat{f}$ be a predictor learned from a random training set $\mathcal{D}$. For squared loss,

$$\mathbb{E}_{\mathcal{D},\epsilon}\big[(\hat{f}(\mathbf{x}) - y)^2 \mid \mathbf{x}\big] = \underbrace{\big(\mathbb{E}_{\mathcal{D}}[\hat{f}(\mathbf{x})] - f(\mathbf{x})\big)^2}_{\text{Bias}^2} + \underbrace{\mathrm{Var}_{\mathcal{D}}(\hat{f}(\mathbf{x}))}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible}} .$$

Integrating over $p(\mathbf{x})$ gives the expected test MSE. For linear regression with $\hat{f}(\mathbf{x}) = \mathbf{x}^\top \hat{\mathbf{w}}$ (centered),

$$\mathrm{Var}_{\mathcal{D}}(\hat{f}(\mathbf{x})) = \mathbf{x}^\top \, \mathrm{Var}(\hat{\mathbf{w}}) \, \mathbf{x}.$$

Bias-Variance – Problem 1: Decomposition — Part 2

With homoscedastic noise and fixed design, $\mathrm{Var}(\hat{\mathbf{w}}) = \sigma^2(X^\top X)^{-1}$ for OLS. Regularization trades bias for reduced variance.

Intuition (plain words)

- Error $=$ what you systematically miss (bias) $+$ how much your answer wiggles across datasets (variance) $+$ irreducible noise.
- Simpler models: higher bias, lower variance. Complex models: lower bias, higher variance.
- Regularization intentionally adds a bit of bias to dramatically cut variance.

Bias-Variance – Problem 1: Simple

Idea

Test error has three parts: a systematic miss (bias), wobbliness across different training sets (variance), and background noise you can't beat. Regularization accepts a tiny systematic miss to greatly reduce wobble.

- Bias: using a too-simple shape to describe a complex curve.
- Variance: overreacting to small quirks in the training sample.
- Regularization: sand off sharp corners so small data changes don't swing predictions.

Bias-Variance – Problem 2

Problem (clear statement): Explain the "double descent" phenomenon: how test risk behaves as model complexity (or the ratio $\gamma = D/N$) increases past the interpolation threshold. Describe qualitatively why a second descent can occur and what role implicit/explicit regularization plays.

Bias-Variance – Problem 2: Double Descent

Let $\gamma = D/N$ denote the aspect ratio. Around the interpolation threshold $\gamma \approx 1$, variance can blow up and then decrease for $\gamma > 1$ with appropriate inductive bias (e.g., min-norm). For isotropic Gaussian design and ridgeless regression, the test risk exhibits a peak near $\gamma = 1$ and then "second descent". Canonical expressions (informal): for $\gamma < 1$,

$$\mathbb{E}\,\|\hat{\mathbf{w}} - \mathbf{w}^*\|_2^2 \approx \frac{\sigma^2\,\gamma}{1-\gamma},$$

while for $\gamma > 1$, min-norm interpolation yields

$$\mathbb{E}\,\|\hat{\mathbf{w}} - \mathbf{w}^*\|_2^2 \approx \left(1 - \tfrac{1}{\gamma}\right)\|\mathbf{w}^*\|_2^2 + \frac{\sigma^2\,\gamma}{\gamma - 1}.$$

Ridge with $\lambda > 0$ smooths the peak; choose λ to minimize risk. The phenomenon generalizes beyond linear models via effective dimension and implicit bias.

Intuition (plain words)

- Near the just-fit point (parameters $\approx$ data), tiny noise gets massively amplified.
- Beyond that, picking the minimum-norm interpolator acts as a strong simplicity bias, reducing test error again.

Bias-Variance – Problem 2: Simple

Idea

As model size grows to just fit the data, tiny noise can explode the fit (big bump). If you keep growing the model but choose the simplest exact fit (minimum-norm) or add a touch of regularization, test error can drop again—hence a second descent.

- Peak near "just-enough parameters": the model is twitchy there.
- Beyond the peak, the min-norm choice acts like a built-in simplicity rule.
- A bit of ridge smooths the bump and is easier to tune.

Decision Trees – Problem 1

Problem (clear statement): Define information gain for classification splits. Given a node with dataset S, show how to compute the entropy before and after a split on attribute A, and explain why maximizing information gain prefers purer child nodes.

Decision Trees – Problem 1: Information Gain — Part 1

Let labels $Y \in \{1, \ldots, K\}$ with empirical distribution $p_k = \frac{|S_k|}{|S|}$. Entropy: $H(S) = -\sum_{k=1}^{K} p_k \log p_k$. For a split on attribute A with values $v \in \mathcal{V}$, the information gain is

$$\mathrm{IG}(S, A) = H(S) - \sum_{v \in \mathcal{V}} \frac{|S_v|}{|S|} H(S_v) = I(Y; A),$$

the empirical mutual information between Y and A. For continuous features, thresholds t define binary splits $A = \not\Vdash\{x_j \leq t\}$.

Decision Trees – Problem 1: Information Gain — Part 2

Caveat: IG is biased toward attributes with many values; gain ratio (C4.5) normalizes by split intrinsic information. Gini index alternative: $G(S) = \sum_k p_k(1 - p_k)$; minimizing weighted post-split Gini often approximates maximizing IG and is cheaper to compute.

Intuition (plain words)

- Good splits make children purer (labels more predictable).
- IG is "how much uncertainty we remove" by splitting on an attribute.
- Beware attributes with many categories; they can look artificially good.

Decision Trees – Problem 1: Simple

Idea

Entropy measures how mixed the labels are; a good split makes children less mixed. Information gain is simply "how much uncertainty we removed" by splitting on a feature.

- Many-valued features can look unfairly good—normalize (gain ratio) or use Gini.
- Continuous features: try thresholds to separate groups cleanly.
- We prefer splits that make each child mostly one label.

Decision Trees – Problem 2

Problem (clear statement): For gradient-boosted trees, derive the second-order approximate objective used to pick tree structure and leaf weights. Show how to compute the optimal weight for a leaf and the formula used to compute the gain of a candidate split.

Decision Trees – Problem 2: XGBoost — Part 1 (objective)

Second-order objective around current predictions $\hat{y}^{(m)}$:

$$\mathcal{L}^{(m)} \approx \sum_{i=1}^{N} \Big[g_i \, f_m(\mathbf{x}_i) + \tfrac{1}{2} h_i \, f_m(\mathbf{x}_i)^2 \Big] + \Omega(f_m), \quad \Omega(f_m) = \gamma T_m + \tfrac{\lambda}{2} \sum_{j=1}^{T_m} w_j^2.$$

Definitions: $g_i = \partial_{\hat{y}} \, \ell(y_i, \hat{y}_i^{(m)})$, $h_i = \partial_{\hat{y}}^2 \, \ell(y_i, \hat{y}_i^{(m)})$. For a fixed tree structure with leaf index $q(\cdot)$ we will aggregate gradients/curvatures per leaf.

Decision Trees – Problem 2: XGBoost — Part 2 (leaf weights)

Aggregates for leaf j with index $q(\cdot)$:

$$G_j = \sum_{i:q(\mathbf{x}_i)=j} g_i, \quad H_j = \sum_{i:q(\mathbf{x}_i)=j} h_i.$$

Optimal leaf weight and its contribution to the objective:

$$w_j^* = -\frac{G_j}{H_j + \lambda}, \qquad \mathcal{L}_j^* = -\tfrac{1}{2}\frac{G_j^2}{H_j + \lambda} + \gamma.$$

Notes:

- λ (L2) stabilizes w_j when H_j is small; γ penalizes adding leaves.
- Only G_j and H_j are needed to score a structure after computing g_i, h_i.

Decision Trees – Problem 2: XGBoost — Part 3 (split gain + logistic)

For a candidate split of a leaf into Left/Right children, the gain is

$$\text{Gain} = \tfrac{1}{2}\left(\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right) - \gamma.$$

Logistic loss example (binary, raw scores $\hat{y}$): with $p_i = \sigma(\hat{y}_i^{(m)})$,

$$g_i = p_i - y_i, \qquad h_i = p_i(1-p_i) \in [0, \tfrac{1}{4}].$$

Regularization and training tips:

- Depth/leaf penalties: γ discourages tiny noisy splits; set via validation.
- λ (and optional ℓ_1) shrink leaf weights; improves stability.
- Shrinkage η and row/column subsampling reduce variance and overfitting.

Decision Trees – Problem 2: XGBoost — Part 4

Intuition (plain words)

- Fit a small tree to residual signals (g_i), temper using curvature (h_i) and regularizers.
- A good split explains gradients better than the parent, by more than its complexity cost (γ).
- Learning rate and subsampling make boosting conservative and robust.

Decision Trees – Problem 2: Simple

Idea

Boosting builds one small tree at a time to fix what the current model gets wrong. We look at gradients (what direction to fix each point) and curvatures (how confident to move). A split is good if the two children explain the gradients better than the parent, after paying a small complexity fee.

- Leaf weight = average gradient divided by (curvature + shrink), with a minus sign.
- Split gain = left score + right score - parent score - penalty.
- Regularization (λ, γ), shrinkage (η), and subsampling keep it stable.

SVM – Problem 1

Problem (clear statement): Derive the maximum-margin classifier for linearly separable data: write the hard-margin primal optimization problem, derive the dual, and explain the role of support vectors and KKT conditions in plain words.

SVM – Problem 1: Maximum Margin

Hard-margin primal (linearly separable):

$$\min_{\mathbf{w},b} \tfrac{1}{2}\|\mathbf{w}\|_2^2 \quad \text{s.t. } y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1,\ i = 1, \dots, N.$$

Soft-margin with slack $\xi_i \geq 0$: add $C \sum_i \xi_i$ and constraints $y_i(\cdot) \geq 1 - \xi_i$. Dual (soft-margin):

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{N} \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j\, \mathbf{x}_i^\top \mathbf{x}_j, \text{ s.t. } 0 \leq \alpha_i \leq C,\ \sum_i \alpha_i y_i = 0.$$

Solution: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$, decision $\operatorname{sign}(\mathbf{w}^\top \mathbf{x} + b)$. KKT conditions identify support vectors (those with $\alpha_i > 0$).

Intuition (plain words)

- Maximize the safety margin between classes; wider margins generalize better.
- Only boundary points (support vectors) matter; others don't affect $\mathbf{w}$.
- Slack and C let us trade training errors for larger margins.

SVM – Problem 1: Simple

Idea

Draw a line that separates classes and leaves the biggest possible safety gap to the closest points. Only the closest points (support vectors) decide where the line ends up; everything far away doesn't matter. Allow a few violations when data aren't perfectly separable.

- Wider margin = more cushion against noise.
- Support vectors are the "boundary coaches"—move them, and the line moves.
- The C parameter trades margin width for training mistakes.

SVM – Problem 2

Problem (clear statement): Explain the kernel trick: how replacing inner products by a kernel function lets us work in high- or infinite-dimensional feature spaces implicitly. State Mercer's condition (intuitively) for a valid kernel and why positive semidefiniteness of the Gram matrix matters.

SVM – Problem 2: Kernels — Part 1

Kernel trick replaces inner products by $k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$ in the dual. Decision function:

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i y_i \, k(\mathbf{x}_i, \mathbf{x}) + b.$$

Mercer: k valid iff for any finite set $\{\mathbf{x}_i\}$, Gram matrix $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is symmetric PSD.

SVM – Problem 2: Kernels — Part 2

PSD $\Rightarrow$ exists (possibly infinite-dim) ϕ with $k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$. Mercer eigen-expansion on compact domain: $k(\mathbf{x}, \mathbf{z}) = \sum_{\ell \geq 1} \lambda_\ell \psi_\ell(\mathbf{x}) \psi_\ell(\mathbf{z})$, $\lambda_\ell \geq 0$. Practical test: build K and check eigenvalues ≥ 0. Common kernels: linear; polynomial $(\mathbf{x}^\top \mathbf{z} + c)^p$; RBF. Non-PSD similarities can break convexity (ill-posed dual).

SVM – Problem 2: Kernels — Part 3 (Intuition)

Intuition

- Kernels give dot-products in feature space without explicit mapping (computational + design benefit).
- PSD ensures geometry like Euclidean inner product & convex optimization stability.
- Choose kernels to encode task-relevant similarity (e.g., RBF for locality, polynomial for feature interactions).

SVM – Problem 2: Simple

Idea

Instead of manually creating lots of features, use a kernel to compare points as if we had those features already. Valid kernels behave like dot-products in some space, keeping the math convex and stable.

- Linear: similarity by straight-line projection.
- Polynomial: adds interactions (like ANDs of features).
- RBF: points are similar if they're close; set how local with γ.

SVM – Problem 3

Problem (clear statement): Describe the Gaussian RBF kernel, its parameters, and the intuitive effect of the kernel width on the classifier. Explain why RBF corresponds to a localized similarity measure and mention pros/cons.

SVM – Problem 3: RBF Kernel

Gaussian RBF kernel:

$$k(\mathbf{x}, \mathbf{z}) = \exp\Big(-\frac{\|\mathbf{x}-\mathbf{z}\|_2^2}{2\sigma^2}\Big) = \exp\big(-\gamma\,\|\mathbf{x}-\mathbf{z}\|_2^2\big),\ \gamma = \tfrac{1}{2\sigma^2}.$$

It corresponds to an infinite-dimensional feature map with localized similarity. Hyperparameters (C, γ) control margin-violations vs. smoothness (effective radius). Scaling features is crucial; cross-validate (C, γ).

Intuition (plain words)

- RBF says two points are similar if they're close in Euclidean distance.
- γ sets how "local" similarity is; larger γ fits more wiggly boundaries.
- Tune (C, γ) together; always scale features first.

SVM – Problem 3: Simple

Idea

RBF acts like drawing soft bubbles around support vectors: nearby points vote strongly, far points barely vote. The width γ controls bubble size; C controls how strictly we separate vs. allow some errors.

- Larger γ = smaller bubbles = detailed, wiggly boundaries (risk overfit).
- Smaller γ = larger bubbles = smoother boundaries (risk underfit).
- Always scale features; tune (C, γ) by validation.

Clustering – Problem 1

Problem (clear statement): Prove why Lloyd's k-means algorithm converges: show that each assignment and update step does not increase the k-means objective, and explain why this implies convergence to a local minimum.

Clustering – Problem 1: k-means Convergence — Part 1

Objective (within-cluster SSE):

$$J(\{\mu_k\}, c) = \sum_{i=1}^{N} \|\mathbf{x}_i - \mu_{c(i)}\|_2^2 = \sum_{k=1}^{K} \sum_{i \in C_k} \|\mathbf{x}_i - \mu_k\|_2^2.$$

Lloyd's algorithm alternates:

1. Assignment: $c(i) \leftarrow \arg\min_{1 \le k \le K} \|\mathbf{x}_i - \mu_k\|^2$.
2. Update: $\mu_k \leftarrow \frac{1}{|C_k|} \sum_{i \in C_k} \mathbf{x}_i$ (cluster mean).

Clustering – Problem 1: k-means Convergence — Part 2

Why it converges (explicit):

- Given centers, the assignment step chooses the closest center for each i, so J cannot increase.
- Given assignments, updating μ_k to the mean minimizes $\sum_{i \in C_k} \|\mathbf{x}_i - \mu_k\|^2$, so J cannot increase.
- Therefore J is monotonically nonincreasing and bounded below by 0, hence converges. With finitely many assignments, the algorithm reaches a fixed point (a local minimum).

Notes: handle ties deterministically; if a cluster becomes empty, reinitialize (e.g., worst-error point). Use k-means++ and restarts for better optima.

Intuition (plain words)

Each step can only improve or keep SSE the same, so the process settles at a local optimum.

Clustering – Problem 1: Simple

Idea

K-means is a tidy-up loop: put each point with the closest center, then move centers to the average of their members. Each tidy-up step can only improve or keep the score, so eventually nothing changes—converged.

- Use k-means$++$ starts and a few restarts to avoid bad local minima.
- If a cluster empties, re-seed it on a far or high-error point.
- Standardize features; Euclidean distance is scale sensitive.

Clustering – Problem 2

Problem (clear statement): For Gaussian mixture models, explain the relation between the log-likelihood, the ELBO, and the KL divergence. Show how EM increases the log-likelihood monotonically and how the E- and M-steps are derived from the ELBO.

Clustering – Problem 2: KL Divergence and EM

For a GMM with parameters Θ, the log-likelihood satisfies

$$\log p(\mathbf{X} \mid \Theta) = \mathcal{F}(q, \Theta) + \mathrm{KL}\big(q(\mathbf{Z}) \,\|\, p(\mathbf{Z} \mid \mathbf{X}, \Theta)\big),$$

with ELBO $\mathcal{F}(q, \Theta) = \mathbb{E}_q[\log p(\mathbf{X}, \mathbf{Z} \mid \Theta)] - \mathbb{E}_q[\log q(\mathbf{Z})]$. EM alternates $q \leftarrow p(\mathbf{Z} \mid \mathbf{X}, \Theta)$ (E-step) and $\Theta \leftarrow \arg\max_\Theta \mathcal{F}(q, \Theta)$ (M-step), monotonically increasing $\log p(\mathbf{X} \mid \Theta)$. As $\sigma^2 \to 0$ in isotropic GMMs, EM approaches k-means.

Intuition (plain words)

- EM alternates: "guess soft assignments" (E) and "re-fit parameters" (M) to improve a tight lower bound on log-likelihood.
- The KL term measures how close our soft assignments are to the true posteriors.
- With tiny variances, soft assignments become hard and EM behaves like k-means.

Clustering – Problem 2: Simple

Idea

EM is guess-and-improve: guess soft cluster memberships (E), then update cluster parameters to fit those guesses (M). This always raises a lower bound on the true likelihood, so you steadily improve until you stall.

- E-step: responsibilities $=$ how much each point belongs to each Gaussian.
- M-step: re-estimate means, covariances, and mixing weights from responsibilities.
- With tiny variances, soft assignments harden and EM behaves like k-means.

Clustering – Problem 3

Problem (clear statement): Define common clustering evaluation metrics. For each metric (Silhouette, Mutual Information / NMI, Adjusted Rand Index), give the formula, intuitive interpretation, and one advantage and limitation.

Clustering – Problem 3: Metrics — Part 1 (Internal)

Silhouette (point i): $a_i =$ mean intra-cluster distance; $b_i =$ min mean distance to other clusters. Score $s_i = \frac{b_i - a_i}{\max\{a_i, b_i\}} \in [-1, 1]$; overall $S = \frac{1}{N} \sum_i s_i$. Purity: $\frac{1}{N} \sum_v \max_u n_{uv}$ (biased to many small clusters). Other: Calinski–Harabasz (higher better), Davies–Bouldin (lower better).

Clustering – Problem 3: Metrics — Part 2 (External MI/ARI/FMI)

Mutual Information: $\mathrm{MI}(U,V)=\sum_{u,v} p(u,v)\log\frac{p(u,v)}{p(u)p(v)}$. Normalizations: $\mathrm{NMI}_{\max}=\frac{\mathrm{MI}}{\max(H(U),H(V))}$, $\mathrm{NMI}_{\surd}=\frac{\mathrm{MI}}{\sqrt{H(U)H(V)}}$. Adjusted MI: $\mathrm{AMI}=\frac{\mathrm{MI}-\mathbb{E}[\mathrm{MI}]}{\max\{H(U),H(V)\}-\mathbb{E}[\mathrm{MI}]}$. Adjusted Rand Index (ARI):

$$\mathrm{ARI}=\frac{\sum_{u,v}\binom{n_{uv}}{2}-\frac{\sum_u\binom{n_{u\cdot}}{2}\sum_v\binom{n_{\cdot v}}{2}}{\binom{N}{2}}}{\frac{1}{2}\left[\sum_u\binom{n_{u\cdot}}{2}+\sum_v\binom{n_{\cdot v}}{2}\right]-\frac{\sum_u\binom{n_{u\cdot}}{2}\sum_v\binom{n_{\cdot v}}{2}}{\binom{N}{2}}}.$$

Fowlkes–Mallows (FMI): $\mathrm{FMI}=\sqrt{\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}\cdot\frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}}$.

Clustering – Problem 3: Metrics — Part 3 (Intuition)

Intuition

- Silhouette: compares cohesion vs. separation per point.
- ARI/AMI: adjust for chance; robust to number of clusters.
- Purity inflates with many clusters; use adjusted metrics for fair comparison.
- Match metric to data scale & label availability; standardize features when using distance-based metrics.

Clustering – Problem 3: Metrics (continued)

Pair-counting view for ARI: let $a = \sum_{u,v} \binom{n_{uv}}{2}$ (pairs in same cluster in both), $b = \sum_u \binom{n_{u\cdot}}{2} - a$ (same in truth only), $c = \sum_v \binom{n_{\cdot v}}{2} - a$ (same in predicted only), $d = \binom{N}{2} - a - b - c$ (different in both). Rand Index $= \frac{a+d}{\binom{N}{2}}$, ARI adjusts RI by subtracting its expected value under random labeling. Use ARI for unbalanced classes; NMI when number of clusters differs.

Clustering – Problem 3: Simple

Idea

Pick metrics that match what you know and what you have. With labels, prefer chance-adjusted comparisons (ARI/AMI). Without labels, use silhouettes or within/between dispersion—mind feature scales.

- Silhouette: good if distance makes sense; standardize first.
- ARI: pairwise agreement, adjusted for random chance; robust to cluster count.
- AMI/NMI: information shared between true and predicted clusters.

Neural Networks – Problem 1

Problem (clear statement): List common activation functions used in neural networks. For each, write its formula and derivative, and explain in simple words when it helps (advantages) and what problems it can cause (disadvantages).

Neural Networks – Problem 1: Activations — Part 1

Common activations (formula + derivative):

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$, $\sigma'(z) = \sigma(z)(1-\sigma(z))$.
- Tanh: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, $\tanh'(z) = 1 - \tanh^2(z)$.
- ReLU: $\mathrm{ReLU}(z) = \max(0, z)$, $\mathrm{ReLU}'(z) = \begin{cases} 1, & z > 0 \\ 0, & z < 0 \end{cases}$; subgradient at 0: $[0, 1]$.
- Leaky ReLU: $\max(\alpha z, z)$, derivative $= \begin{cases} 1, & z > 0 \\ \alpha, & z < 0 \end{cases}$.
- GELU: $\mathrm{GELU}(z) = z\,\Phi(z)$ (approx: $0.5z(1 + \tanh(\sqrt{2/\pi}(z + 0.0447z^3)))$).

Neural Networks – Problem 1: Activations — Part 2

Universal approximation: non-polynomial activation + one hidden layer approximates continuous functions (compact set). Lipschitz: if each layer has operator norm $\|W_l\|$ and activation is 1-Lipschitz, whole net is $\prod_l \|W_l\|$-Lipschitz.

Intuition (plain words)

- ReLU: fast, sparse; risk of dead neurons (mitigate with leaky variants).
- Sigmoid/tanh: good for bounded outputs; saturation slows learning (vanishing gradients).
- GELU: smoother stochastic gating; empirically strong in Transformers.
- Pick activations balancing gradient flow, expressivity, and computational cost.

Neural Networks – Problem 1: Simple

Idea

Activations shape how neurons respond. ReLU is like a gate that passes positives, blocks negatives; sigmoid/tanh squeeze values to a small range (can saturate); GELU is a smooth, probabilistic gate.

- ReLU: fast, sparse signals; watch out for dead units (use leaky).
- Sigmoid/tanh: bounded outputs; slower when saturated (tiny gradients).
- GELU: smoother than ReLU; often strong in Transformers.

Neural Networks – Problem 2

Problem (clear statement): For optimization with gradient descent, state convergence bounds for (a) strongly convex, smooth functions and (b) general nonconvex smooth functions. Explain what these bounds mean in plain words (how fast the method reduces error or gradient norm).

Neural Networks – Problem 2: GD Iteration Bounds

For L-smooth, μ-strongly convex f, gradient descent with $\eta \in (0, 2/L)$ satisfies

$$\|\mathbf{w}^{(t)} - \mathbf{w}^*\|_2 \leq \left(1 - \eta\mu\right)^t \|\mathbf{w}^{(0)} - \mathbf{w}^*\|_2, \quad t \geq 0.$$

Thus $t \geq \frac{1}{\eta\mu} \log \frac{\|\mathbf{w}^{(0)} - \mathbf{w}^*\|}{\varepsilon}$ iterations to reach error ε. For nonconvex L-smooth f, with fixed $\eta \leq 1/L$,

$$\min_{0 \leq t < T} \|\nabla f(\mathbf{w}^{(t)})\|_2^2 \leq \frac{2L}{T}\big(f(\mathbf{w}^{(0)}) - f_{\inf}\big),$$

so gradient norm decays as $\mathcal{O}(1/\sqrt{T})$ in stochastic settings and $\mathcal{O}(1/T)$ in deterministic.

Intuition (plain words)

- With curvature ($\mu > 0$), fixed-step GD shrinks distance to optimum at a constant rate.
- Without convexity, you still make progress: gradients get smaller on average.
- Step sizes must respect smoothness (L) to avoid divergence.

Neural Networks – Problem 2: Simple

Idea

With some curvature (strong convexity), each gradient step cuts your error by a fixed fraction—steady progress. Without convexity, you can still guarantee the gradients get small on average—slower, but moving.

- Step size must respect smoothness to avoid bouncing or diverging.
- Nonconvex: aim for small gradients; restarts and schedules can help.

Neural Networks – Problem 3

Problem (clear statement): Write the Adam optimizer update rules (first and second moment estimates and bias corrections). Explain intuitively why bias correction is used and what AdamW changes about weight decay.

Neural Networks – Problem 3: Adam Optimizer

Adam maintains first/second moment estimates of gradients $g_t = \nabla f(\mathbf{w}_t)$:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t, & \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t \odot g_t, & \hat{v}_t &= \frac{v_t}{1-\beta_2^t}, \\ \mathbf{w}_{t+1} &= \mathbf{w}_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}. \end{aligned}$$

Bias correction $(\hat{m}_t, \hat{v}_t)$ is essential at early steps. Typical: $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. AdamW decouples weight decay: $\mathbf{w} \leftarrow (1-\eta\lambda)\mathbf{w} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon}$.

Intuition (plain words)

- Adam smooths gradients (momentum) and scales steps per-parameter (RMS), adapting to curvature.
- Early steps need bias-correction to avoid underestimation.
- AdamW's decoupled decay is usually better than adding $\lambda\mathbf{w}$ to the gradient.

Neural Networks – Problem 3: Simple

Idea

Adam keeps moving averages of gradients (momentum) and of their squares (how noisy each direction is). It corrects early-step bias and takes bigger steps where gradients are stable and smaller where they're noisy.

- Bias correction = remove startup lag in the moving averages.
- AdamW decouples weight decay: shrink weights separately from the gradient.
- Defaults work surprisingly well; still tune the learning rate.

Regularization – Problem 1

Problem (clear statement): Explain what inductive bias is in machine learning. Give examples of how regularization (priors or penalties) encodes inductive bias, and state the connection between Bayesian priors and common penalties (e.g., Gaussian prior $\rightarrow$ L2).

Regularization – Problem 1: Inductive Bias

Inductive bias defines preferences among hypotheses. Regularization encodes bias via penalties or constraints. Bayesian view: prior $p(\mathbf{w})$ induces a penalty $-\log p(\mathbf{w})$. For linear-Gaussian models with $\mathbf{y} \mid X, \mathbf{w} \sim \mathcal{N}(X\mathbf{w}, \sigma^2 I)$ and prior $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \tau^2 I)$, MAP equals ridge:

$$\hat{\mathbf{w}}_{\text{MAP}} = \arg\min_{\mathbf{w}} \tfrac{1}{2}\|X\mathbf{w} - \mathbf{y}\|_2^2 + \frac{\sigma^2}{2\tau^2}\|\mathbf{w}\|_2^2.$$

Sparsity bias arises from Laplace prior leading to LASSO (ℓ_1 penalty).

Intuition (plain words)

- Regularization encodes preferences: smaller weights, fewer nonzeros, smoother functions, etc.
- Bayesian view: penalties are just negative log-priors.
- Choose the bias that matches what you expect about the true function.

Regularization – Problem 1: Simple

Idea

Inductive bias is your model's preference: "I like small weights" or "I assume only a few features matter." Adding penalties or priors nudges training toward those preferences so it doesn't chase noise.

- L2 = prefer smooth, spread-out small weights.
- L1 = prefer sparse weights (few nonzeros).
- Prior view: penalties are just saying what kinds of solutions we believe in.

Regularization – Problem 2

Problem (clear statement): Define weight decay (L2 regularization). Show the gradient descent update with weight decay and explain in simple terms why it reduces overfitting and how it relates to a Gaussian prior.

Regularization – Problem 2: Weight Decay

With L2 regularization $\frac{\lambda}{2}\|\mathbf{w}\|_2^2$, gradient descent update is

$$\mathbf{w} \leftarrow \mathbf{w} - \eta(\nabla_{\mathbf{w}}\mathcal{L} + \lambda\mathbf{w}) = (1 - \eta\lambda)\mathbf{w} - \eta\nabla_{\mathbf{w}}\mathcal{L},$$

shrinking weights each step (excluding the bias improves fit). In Bayesian linear regression, L2 corresponds to Gaussian prior; posterior mean equals ridge solution. Early stopping can approximate an implicit L2 penalty.

Intuition (plain words)

- Weight decay gently pulls parameters toward zero to reduce variance.
- Don't decay the bias—it doesn't multiply inputs and shouldn't be shrunk.
- Early stopping often behaves like adding an L2 penalty.

Regularization – Problem 2: Simple

Idea

Weight decay gently scales weights down every step, preventing any single feature from dominating and making the model less jittery. It's like a soft leash keeping parameters near zero unless data really justify them.

- Shrinking reduces variance (less overfit wiggle).
- Don't shrink the bias—its job is different.
- Early stopping can act like implicit decay by limiting growth time.

Regularization – Problem 3

Problem (clear statement): Describe residual connections (skip connections) used in deep networks. Explain how they help training (gradient flow intuition) and give the simple backward expression that shows gradients can pass through the identity path.

Regularization – Problem 3: Residual Connections

Residual block: $\mathbf{x}_{l+1} = \mathbf{x}_l + F(\mathbf{x}_l; \theta_l)$. Backprop:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{l+1}} \Big(I + \frac{\partial F}{\partial \mathbf{x}_l} \Big),$$

so gradients can flow through the identity, mitigating vanishing. As a forward Euler discretization of $\frac{d\mathbf{x}}{dt} = F(\mathbf{x}, t)$, deeper nets resemble ODE solvers; stability benefits from Lipschitz control on F.

Intuition (plain words)

- Residuals give highways for gradients and features—making very deep nets trainable.
- Small residuals mean "close to identity," which is easier to learn and stabilize.
- Think of depth as time; residuals are like stepping an ODE forward.

Regularization – Problem 3: Simple

Idea

Skip connections are hallways that let information and gradients bypass complex transformations. If the fancy part of the block doesn't help, the network can lean on the identity path—making very deep nets trainable.

- Prevent vanishing: gradients have a shortcut.
- Easier to learn tweaks (residuals) than full transformations.
- Think of depth as time steps in an evolving system.

Attention – Problem 1

Problem (clear statement): For scaled dot-product attention, write the formula and compute the time and memory complexity in terms of sequence length n and feature dimension d. Explain why attention becomes the bottleneck for long sequences.

Attention – Problem 1: Complexity

Scaled dot-product attention with queries $Q \in \mathbb{R}^{n \times d_k}$, keys $K \in \mathbb{R}^{n \times d_k}$, values $V \in \mathbb{R}^{n \times d_v}$:

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

Time: $\mathcal{O}(n^2 d_k + n^2 + n^2 d_v) = \mathcal{O}(n^2 d)$; memory: $\mathcal{O}(n^2)$ for the attention matrix. This quadratically scales with sequence length n and is the main bottleneck. Implementation note: fused kernels like FlashAttention compute softmax-attention in tiles to reduce memory reads/writes and avoid materializing the full $n \times n$ matrix, lowering memory without changing asymptotic time.

Intuition (plain words)

- Full attention compares every token with every other—hence the n^2 cost.
- Memory, not just compute, is the bottleneck; tiling/fused kernels reduce traffic.
- Long sequences need smarter patterns or approximations.

Attention – Problem 1: Simple

Idea

Full attention makes every token talk to every other token—both computing and storing all those conversations grows with the square of length. Memory becomes the choke point; tiled kernels keep the same math but reduce traffic.

- n^2 pairs $=$ heavy for long sequences.
- Memory, not just FLOPs, limits batch sizes.

Attention – Problem 2

Problem (clear statement): Define self-attention, cross-attention, and multi-head attention. For multi-head attention, write the shapes of the projection matrices and explain why multiple heads are useful.

Attention – Problem 2: Types

Self-attention: queries, keys, values from the same sequence; cross-attention: queries from target, keys/values from source. Multi-head with H heads:

$$\mathrm{MHA}(X) = \mathrm{Concat}(\text{head}_1, \ldots, \text{head}_H)W^O, \ \text{head}_h = \mathrm{Attn}(XW_h^Q, XW_h^K, XW_h^V).$$

Shapes: $W_h^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, similarly for W_h^K, W_h^V; typically $Hd_k = Hd_v = d_{\text{model}}$.

Intuition (plain words)

- Self-attention mixes information within a sequence; cross-attention lets one sequence query another.
- Multiple heads learn different relation types in parallel.
- Head sizes typically split the model dimension evenly.

Attention – Problem 2: Simple

Idea

Self-attention lets words in a sentence look at each other; cross-attention lets a decoder look at the encoder. Multiple heads are different "views"—one might focus on nearby words, another on long-range matches.

- Heads split the model dimension into parallel subspaces.
- Different heads specialize in different patterns.

Attention – Problem 3

Problem (clear statement): List and briefly explain common methods to approximate attention to reduce complexity (kernel/linear attention, sparse/local attention, Nyström). For each, state the resulting time complexity and a trade-off.

Attention – Problem 3: Approximate Attention

Low-rank/structured approximations reduce $\mathcal{O}(n^2)$ to near-linear:

- Kernelized/linear attention: approximate $\exp(QK^\top) \approx \phi(Q)\phi(K)^\top$ with feature map ϕ, yielding $\mathcal{O}(nd^2)$ or $\mathcal{O}(nd)$ time.
- Sparse/Local: restrict attention to a banded neighborhood; complexity $\mathcal{O}(n\,w)$ with window w.
- Nyström/landmarks: $K \approx CW^\dagger C^\top$ using $r \ll n$ landmarks; cost $\mathcal{O}(nr)$.

Approximation error depends on rank/sparsity of attention patterns; choose r or w based on accuracy-budget trade-off.

Intuition (plain words)

- Many attention matrices are near low-rank or local—use that structure to cut cost.
- Kernel tricks turn softmax into dot-products in a feature space; sparse windows limit who can attend to whom.
- Tune ranks/windows to balance speed and accuracy.

Attention – Problem 3: Simple

Idea

To scale beyond n^2, either summarize interactions (low-rank landmarks), limit who talks to whom (local windows), or rewrite the softmax so it becomes a chain of dot-products (kernel features).

- Landmark/Nystr"om: talk to a small set of representatives.
- Local/sparse: only nearby tokens exchange messages.
- Kernelized: approximate softmax via feature maps to avoid building $n \times n$.

Summary

Thank you!