EASYDEEPLEARN

Supervised Learning

Interview cheat sheet

215 questions · Classification, regression, evaluation and the classic bias-variance interview questions.

#classification #metrics #features #theory #regularization #imbalance #linear-regression #feature-selection #hyperparameter-tuning #metrics-regression

Fundamentals & theory

What is the bias-variance tradeoff?

Bias is error from wrong assumptions (underfitting). Variance is error from sensitivity to training data (overfitting).

How do you detect and fix overfitting?

Detect it when training loss keeps dropping but validation loss rises, or when train accuracy is much higher than validation. Fixes: more data or augmentation, regularization (L1/L2, dropout), simpler model, early stopping, cross-validation, and ensembling.

What is k-fold cross-validation and when do you use stratified or grouped folds?

Split the data into k folds; train on k-1 and validate on the held-out fold, then average. Use stratified k-fold for classification (preserves class ratios per fold).

Generative vs discriminative classifier — what's the difference?

A discriminative model learns $P(y \mid x)$ directly (logistic regression, SVM, neural nets). A generative model learns P(x, y) or $P(x \mid y)$ and applies Bayes' rule to get $P(y \mid x)$ (Naive Bayes, GDA, GANs, diffusion for x).

What is the curse of dimensionality?

As dimensions grow, data becomes sparse: distances between points concentrate, the volume needed to cover the space grows exponentially, and models that rely on locality (k-NN, kernel methods) degrade. You need exponentially more data to maintain density.

Parametric vs non-parametric models — what's the difference?

Parametric models have a fixed number of parameters that doesn't grow with the dataset (linear regression, logistic regression, GLMs, Naive Bayes). They make strong assumptions about the functional form and are fast + data-efficient when those assumptions hold.

Why do we prefer simpler models when performance is equal?

Occam's razor: among models with similar validation performance, the simpler one usually generalizes better and is cheaper to serve, easier to debug and monitor, and less prone to overfit noise. Complexity should be justified by clear gains.

What does the no free lunch theorem say for ML?

Averaged over all possible problems, no learning algorithm is better than random guessing. Any advantage of an algorithm on one class of problems is offset by worse performance elsewhere.

In one sentence, what is PAC learning?

Probably Approximately Correct learning formalizes when a hypothesis class can be learned from a finite sample: for any small epsilon and delta, with enough samples the algorithm returns a hypothesis with error at most epsilon with probability at least 1-delta. The required sample size scales with the complexity of the hypothesis class (VC dimension) and 1/epsilon.

What is VC dimension and why should you care?

The Vapnik-Chervonenkis dimension of a hypothesis class is the largest number of points it can shatter (label with any possible +/- assignment). Higher VC dimension = more expressive class = higher risk of overfitting on small samples.

What is Empirical Risk Minimization?

ERM is the standard learning principle: pick the hypothesis in your class that minimizes the average loss on the training sample, as a proxy for the true (unknown) expected loss. It works when the class is not too flexible relative to the sample size — otherwise you overfit.

Why do we care whether the loss landscape is convex?

A convex loss (linear regression MSE, logistic regression log loss, SVM hinge, LASSO) has a single global minimum, so any local optimizer finds the best solution and results are reproducible. Non-convex losses (neural networks, matrix factorization, deep boosting) have many local minima and saddle points — training is sensitive to initialization, learning rate, and stochasticity.

What are the roles of the training, validation and test sets?

Training set: fit the model parameters. Validation set: tune hyperparameters, choose between models, early stop.

What does the IID assumption mean and when is it violated?

IID = samples are independent and identically distributed, drawn from the same population. Standard ML theory (CV, generalization bounds, resampling) assumes IID.

Why is the base rate of the positive class critical to know?

The base rate is the prior probability of the positive class in the population you'll deploy on. It sets the floor for any metric — 99% accuracy is trivial on a 1%-positive problem by always predicting negative.

What is inductive bias and why does every model have one?

Inductive bias is the set of assumptions a learning algorithm uses to generalize from finite data to unseen examples. Without an inductive bias, any function consistent with the training data is equally likely — no learning is possible.

What is the generalization gap and how do you shrink it?

Generalization gap = expected test loss minus training loss. A large gap means overfitting.

Why does OLS have a closed-form solution but logistic regression doesn't?

OLS minimizes ||y - Xw||^2 which is quadratic in w — setting the gradient to zero yields the normal equations $w = (X^T X) - 1 X^T$ y, a closed form. Logistic regression minimizes the log-loss which is convex but not quadratic (log of a sigmoid); its gradient has no closed-form root, so we solve it iteratively (IRLS, Newton, or SGD).

What's the Bayesian interpretation of Ridge regression?

Ridge regression is the maximum a posteriori (MAP) estimate under a Gaussian prior on the weights: w ~ $N(0,\ \sigma^2/\lambda)$. Equivalently, adding an L2 penalty on w is equivalent to assuming the coefficients are a priori small.

What is a Generalized Linear Model and when is it the right tool?

A GLM has three ingredients: (1) a linear predictor $\eta = \mathbf{w}^\top \mathbf{x}$; (2) a link function g relating η to $\mathbb{E}[y \mid x]$, so $g(\mu) = \eta$; (3) a distribution from the exponential family for y (Gaussian → linear regression, Bernoulli → logistic, Poisson → Poisson regression, Gamma → Gamma regression). Use a GLM when your target has a known non-Gaussian distribution but the mean-predictor relationship is approximately linear on the right scale.

Why do we use cross-entropy (log loss) instead of MSE for classification?

Cross-entropy matches the maximum-likelihood estimator for a Bernoulli / categorical target and has a well-shaped gradient when combined with sigmoid/softmax outputs — large errors give large gradients so the model learns fast. MSE on top of sigmoid gives a very small gradient when predictions are confidently wrong (the sigmoid saturates), so training is slow and can get stuck.

How is logistic regression fit in practice?

Two main approaches: (1) Iteratively Reweighted Least Squares (IRLS) — an implementation of Newton-Raphson on the log-likelihood. Each iteration solves a weighted least squares problem with weights = p(1-p).

Your scikit-learn logistic regression warns 'lbfgs failed to converge'. What do you do?

Common fixes: (1) standardize features — most convergence issues come from ill-scaled features; (2) increase $\max_{\text{iter}}$ (e.g., 1000); (3) increase regularization C down (stronger penalty smooths the loss); (4) switch solver ('saga' or 'liblinear' for L1, 'newton-cg' for very small data); (5) check for perfect / near-perfect separation and add L2 or drop the offending feature; (6) verify the target is not constant.

How does Bayes' theorem drive Naive Bayes classification?

For class c and features x, $P(c \mid x) \propto P(x \mid c) \cdot P(c)$. Naive Bayes assumes features are conditionally independent given the class: $P(x \mid c) = \text{prod}_j\ P(x_j \mid c)$.

The 'naive' independence assumption is almost always false. Why does Naive Bayes still work?

Because classification only requires that the argmax over classes be correct — not that the posterior probabilities themselves be accurate. Even when the independence assumption is violated, the decision boundary from NB can be close to optimal on many problems.

Why do we compute Naive Bayes scores in log space?

The product $\text{prod}_j\ P(x_j \mid c)$ can be astronomically small (underflow) when there are many features (e.g., thousands of words in a document). Taking logs turns the product into a sum: log $P(c) + \text{sum}_j \log P(x_j \mid c)$.

What are the core assumptions behind Linear Discriminant Analysis (LDA)?

LDA assumes each class-conditional distribution is Gaussian with class-specific mean but a *shared* covariance matrix Σ. Under this assumption, the log-posterior is linear in x — the decision boundary between any two classes is a hyperplane.

Why does KNN degrade badly in high dimensions?

In high dimensions, pairwise distances between random points concentrate — the ratio (max - min) / min distance approaches 0. Every point becomes roughly the same distance from every other, so the concept of 'nearest' loses meaning and KNN votes become dominated by noise.

How do KD-trees and ball-trees speed up KNN, and when do they stop helping?

Both are spatial index structures that let you skip large portions of the training set during queries. KD-trees split axis-aligned; efficient for low-dimensional data (~≤ 20 dims) with continuous features.

Why not just fit a linear regression for a binary label (linear probability model)?

You can — sometimes it even works reasonably as a quick baseline — but three problems bite in practice: (1) predicted probabilities aren't bounded to [0, 1]; (2) errors are heteroscedastic (variance = p(1-p) depends on x), so OLS standard errors are wrong; (3) it isn't the MLE for Bernoulli data. Logistic regression fixes all three via the sigmoid link and the Bernoulli likelihood — that's why it's the default.

What's the difference between probability, odds, and log-odds?

Probability p in [0, 1]. Odds = p / (1 - p) in [0, ∞): 'how many times more likely than not'.

Why is a single decision tree considered a 'high-variance' model?

A small change in the training data can produce a completely different tree: the first split cascades, so if a different feature is chosen at the top, everything below changes. This makes individual trees unstable — predictions vary a lot with resampling.

Why do decision trees (and boosted trees) fail to extrapolate?

A tree predicts by returning the leaf value; leaves are bounded by the observed training range. If a test feature is beyond the training range, the tree returns whatever leaf value corresponds to the closest region — no extrapolation.

What is the out-of-bag (OOB) score in Random Forests?

Each bootstrap sample leaves out ~37% of the training data (untouched by that tree). For each training point x, aggregate predictions from the trees where x was NOT in-bag; compare to y.

What is a support vector, and why does the SVM only depend on them?

Support vectors are the training points that lie on the margin or inside it (or misclassified for soft margins) — points with non-zero Lagrange multipliers in the dual problem. The optimal hyperplane is a linear combination of these support vectors alone: adding or removing non-SV points doesn't change the solution.

Why don't kernel SVMs scale well to millions of samples?

The dual problem has $O(n^2)$ memory (kernel matrix) and $O(n^2)$ to $O(n^3)$ training complexity. For n = 1M, that's a trillion entries — impossible.

What is 'deviance' and why is it used to evaluate GLMs?

Deviance = -2 * (log-likelihood of your model - log-likelihood of the saturated model). Analog of squared error for non-Gaussian likelihoods: Poisson deviance for count data, Tweedie deviance for insurance-style claims, binomial deviance for logistic regression.

You must pick ONE metric for your model. How do you decide?

Start from the business objective, not the model: what does 'error' cost, and to whom? Classification with asymmetric costs → expected cost or F-beta with the appropriate beta.

An interviewer asks you to build a churn model. What do you do before touching any algorithm?

Define the prediction problem precisely: what counts as churn, and over what horizon. Fix the prediction time, so every feature is something you would actually have at that moment.

Linear & regularized regression

L1 vs L2 regularization — what's the difference?

L2 (ridge) adds $\mathrm{sum}(w^2)$ to the loss and shrinks weights smoothly toward zero. L1 (lasso) adds $\mathrm{sum}(|\ w\ |)$ and pushes weights exactly to zero, giving sparse solutions and doing feature selection.

Why is logistic regression called a linear model if it uses a sigmoid?

The decision boundary is linear in the input features because the model is linear in the log-odds: $\mathrm{logit}(p)\ =\ w^T$ x + b. The sigmoid only maps that linear score into [0, 1] for a probability.

What are the classical assumptions of linear regression?

(1) Linearity: the conditional mean $E[y\mid x]$ is linear in the parameters. (2) Independence of errors.

How do you interpret a coefficient in a multiple linear regression?

The coefficient β_j is the expected change in y per one-unit increase in x_j while holding all other features fixed — the partial or ceteris-paribus effect. Sign gives the direction; magnitude depends on the scale of x_j.

What is VIF and when do you worry about multicollinearity?

Variance Inflation Factor of feature $x_j\ =\ 1\ /\ (1\ -\ R_j^2)$, where R_j^2 is the R^2 of regressing x_j on the other features. VIF = 1 means no collinearity; VIF > 5 or 10 is a common warning threshold.

How do you handle strong multicollinearity in a linear model?

(1) Drop one of the correlated features (highest VIF first). (2) Combine them via PCA or domain-driven aggregation.

How do you detect and fix heteroscedasticity?

Heteroscedasticity = error variance depends on x. Detect with a residuals-vs-fitted plot (funnel shape), Breusch-Pagan or White test.

Adding a feature raised R^2. Does that mean the model improved?

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

is the fraction of variance explained; it never decreases when you add a feature, even a useless one. Adjusted R^2 penalizes for the number of features and can decrease when you add a feature that doesn't help.

Can R^2 be negative? What does that mean?

Yes.

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

compares your model's residual sum of squares to that of a constant-mean baseline. A negative R^2 on a test set means your model is worse than predicting the mean everywhere — usually a sign of severe overfitting or distribution shift.

When would you use robust regression (Huber, RANSAC) instead of OLS?

Use robust regression when the data contains outliers that shouldn't dominate the fit. OLS minimizes squared error, so a single large-residual point can pull the line dramatically.

What's the main risk of polynomial regression?

High-degree polynomial features overfit dramatically: they can hit training points exactly while oscillating wildly between them (Runge's phenomenon). Predictions extrapolate very poorly outside the training range.

When should you add interaction features to a linear model?

Add $x_i\ \cdot\ x_j$ when the effect of one feature on y depends on the value of another and your model class can't discover this on its own (linear models can't). Common in marketing (channel × segment), medicine (treatment × age), and pricing (product × geography).

When does log-transforming the target help a regression model?

Log-transform y when (1) y is strictly positive and right-skewed (revenue, counts, biological measurements, house prices), (2) residuals show heteroscedasticity that shrinks after the transform, (3) you care about relative rather than absolute errors — modeling $\log(y)$ is equivalent to modeling percentage errors. Watch out: predictions in log space need $\exp()$ to be brought back, and the expectation is biased ($E[\exp(z)]\ \neq\ \exp(E[z])$).

What do Box-Cox and Yeo-Johnson transforms do?

Both are power transforms that find a lambda making the data approximately Gaussian and stabilizing variance. Box-Cox requires strictly positive data.

Why does Ridge regression give more stable coefficients than OLS?

Ridge solves $(X^T\ X\ +\ \lambda\ I)\ w\ =\ X^T$ y instead of X^T X $w\ =\ X^T$ y. Adding lambda I to the diagonal keeps the matrix invertible even when features are collinear or when p > n, and shrinks coefficients toward zero proportionally to their scale.

Why does L1 (Lasso) produce sparse coefficients but L2 (Ridge) does not?

Geometrically, the L1 constraint region is a diamond with corners on the axes, so the loss contour tends to intersect the constraint at a corner where some coefficients are exactly zero. The L2 region is a sphere, so intersections are typically off-axis — coefficients become small but not zero.

When is Elastic Net better than pure Lasso or Ridge?

Elastic Net combines L1 and L2 penalties: $\alpha \cdot (\rho \cdot |w|1 + (1-\rho) \cdot 0.5 \cdot |w|2^2)$. Use it when (1) features are highly correlated — Lasso arbitrarily picks one and drops the rest, Elastic Net groups correlated features together via the L2 term; (2) you have more features than samples (p >> n) — Lasso caps the number of selected features at n.

How do you choose the regularization strength (lambda / alpha)?

Cross-validation over a log-spaced grid is the standard approach — pick alpha minimizing average validation loss. Coefficient path plots (coefficient vs log alpha) help understand which features enter/exit at what strength.

Why must you standardize features before applying L1 or L2 regularization?

The penalty is applied uniformly to all coefficients, but the natural scale of a coefficient depends on the scale of its feature (weight * feature = signal). Without standardization, a feature measured in millimetres would get a huge coefficient and be over-penalized versus one in metres.

In one sentence, what does the LARS algorithm compute?

Least Angle Regression is a stagewise algorithm that computes the entire regularization path of Lasso coefficients (from full L1 penalty down to zero penalty) in about the same cost as one OLS fit, by adding features to the active set in the order of highest correlation with the current residual and moving in a direction equiangular to them.

When would you use Group Lasso instead of standard Lasso?

Group Lasso applies an L2 penalty within groups of coefficients and an L1 penalty across groups: entire groups get selected or dropped together. Use it when features come in natural groups — one-hot dummies of a categorical variable, spline basis functions, features from the same sensor.

When is quantile regression more useful than mean (OLS) regression?

Quantile regression models a specific conditional quantile of y (e.g., the median, 90th percentile) instead of the mean. Use it when (1) you care about tails, not the average — delivery time SLA, inventory sizing, risk quantiles; (2) the target distribution is asymmetric or heavy-tailed; (3) you want a prediction interval by fitting several quantiles.

What is 'perfect separation' in logistic regression and how do you fix it?

Perfect separation happens when a feature (or combination) fully separates the classes: the MLE for the corresponding weight goes to +/- infinity and the solver never converges (or gives massive coefficients with huge standard errors). Fixes: (1) add L2 regularization (drives the weight to a finite value); (2) use Firth's penalized likelihood; (3) drop or bin the separating feature.

L1 vs L2 in logistic regression — practical differences.

Same intuition as in linear regression: L2 (default) shrinks all coefficients smoothly and stabilizes training under multicollinearity; L1 produces sparse coefficients and performs feature selection during training. In scikit-learn, use penalty='l1' with the 'liblinear' or 'saga' solver.

What is Laplace (additive) smoothing in Naive Bayes?

It adds a small constant alpha to every count when estimating $P(x_j \mid c)$ so no probability is ever exactly zero. Without smoothing, a single word never seen with class c in training would give $P(\text{document} \mid c) = 0$, killing the posterior.

Pre-pruning vs post-pruning in decision trees — what's the difference and when do you use each?

Pre-pruning stops the tree from growing during construction, via $\max_{\text{depth}}$, $\min_{\text{samples}}$ split, $\min_{\text{samples}}$ leaf, or $\min_{\text{impurity}}$ decrease. Fast and simple but you may stop too early (horizon effect).

How does cost-complexity (CCP) pruning work?

Grow the tree fully. Define $R_\alpha(T) = R(T) + \alpha$ * |T|, where R(T) is the training error, $|T|$ is the number of leaves, and alpha ≥ 0 penalizes complexity.

How does early stopping work in gradient boosting and why is it important?

Monitor validation loss after each tree is added; stop when it hasn't improved for $\text{early}_{\text{stopping}}\text{rounds}$ boosting iterations. Returns the model at the best iteration (not the last).

How does the C parameter in an SVM affect the fit?

C weights the misclassification penalty in the soft-margin loss (opposite of a typical regularization strength — larger C means less regularization). Large C: try hard to classify every training point correctly → narrow margin, complex boundary, high variance.

How does SVM adapt to regression (SVR)?

SVR uses an epsilon-insensitive loss: residuals $\text{with } |y - y_{\text{hat}}| \leq$ epsilon are free (zero penalty), residuals outside are penalized linearly (like L1). Combined with the kernel trick, it fits a flexible non-linear function using only a subset of training points as support vectors.

What is Huber loss and why is it a compromise between MAE and MSE?

Huber loss is quadratic $\text{for } |r| \leq$ delta (like MSE) and linear beyond delta (like MAE), joined smoothly at delta. Combines MSE's differentiability near zero (good gradients for optimization) with MAE's robustness to outliers (linear tail doesn't blow up on rare huge errors). delta acts as an outlier threshold — typical values are $1 \cdot \sigma_y$ or set via CV.

What does the quantile (pinball) loss measure?

For quantile tau in (0, 1), pinball loss(r) = tau * r if r > 0 else (tau - 1) * r, where $r = y - y_{\text{hat}}$. Minimizing it fits the conditional tau-quantile of y given x. $\tau = 0.5$ → median (equivalent to MAE). $\tau = 0.9$ → 90th percentile prediction — great for delivery-time SLAs, inventory sizing, risk quantiles.

What is stability selection?

Run your feature-selection method (usually Lasso) on many random subsamples of the data. Record how often each feature is selected.

What are monotonic constraints in gradient boosting and when are they worth using?

They force the predicted output to move in one direction as a feature increases. Useful when domain knowledge is certain: risk should not fall as debt rises, and price should not fall as square footage rises.

When would you use quantile regression instead of predicting the mean?

When the decision needs a range rather than a point. Inventory planning cares about the 90th percentile of demand, not the average, because stocking to the mean stocks out half the time.

Logistic regression & classification basics

Precision vs recall — when do you optimize each?

Precision = TP / (TP + FP): of the items flagged positive, how many really are. Recall = TP / (TP + FN): of the actual positives, how many we caught.

How do you read a confusion matrix and derive the key metrics?

Rows = actual class, columns = predicted class (or vice versa). Cells: TP, FP, TN, FN.

One-vs-Rest, One-vs-One, and softmax — how do you choose for multi-class?

One-vs-Rest trains K binary classifiers (class k vs the rest); simple and calibrates per-class, but scores aren't jointly normalized. One-vs-One trains K(K-1)/2 binary models between every pair; more models but each sees a balanced binary problem, useful for SVMs. Softmax / multinomial logistic regression jointly models all classes with a single objective; the natural choice for logistic regression and neural nets and it produces coherent probabilities that sum to one.

Write out the softmax function and its main property.

$\mathrm{softmax}(z)k = \exp(z_k) \;/\; \mathrm{sum}_j \exp(z_j)$. It maps a vector of real logits to a probability distribution: values in (0, 1) that sum to 1.

How do you interpret a logistic regression coefficient as an odds ratio?

In $\log(\mathrm{odds}) = w^T$ x, a one-unit increase in x_j multiplies the odds by $\exp(w_j)$. $\exp(w_j)$ is the odds ratio: 1.5 means a 50% relative increase in odds per unit of x_j. Positive $w \rightarrow \mathrm{odds\ ratio} > 1 \rightarrow \mathrm{outcome}$ more likely; negative w -> < $1 \rightarrow$ less likely.

How do class weights work in logistic regression, and when do you use them?

Class weights re-weight the log-loss contributions of each class so the minority class carries more penalty when misclassified. 'balanced' in scikit-learn sets $\mathrm{weight}_c = n_{\mathrm{samples}} \;/\; (n_{\mathrm{classes}} \cdot n_c)$.

'multinomial' vs 'ovr' setting in scikit-learn's LogisticRegression — what changes?

'multinomial' fits a joint softmax model over all K classes with a single log-loss objective — the natural probabilistic model for multi-class. 'ovr' trains K independent binary logistic regressions (class k vs the rest) and normalizes per-example at prediction time.

Gaussian NB vs Multinomial NB vs Bernoulli NB — when do you use each?

Gaussian NB: continuous features assumed to be normally distributed per class (baseline for tabular numeric data). Multinomial NB: counts / term frequencies — the go-to for bag-of-words text classification.

When would LDA outperform logistic regression?

When (1) classes are approximately Gaussian with similar covariances — LDA is the maximum-likelihood classifier for that model, more efficient than logistic regression which is agnostic to the feature distribution; (2) the training set is small — LDA has fewer parameters and estimates them from all of the data at once; (3) classes are well-separated — logistic regression can suffer from perfect separation, LDA doesn't. Logistic regression usually wins when features are non-Gaussian or highly correlated.

What is the nearest centroid (Rocchio) classifier and when is it useful?

Compute the mean feature vector per class; classify a new point as the class whose centroid is closest (usually Euclidean or cosine). Extremely simple, no hyperparameters, very fast, no training beyond averaging.

How does temperature scaling calibrate a classifier's probabilities?

You take the trained logits z, divide by a scalar T > 0, then softmax: p = softmax(z / T). T is fit on a held-out validation set by minimizing NLL.

Why is accuracy a bad primary metric for many real-world problems?

Accuracy weights every error equally and ignores class balance. On a 99% negative dataset, always predicting negative gives 99% accuracy — a model that's completely useless.

What is F-beta score and when is F1 not enough?

$F_\beta = (1 + \beta^2) \cdot P \cdot R \;/\; (\beta^2 \cdot P + R)$ generalizes F1 with a weight on recall vs precision. $\beta = 1 \rightarrow$ F1 (equal). $\beta = 2$ (F2) → recall matters 4x more than precision — appropriate for medical screening where missing positives is very costly. $\beta = 0.5$ (F0.5) → precision matters 4x more — for a spam filter where false positives annoy users. F1's implicit assumption is that precision and recall are equally important — often they aren't.

Macro vs micro vs weighted averaging in multi-class metrics — how do you choose?

Macro: compute metric per class, then average with equal weight. Treats each class equally — useful when small classes matter (e.g., rare diseases).

What is Cohen's kappa and why is it useful?

Cohen's $\kappa = (p_o - p_e) \;/\; (1 - p_e)$, where p_o is observed agreement (accuracy) and p_e is the agreement expected by chance given class marginals. It corrects accuracy for the baseline of random guessing given the class prior.

What is the Matthews Correlation Coefficient (MCC) and when should you use it?

MCC = (TP*TN - FP*FN) / sqrt((TP+FP)(TP+FN)(TN+FP)(TN+FN)) — the Pearson correlation coefficient between predicted and true binary labels. Range [-1, 1]: 1 perfect, 0 chance, -1 perfectly wrong.

What is log loss (cross-entropy) and what does a specific value mean?

For binary: log loss = -1/n * sum [y log(p) + (1-y) log(1-p)]. It's the negative log-likelihood of the observed labels under the predicted probabilities — proper scoring rule that rewards well-calibrated confident predictions and heavily punishes confident mistakes. $\log(0.5) \approx 0.693$ is the loss of always predicting 0.5 (max-uncertainty baseline).

What is the Brier score and how does it compare to log loss?

Brier = 1/n * sum $(p_i - y_i)2$ — mean squared error between predicted probability and true label. Range [0, 1] for binary.

What is top-k accuracy and when do you use it?

Top-k counts a prediction as correct if the true class is among the top-k predicted classes ranked by score. Common in image classification with many classes (ImageNet top-5), recommender systems (was the right item in the top-10 recommendations?), and retrieval-style tasks.

What is balanced accuracy?

Balanced accuracy = (sensitivity + specificity) / 2 for binary, or the macro average of per-class recall for multi-class. It corrects accuracy by treating every class equally regardless of size — a 50% baseline for random guessing on any class balance.

What is the geometric mean (G-mean) in classification metrics?

G-mean = sqrt(sensitivity * specificity). Only high when both true positive and true negative rates are high — a single point going to zero drags the metric to zero.

What is Youden's J statistic and how is it used?

J = sensitivity + specificity - 1. It's the vertical distance from the ROC curve to the diagonal at a given operating point.

How do you interpret an ROC curve, and what point matters?

ROC plots True Positive Rate (sensitivity) vs False Positive Rate (1 - specificity) as the decision threshold sweeps from 1 to 0. The diagonal is random guessing; the top-left corner is perfect.

What does ROC-AUC really measure, and what's a good value?

AUC is the probability that a random positive example is scored higher than a random negative example (equivalent to the Mann-Whitney U statistic normalized). 0.5 = random; 1.0 = perfect ranking. Rules of thumb: 0.7 acceptable, 0.8 good, 0.9+ excellent.

What are the main pitfalls of relying on ROC-AUC?

(1) On heavily imbalanced data, ROC-AUC is optimistic because the false positive rate is dominated by a huge number of negatives — PR-AUC is more informative. (2) AUC is threshold-independent — a model with great AUC can still have terrible precision at any usable threshold.

How do you select the decision threshold for a binary classifier?

Compute predicted probabilities on validation, then choose the threshold based on your business objective: (a) maximize F1 → argmax over t of F1(y, p > t); (b) fixed recall → smallest t with recall ≥ target, maximizing precision; (c) expected cost → t that minimizes $\mathrm{FP}_{\text{cost}} \cdot \mathrm{FP} + \mathrm{FN}_{\text{cost}} \cdot \mathrm{FN}$. Never tune threshold on the test set.

How do you evaluate a multi-label classifier?

Common metrics: (a) Hamming loss = fraction of misclassified labels (per-label error) — lenient; (b) exact match / subset accuracy = fraction of examples where ALL labels are correctly predicted — strict; (c) per-label F1 with macro/micro averaging — the workhorse; (d) label ranking metrics (mean average precision, coverage error) when scores matter more than binary decisions. Always report per-label metrics too, especially when label frequencies differ.

MAP, NDCG, MRR — when do you use each in ranking?

MRR (Mean Reciprocal Rank): 1/rank of the first relevant item, averaged over queries. Ideal for 'find one right answer' tasks.

How do you evaluate a classifier when FP and FN have different costs?

Assign explicit costs C_{FP} and C_{FN}. Compute expected cost per $\text{example} = C_{\mathrm{FP}} \cdot \mathrm{FPR} \cdot P(\mathrm{neg}) + C_{\mathrm{FN}} \cdot \mathrm{FNR} \cdot P(\mathrm{pos})$.

Naive Bayes, LDA & KNN

Why is Naive Bayes still a solid baseline for text classification?

For bag-of-words / TF-IDF features it is (1) very fast to train and predict — a few matrix operations; (2) memory-efficient — you only store per-class word probabilities; (3) works well even with tiny labeled datasets since parameters are estimated per feature; (4) linear in the number of features; (5) resilient to irrelevant features because they contribute little to the score. It's the default first baseline before trying logistic regression or transformers.

QDA vs LDA — how do you choose between them?

QDA relaxes LDA's shared-covariance assumption: each class has its own Σ_c, so the decision boundary is quadratic. Prefer QDA when class covariances clearly differ (visible from ellipsoid plots or per-class covariance estimates) and you have enough data per class to estimate them reliably.

How is LDA used for supervised dimensionality reduction?

Beyond classification, LDA finds up to K-1 linear directions (K = number of classes) that maximize between-class variance while minimizing within-class variance. Project the data onto these axes and you get a low-dimensional embedding that keeps class separation.

How do you choose k in k-Nearest Neighbours?

Cross-validation over odd values (avoids ties in binary problems). Small k (1-3) = low bias, high variance — sensitive to noise.

How do you choose the distance metric for KNN?

Euclidean is the default for continuous features that have been standardized. Manhattan (L1) is more robust to outliers and often used with sparse or high-dimensional data.

What's the point of distance-weighted KNN?

Instead of every neighbour voting equally, weight closer neighbours more (e.g., weight = 1/distance or a Gaussian kernel). This makes KNN less sensitive to k, gives a smoother decision boundary near class overlaps, and gives the closest point the strongest vote — useful when the local geometry matters.

You trained Naive Bayes on balanced data but deploy on data where positives are 1%. What happens and what do you do?

The prior P(c) in Bayes' rule shifts, so probabilities and thresholds are off. Fixes: (1) refit priors from the deployment distribution while keeping the likelihoods ($P(x \mid c)$) from training — this is exactly what you need since P(c) is easy to estimate from deployment counts; (2) equivalently, adjust logit predictions by $\log(P_{\text{deploy}}(c) \;/\; P_{\text{train}}(c))$ per class; (3) recalibrate on a small deployment-labelled sample if you can get one.

How does KNN imputation work and when is it a good choice?

For each row with missing values, find the k nearest complete rows using the observed features, then fill the missing values with a weighted average (regression) or mode (categorical) of those neighbours. Handles multivariate structure automatically — better than mean/median for correlated features.

Decision trees & random forests

Random forest vs gradient boosting — which do you pick and why?

Random forest averages many deep trees trained on bootstrapped samples with random feature subsets — it reduces variance and is robust with little tuning. Gradient boosting (XGBoost, LightGBM, CatBoost) fits trees sequentially, each correcting residuals — it usually gives better accuracy on tabular data but needs more tuning and is more prone to overfitting.

Gini impurity, entropy, and classification error — which do trees actually use?

For classification, trees choose the split that most reduces impurity. Gini = 1 - sum p_c^2 and entropy = -sum p_c log p_c are almost interchangeable in practice, both being smooth and differentiable proxies for classification error.

How do decision trees handle categorical features?

Native support depends on the library. Classic CART considers only binary splits: partition category set A vs the rest, or the exhaustive best partition
$(2(k-1)-1 \text{ possibilities, expensive for high cardinality})$. scikit-learn does *not* support categoricals natively — you must one-hot encode.

What are the most impactful hyperparameters of a decision tree?

$\max_{\text{depth}}$: hard limit on depth — the biggest lever. $\min_{\text{samples}}$ split: minimum samples needed to consider splitting a node. $\min_{\text{samples}}$ leaf: minimum in each leaf — controls leaf size and variance. $\max_{\text{features}}$: number of features to consider per split (=all for a tree, sqrt(d) or similar for a forest). $\min_{\text{impurity}}$ decrease: don't split unless impurity drops by at least this much. Together they control the bias-variance tradeoff and inference latency.

How do decision trees handle missing values?

Options: (1) surrogate splits (CART's approach): find a backup feature that best mimics the chosen split's partition — used when the primary feature is missing at prediction time. (2) Default direction (XGBoost, LightGBM): during training, learn per split which direction — left or right — missing values should go.

What criterion does a regression tree use to choose splits?

Variance reduction: pick the split that most reduces sum of squared errors of the target within child nodes. Equivalently, maximize the total variance of the parent minus the weighted sum of children variances.

How is feature importance computed from a decision tree or random forest?

Impurity-based (MDI): sum, over all splits using that feature, of the impurity decrease weighted by the number of samples reaching the split. Cheap and comes free from training.

How does bagging reduce variance?

Bagging (bootstrap aggregating) trains B models on B bootstrap samples of the training set and averages predictions. If the models were independent with variance σ^2, the average has variance $\sigma^2 \ / \ B$.

Why does Random Forest sample features at each split, not just once per tree?

Per-split feature subsampling forces different trees to explore different features and different splits. Without it, if one feature is very predictive, every tree would keep splitting on it near the root and become highly correlated — averaging correlated trees barely reduces variance.

How do Extra Trees differ from Random Forests?

Two changes: (1) each tree is trained on the whole training set (no bootstrapping); (2) splits are chosen randomly — for each candidate feature, a random split threshold is drawn rather than the optimal one, and the best of those random splits is picked. Result: more randomness, more bias, less variance, and *faster* training.

Which Random Forest hyperparameters actually matter for tuning?

$n_{\text{estimators}}$: 'more is better' until returns diminish — 200-500 is a solid default. $\max_{\text{features}}$: the main lever for correlation between trees; try sqrt(d), log2(d), or 0.3-0.5*d. $\max_{\text{depth}}$ / $\min_{\text{samples}}$ leaf: control tree size; deeper trees favor bias-reduction, larger $\min_{\text{samples}}$ leaf reduces variance. $\text{class}_{\text{weight}}$='balanced' for imbalanced data. n_{jobs}=-1 for speed. Skip fiddling with $\min_{\text{samples}}$ split — $\min_{\text{samples}}$ leaf is enough.

Are Random Forest probability estimates well-calibrated?

Not always. RF probabilities come from averaging tree votes and tend to be pushed away from 0 and 1 (over-cautious).

What is Boruta and when do you reach for it?

A wrapper feature selection algorithm that compares each feature's importance to that of 'shadow' features — random permutations of that feature. A feature is confirmed if its importance is significantly higher than the max shadow importance across multiple random forest fits (statistical test with Bonferroni correction).

Gradient boosting

In one sentence, what is Gradient Boosting?

Fit a sequence of weak learners (typically shallow trees), where each new learner is trained to predict the negative gradient of the loss w.r.t. the current ensemble's prediction — additive stagewise optimization in function space. Final prediction is the sum of the learners' outputs (each scaled by a learning rate).

What is the role of the learning rate (shrinkage) in GBM?

Each new tree's contribution is scaled by eta in (0, 1] before being added: $F_{m+1} = F_m + \eta \cdot h_m$. Smaller eta needs more trees ($n_{\text{estimators}}$) but generalizes better — analogous to a slow, cautious descent that lets many complementary trees contribute rather than one big correction.

What did XGBoost bring on top of vanilla GBM?

(1) Regularization terms on tree structure: gamma * T (penalty per leaf) and lambda * sum w_j^2 (L2 on leaf scores), reducing overfitting. (2) Second-order Taylor expansion of the loss for split gain (Newton-like), giving better splits.

What are LightGBM's key innovations vs XGBoost?

(1) Leaf-wise (best-first) tree growth instead of level-wise — grows the leaf with the largest loss reduction, giving deeper asymmetric trees and often lower loss for the same tree count. (2) GOSS (Gradient-based One-Side Sampling): keep all large-gradient samples, subsample small-gradient ones — trains faster with little accuracy loss.

What makes CatBoost different from XGBoost / LightGBM?

(1) Ordered boosting: uses a permutation of the data so that for each example, the model used to estimate its gradient is fit on other examples only — reduces prediction shift / target leakage that plagues target encoding in vanilla GBM. (2) Native categorical support via ordered target statistics — handles high-cardinality categoricals without one-hot.

Which XGBoost hyperparameters have the biggest impact, and in what tuning order?

First: fix $n_{\text{estimators}}$ large $(eg,\ 2000)\ +\ \text{early}_{\text{stopping}}\text{rounds}$ so the count auto-selects. Then tune (roughly in this order): (1) $\text{learning}_{\text{rate}}$ (0.01-0.1); (2) $\max_{\text{depth}}$ (3-10) or $\text{num}_{\text{leaves}}$; (3) $\min_{\text{child}}\,\text{weight}$ (regularization on leaves); (4) subsample and $\text{colsample}_{\text{bytree}}$ (0.5-1); (5) $\gamma\ /\ \min_{\text{split}}\,\text{loss}$ (0-5); (6) reg_{α} (L1) and reg_{λ} (L2).

How do you handle class imbalance in XGBoost / LightGBM?

Set $\text{scale}_{\text{pos}}\text{weight}\ =\ \text{negatives}\ /\ \text{positives}$ (for binary) so gradients from the minority class are up-weighted. Alternatively use $\text{is}_{\text{unbalance}}\ =\ \text{true}$ in LightGBM.

How do XGBoost, LightGBM, and CatBoost each handle categorical features?

XGBoost: needs one-hot or ordinal encoding for older versions; native categorical support from 1.5+ using ordered partitioning. LightGBM: native support via Fisher-style optimal categorical partitioning; expects category indices as int type or the special 'category' dtype.

How does gradient boosting handle missing values without imputation?

For each split, the algorithm considers routing missing values to the left or the right child and picks the direction that yields the largest gain on the training data. This 'learned default direction' is stored per split and used at prediction time.

What are monotonic constraints in XGBoost / LightGBM and when do you use them?

You can force the prediction to be non-decreasing (or non-increasing) in a specified feature. Useful for regulatory / business constraints: 'credit score shouldn't decrease when income goes up', 'insurance premium shouldn't drop when age increases'.

Rule of thumb: when do you pick XGBoost vs LightGBM vs CatBoost?

Roughly: LightGBM is fastest and often top on large, mostly numeric datasets with many features; leaf-wise growth needs careful $\max_{\text{depth}}\ /\ \text{num}_{\text{leaves}}$ control to avoid overfitting. CatBoost is the safest default when you have many high-cardinality categoricals or you want strong out-of-the-box performance with less tuning.

Is adding more features always safe with gradient boosting?

No. Boosting tolerates irrelevant features better than linear models, but there are real costs. Each extra feature is another thing that can break, drift or be unavailable at serving time, so the maintenance burden grows.

SVMs & kernels

Why do SVMs use kernels?

Kernels let SVMs learn nonlinear boundaries without explicitly computing high-dimensional features — the kernel trick evaluates inner products in an implicit feature space. Common kernels: linear (baseline), polynomial, RBF (default nonlinear choice), sigmoid.

Hard-margin vs soft-margin SVM — what's the difference?

Hard-margin SVM finds the hyperplane that separates classes with the largest margin — only works when classes are linearly separable, otherwise there is no feasible solution. Soft-margin adds slack variables $\xi_i \geq 0$ that let some points violate the margin (xi in the margin, xi > 1 misclassified) and penalizes them via C * sum xi.

What does gamma control in an RBF-kernel SVM?

The RBF kernel is $\exp(-\gamma \cdot\ ||\ x\ -\ x\ ||\ 2)$. gamma is $1/(2 \cdot \sigma^2)$ — the inverse of the kernel bandwidth. Small gamma → wide bell, each point influences a large region → smooth boundary, high bias.

How do SVMs handle multi-class problems?

Native SVM is binary. Multi-class strategies: (1) One-vs-Rest (OvR): K binary SVMs, one per class vs the rest.

Why is feature scaling critical for SVMs?

SVMs measure distances between points (via kernels or margins). Features on very different scales dominate the distance computation — a feature in kilometres will completely swamp a feature in millimetres.

Ensembling & stacking

How does stacking work and when does it help?

Level 0: train several diverse base models (e.g., logistic regression, random forest, XGBoost, KNN) using K-fold CV, producing out-of-fold predictions for each. Level 1: train a meta-learner (usually a simple regularized linear model — Ridge / logistic regression) on the level-0 predictions to combine them.

Soft voting vs hard voting — which is usually better and why?

Hard voting: each model outputs a predicted class; the ensemble takes the majority. Simple but throws away probability information.

Why is diversity between base models more important than their individual accuracy in an ensemble?

If two 95%-accurate models make exactly the same mistakes, averaging them still gives 95% — no improvement. If two 90%-accurate models make *different* mistakes, averaging can push accuracy well above either.

Classification metrics

When should you use PR-AUC instead of ROC-AUC?

Use PR-AUC when the positive class is rare (heavy imbalance). ROC-AUC can look optimistic on imbalanced data because the true-negative rate dominates the false-positive rate.

How do you check whether a classifier's probabilities are well-calibrated?

Draw a reliability diagram: bin predictions by predicted probability (e.g., 10 bins), for each bin plot mean predicted probability (x) against fraction of positives (y). A perfectly calibrated model lies on the diagonal.

Platt scaling vs isotonic regression for probability calibration — how do you choose?

Platt: fit a logistic regression on the classifier's scores → sigmoid recalibration. Parametric (2 params), works well with small calibration sets, assumes a sigmoidal miscalibration shape.

How do you choose the decision threshold for a classifier in a business setting?

Not at 0.5. Write down the cost of a false positive and the value of a true positive, then pick the threshold that maximizes expected value.

When do you need calibrated probabilities rather than a good ranking?

You need calibration whenever the probability itself enters a downstream calculation. Expected-value decisions, such as multiplying a predicted default probability by a loan amount, break if the number is not a real probability.

Your positive class is 0.5%. Is resampling your first move?

Usually not. First check whether you have an evaluation problem rather than a training problem: switch from accuracy to precision-recall AUC or precision at the operating point, since accuracy is meaningless at that base rate.

Regression metrics

RMSE vs MAE — how do you pick?

RMSE = sqrt($\mathrm{mean}((y - y_{\mathrm{hat}})2)$) — same units as y, penalizes large errors quadratically. Use when big errors are disproportionately bad (financial forecasting, energy grids) or when errors are approximately Gaussian.

What are the pitfalls of MAPE (Mean Absolute Percentage Error)?

$\mathrm{MAPE} = \mathrm{mean}(|\, y - y_{\mathrm{hat}} \,| \;/\; |\, y \,|) \cdot 100\%$. Issues: (1) undefined when y = 0 and explodes when y is close to 0; (2) asymmetric — over-forecasting is bounded (max 100% error), under-forecasting is unbounded; (3) biases model selection toward under-prediction.

What is sMAPE and why is it used?

Symmetric MAPE = mean($|y - y_{\mathrm{hat}}|$ / (($|\, y \,| + |\, y_{\mathrm{hat}} \,|$)/2)) * 100%. Bounded in [0%, 200%] and symmetric in over-/under-prediction: over-forecasting and under-forecasting by the same absolute amount give equal error.

What is explained variance score and how does it differ from R^2?

$\mathrm{explained}_{\mathrm{variance}} = 1 - \mathrm{Var}(y - y_{\mathrm{hat}}) \;/\; \mathrm{Var}(y)$. It measures the fraction of variance the model captures.

When would you use MSLE (mean squared log error) instead of MSE?

MSLE = mean($(\log(1+y) - \log(1+y_{\mathrm{hat}}))2$). Penalizes relative errors instead of absolute ones — an error of 10 units when y=1000 is treated the same as an error of 1 unit when y=100.

Is R^2 a useful metric for non-linear models (RF, GBM, neural nets)?

It's a valid summary but has caveats: (1) R^2 is not scale-free per dataset — it depends on $\mathrm{Var}(y)$ — so comparing across datasets is misleading; (2) it's easy to inflate by adding features and can overstate goodness for wiggly models on small data; (3) it doesn't reflect calibration or heteroscedastic errors. Prefer RMSE / MAE / quantile loss for absolute quality, MASE for time series comparison, and Bayesian criteria (AIC, BIC) for model selection.

Cross-validation & data splitting

Leave-One-Out CV — when is it a good idea, and when is it a bad idea?

LOOCV uses n-1 samples for training and 1 for validation, repeated n times. Pros: nearly unbiased estimate of generalization error, uses maximum training data per fold.

What is nested cross-validation and when do you need it?

You need it whenever you tune hyperparameters *and* want an unbiased estimate of the tuned model's generalization error. Outer loop: k-fold split — each outer fold's test set is truly held out.

Why would you use repeated k-fold instead of standard k-fold?

A single k-fold estimate has significant variance from the specific random split — especially on small data. Repeated k-fold runs k-fold R times with different seeds, averaging results.

How do you set up cross-validation for time series?

Never random splits — that leaks future info into training. Options: (1) walk-forward / rolling-origin: expand or slide a training window and evaluate on the next block.

How do you pick the train/validation/test split sizes?

Depends on total n and problem noise. Small (n < 10k): 70/15/15 or use CV on train+val and hold out ~15% for final test.

How does cross-validation change for heavily imbalanced classification?

Two things: (1) use stratified k-fold to guarantee each fold contains a representative share of the minority class — otherwise some folds might have zero positives, breaking metrics. (2) evaluate with imbalance-aware metrics (PR-AUC, F1, recall at fixed precision) — accuracy is meaningless.

You use stratified k-fold on a dataset with duplicate customer records — why is it wrong?

Stratified k-fold preserves class proportions but ignores group structure. If the same customer appears in both training and validation, the model can 'memorize' that customer via any customer-specific features (device, location patterns, etc.) — leakage that inflates validation performance.

When would you use the bootstrap for model evaluation instead of k-fold?

Bootstrap resamples the training set with replacement to build many pseudo-datasets. Fit the model on each and evaluate on the out-of-bootstrap (OOB) samples (~37% left out).

When is cross-validation NOT necessary?

When you have enough data that a single held-out validation set is already a low-variance estimate $(\text{typically } n_{\text{val}} \geq 10k \text{ with well} - \text{behaved labels})$. At Google/Facebook scale you almost never do k-fold — a single split is enough.

When does a random train/test split give you a misleading score?

Whenever rows are not independent. With repeated measurements per user, a random split puts the same user on both sides and the model recognizes the user rather than the pattern; use GroupKFold on the user id.

Feature engineering & encoding

Which models need feature scaling and which don't?

Scale features for models that use distances or gradient descent: k-NN, k-means, SVM, PCA, linear/logistic regression with regularization, and neural networks. Tree-based models (decision trees, random forest, gradient boosting) don't need scaling because splits are threshold-based and invariant to monotonic transforms.

What is target leakage and how do you prevent it?

Target leakage is when a feature contains information about the label that would not be available at prediction time (e.g., using post-outcome features, or fitting a scaler/encoder on the full dataset before splitting). Prevent it by fitting all transformations only on the training fold (use pipelines), splitting temporally when time matters, and reviewing features for future information.

Ordinal vs nominal encoding — how do you choose?

Ordinal encoding maps categories to integers (0, 1, 2, ...) — makes sense only for ordered categories (education level, star rating), or when passing to a tree-based model that will find splits regardless of numeric order. Never use it for unordered categories with a linear/distance model — the integers imply an ordering that doesn't exist.

What is target encoding and when is it useful?

Replace each category value with a statistic of the target for that category — typically the mean of y within the category (mean encoding). Great for high-cardinality categoricals (zip codes, product SKUs, user IDs) where one-hot creates thousands of columns.

How do you prevent leakage in target encoding?

Never compute the encoding on the same rows you'll evaluate. Techniques: (1) K-fold target encoding — for each fold, compute encoding from the other folds; (2) Leave-one-out encoding — for row i, exclude i from the statistic; (3) Smoothing — blend per-category mean with global mean by category count so tiny categories don't overfit; (4) Additive Gaussian noise on the encoded value; (5) CatBoost's ordered target encoding — the correct principled version.

What is frequency (count) encoding?

Replace each category with its count (or frequency) in the training set. Works well for tree models: high-frequency categories often behave differently from rare ones, and this captures that in one column instead of one-hot's thousands.

What is feature hashing (the 'hashing trick') and when is it useful?

Hash each category / n-gram to an index in a fixed-size vector $(eg,\ 2^{18}\ \text{columns})$ using a fast hash function. Handles unbounded / very high cardinality without an explicit vocabulary (users, URLs, streaming categoricals).

What is Weight of Evidence (WoE) encoding?

For a binary target and each category c: $\text{WoE}(c) = \ln(P(x = c \mid y = 1) / P(x = c \mid y = 0))$ — the log-ratio of positive-class share to negative-class share within that category, offset by the global class ratio. Comes from credit scoring: after WoE, categories are on a monotonic log-odds scale that plugs cleanly into logistic regression.

How do you encode cyclical features like hour-of-day or day-of-week?

Naive integer encoding breaks cyclicity: hour 23 and hour 0 look very far apart, but they're adjacent. Encode as two columns: sin(2*pi*x / P) and cos(2*pi*x / P) where P is the period (24, 7, 12, 365).

How do you turn text into features for a classical model?

Bag-of-words (BoW): count each token per document, sparse count matrix. TF-IDF: down-weights common terms by document frequency — usually beats raw counts.

You have a feature with 50,000 unique category values. How do you encode it?

One-hot is out — 50k sparse columns kill most models. Options: (1) target encoding (with K-fold + smoothing) for gradient boosting or linear models; (2) frequency encoding to distinguish common vs rare; (3) feature hashing to a fixed-size vector; (4) entity embeddings — a learned dense vector per category, trained with a neural net or via factorization machines; (5) group rare categories into 'Other' below a min-count threshold.

When should you manually engineer interaction features?

For linear / logistic models, always — they can't discover interactions by themselves. For deep nets and gradient boosting, only in cases where you know a specific multiplicative or ratio structure matters $(\text{BMI} = \text{weight}/\text{height}^2,\ \text{price} \cdot \text{volume for revenue})$.

When is discretization (binning) a useful feature transformation?

Binning turns a continuous feature into categorical buckets — great when (1) the relationship with y is highly non-linear and non-monotonic, so a linear model can't capture it; (2) you want to expose interactions between binned features cleanly; (3) domain knowledge suggests thresholds (age < 18, 18-65, > 65). Trees don't benefit — they discretize on their own.

StandardScaler vs MinMaxScaler vs RobustScaler — how do you choose?

StandardScaler: mean 0 / std 1. Best for approximately Gaussian data and models sensitive to variance (linear + regularization, SVM, PCA, neural nets).

Filter, wrapper, and embedded feature selection — how do they differ?

Filter: rank features by a statistic $(\text{mutual information},\ \chi^2,\ \text{ANOVA } F,\ \text{variance})$ — fast, model-agnostic, but ignores feature interactions and doesn't optimize for the downstream model. Wrapper: try different feature subsets, retrain the model, pick the best — most accurate but expensive (forward, backward, RFE).

How exactly does SMOTE generate synthetic minority-class samples?

For each minority-class point x, find its k nearest minority-class neighbours (default k=5). Pick one at random; call it x'.

What does ColumnTransformer do and when do you need it?

Applies different transformers to different subsets of columns in the same DataFrame and stitches the outputs back together into a single feature matrix. Standard use: one-hot encode categoricals, standardize numerics, drop or passthrough others — all in one step, fit inside a Pipeline.

You did StandardScaler().$\text{fit}_{\text{transform}}(X)$ then split into train/test. Why is this wrong?

The scaler was fit on the full dataset including the test set, so mean and std leak information from test data into training preprocessing. Effect: test metric is slightly optimistic, sometimes a lot on small data.

Why is target encoding dangerous, and how do you do it safely?

Target encoding replaces a category with the mean target for that category, so the target leaks directly into the feature. Fit it on the whole training set and the model memorizes rare categories, which look perfectly predictive in training and useless later.

Feature selection

How does Recursive Feature Elimination (RFE) work?

Train the model; rank features by importance $(\text{coefficients for linear},\ \text{feature}_{\text{importances}}\ \text{for trees})$; remove the least important K features; retrain; repeat until you reach the target count or the score peaks. RFECV wraps this in cross-validation to automatically pick the optimal number of features.

How does permutation importance work and when is it better than impurity-based importance?

Shuffle a feature's values on a held-out set, keeping the target intact; measure the drop in the model's metric. Larger drop → more important.

Can you use SHAP values for feature selection?

Yes — mean absolute SHAP value across a validation sample gives a robust, model-consistent importance ranking. Advantages over impurity: unbiased for cardinality, respects the model's actual behavior, works per-example.

When can aggressive feature selection actually hurt performance?

(1) Modern regularized models (L2, Lasso, elastic net) and modern gradient boosting handle irrelevant features via regularization — you rarely gain by manual selection. (2) Correlated informative features: dropping one from a redundant pair can lose subtle signal.

What does VarianceThreshold do and when is it useful?

Drops features with variance below a threshold — most commonly zero-variance (constants) that carry no information. Useful as a cheap pre-filter to remove degenerate features before training or before applying more expensive selection methods.

Mutual information vs chi-square vs ANOVA F for feature selection — how do you pick?

Mutual information: measures general (non-linear, non-monotonic) dependency between feature and target — the most powerful filter, works for both categorical and continuous. Chi-square: for categorical features + categorical target; tests independence in a contingency table.

Imbalanced data

How do you handle class imbalance in a dataset?

Options: (1) resampling — oversample minority (SMOTE) or undersample majority; (2) class weights in the loss function; (3) threshold tuning on the probability output; (4) anomaly-detection framing when positives are extremely rare; (5) collect more minority data. Always evaluate with PR-AUC, F1 or recall at fixed precision — not plain accuracy.

SMOTE, Borderline-SMOTE, SVM-SMOTE, ADASYN — when do you use each?

Vanilla SMOTE oversamples uniformly, including points deep in the class interior (redundant). Borderline-SMOTE: only synthesize from minority points near the decision boundary (their kNN mixes both classes) — where the model actually needs help.

Random undersampling vs Tomek links vs cluster-centroid undersampling — what's each for?

Random undersampling: drop majority-class rows uniformly to balance. Fast; wastes majority-class information.

What is focal loss and why does it help with class imbalance?

Focal loss = $-(1-p_t)^\gamma \cdot \log(p_t)$, a modification of cross-entropy that adds a modulating factor $(1-p_t)^\gamma$. When p_t is high (easy example), the loss is heavily down-weighted; when p_t is low (hard example), the modulator is close to 1 — focus on hard examples. $\gamma = 0$ recovers cross-entropy; $\gamma = 2$ is the standard for RetinaNet.

Class weights vs resampling for imbalance — which do you reach for first?

Class weights (or a weighted loss) first — they're a one-line change, preserve the dataset, don't discard data or synthesize points, and often match resampling in accuracy. If class weights aren't enough (extremely rare positives, non-linear boundary near the minority), try SMOTE variants for oversampling.

Missing data & outliers

MCAR, MAR, MNAR — what are these and why do they matter?

Missing Completely At Random (MCAR): missingness independent of everything — dropping rows is unbiased. Missing At Random (MAR): missingness depends only on observed features — imputation with these features is unbiased.

What is iterative (MICE) imputation?

Model each feature with missing values as the target of a regression/classification against all other features; iteratively predict and impute round-robin until convergence. Handles arbitrary correlations, mixed types (with the right per-column estimator), and uncertainty estimates when combined with multiple imputation (average predictions from several draws). scikit-learn's IterativeImputer with BayesianRidge is a solid default; use RandomForest for non-linear relationships.

IQR, Z-score, Isolation Forest, LOF — how do you pick an outlier detection method?

IQR: 1.5 * IQR beyond Q1/Q3 — robust univariate baseline, great for a quick 'known distribution' check. Z-score ($|z| > 3$): assumes Gaussian, sensitive to the very outliers you want to find.

Hyperparameter tuning

Grid search vs random search for hyperparameter tuning — which do you use?

Random search almost always beats grid search when you have more than 2-3 hyperparameters. Reason: most hyperparameters have few impactful settings; grid search wastes budget on irrelevant dimensions.

How does Bayesian hyperparameter optimization work at a high level?

Maintain a surrogate model of the objective (usually a Gaussian process or Tree-structured Parzen Estimator) as a function of hyperparameters. At each iteration: (1) predict the objective's mean and uncertainty across the search space; (2) use an acquisition function (Expected Improvement, UCB, or TPE-specific rules) to pick the next config — balancing exploration of uncertain regions and exploitation of promising ones; (3) run that config, update the surrogate.

What is Optuna's TPE sampler and why is it popular?

Tree-structured Parzen Estimator: for each hyperparameter, model $p(x \mid y > y_\cdot)$ and $p(x \mid y \leq y_\cdot)$ — two densities over past trials, split by an objective quantile. Pick the next config to maximize the ratio $p_{\text{good}}/p_{\text{bad}}$.

How do Successive Halving / Hyperband / ASHA speed up hyperparameter search?

Instead of running every config to completion, allocate a small budget to many configs, kill the worst half (or bottom N/eta), double the budget for survivors, repeat. Successive Halving: fixed budget schedule.

How do you tune when you have multiple objectives (accuracy AND latency AND size)?

Multi-objective optimization returns a Pareto frontier — configurations not dominated by any other on all objectives. Optuna's NSGA-II or the pymoo library support this natively.

Pipelines & leakage

Why should preprocessing live inside a scikit-learn Pipeline rather than being applied manually before?

Pipelines guarantee that every preprocessing step is fit on the *training fold* only (during CV, hyperparameter search, and grid search) and applied to validation/test. Manual preprocessing before splitting is the classic source of leakage: a StandardScaler fit on the full data has already seen the test set's variance.

Your new model has 60% AUC and the baseline had 75%. How do you debug?

Systematic checklist: (1) verify label correctness on validation — often the bug; (2) check for data leakage in the *baseline* (not the new model); (3) confirm train/val came from the same distribution — look for schema drift, temporal drift, missing categories; (4) inspect learning curves: is training loss even decreasing? (5) rule out preprocessing issues (unscaled features, wrong encoding, target leakage after refactor); (6) sanity-check with a very simple model (logistic regression) on the same features and pipeline.

You detect concept drift in production. What do you do?

(1) Quantify: is it data drift (features change) or concept drift ($\text{relationship } P(y \mid x) \text{ changes}$)? Different fixes.

How do you train and evaluate a model when labels arrive weeks after predictions?

(1) Choose an evaluation delay matching the label horizon — 'test set is data from more than N weeks ago'. (2) Never mix current features with future-only labels — check every feature for time-of-availability.

Your model has great offline metrics. What do you check before serving it in production?

(1) Latency at target p99 under realistic load. (2) Memory / cost at expected traffic.

Your model scores AUC 0.92 offline but barely helps in production. What are the usual causes?

Leakage is the first suspect: a feature that encodes the future, so it exists in training but not at inference. Second, training / serving skew — the feature is computed differently by the training job and the serving path.

How do you handle noisy labels in a supervised dataset?

First quantify it: relabel a random sample of a few hundred rows yourself and measure the disagreement rate, which bounds the accuracy any model can reach. Then reduce sensitivity: prefer robust losses such as Huber for regression, use label smoothing for classification, and avoid training to zero error since heavy overfitting memorizes the noise.

How do you decide how often to retrain a supervised model?

Measure rather than guess. Take the current model, evaluate it on successive time slices of held-out data, and plot how performance decays with the age of the training window.

Interview scenarios

Name three situations where you should NOT reach for machine learning.

(1) A hand-crafted rule solves it deterministically — 'if age < 18, deny' beats a model both in accuracy and interpretability. (2) No representative labeled data (or feasible way to get any) — you'll just fit noise.

What are the limits of SHAP values when explaining a model to a stakeholder?

SHAP explains the model, not the world: it attributes the model's output, so a spurious feature gets a large attribution if the model relies on it. With correlated features, the credit split between them is arbitrary, and swapping two collinear features can move attributions dramatically without changing predictions.

How do you justify shipping logistic regression over a boosted ensemble that scores better?

Frame it as total cost, not offline metric. If the gap is a fraction of a point of AUC and the decision threshold sits in a region where both models rank the same customers, the business outcome is identical.