

Choosing an Algorithm

A selection guide for machine learning, optimization and AI

Computational Labs · Michael Carter, Ph.D. · Free to share

104 methods across machine learning, optimization and AI, with the two things a comparison table usually leaves out: when each one is the right choice, and when it is the wrong one.

Start with the questions below. They route you to a section. Within each section the methods are ordered roughly from simplest to most capable — start at the top and stop when the result is good enough. The last page carries eight rules that matter more than the choice itself.

Start here

Seven questions. Each answer sends you to a section.

1 Do you have labelled examples of the answer you want?

- | | |
|--|-----------------------------------|
| » Yes — each row has a known outcome | Go to 2 |
| » No — you have data but no answers | Go to 4 |
| » Neither — you are searching for a best configuration | Go to 5 |
| » An agent acts and receives rewards over time | §12 Reinforcement learning |

2 Is the thing you are predicting a category or a number?

- | | |
|--|--------------------------------|
| » A category (spam / not spam, which species) | §1 Classification |
| » A number (price, demand, temperature) | §2 Regression |
| » A number that depends on its own past | §8 Time series |
| » Something structured — an image, text, audio | §7 Neural architectures |

3 Do you need to explain the prediction to someone?

- | | |
|---|--|
| » Yes, to a regulator, clinician or court | Linear/logistic regression, decision trees, Naive Bayes |
| » Helpful but not required | Random forest, gradient boosting with SHAP |
| » No — accuracy is all that matters | Gradient boosting, neural networks |

4 What are you trying to find in unlabelled data?

- » Groups of similar records
- » Fewer variables that keep the structure
- » The rare, the broken, the fraudulent
- » What to recommend to whom

§4 Clustering

§5 Dimensionality reduction

§6 Anomaly detection

§9 Recommendation and ranking

5 Can you compute a gradient of your objective?

- » Yes, and the function is smooth
- » No — it is noisy, discontinuous, or a simulation
- » Each evaluation is very expensive

§10 Gradient-based optimization

Go to 6

Bayesian optimization (§11)

6 What shape is a solution?

- » A point in continuous space (x, y, z ...)
- » An arrangement, route, schedule or subset
- » A set of hyperparameters for another model

§11 Continuous metaheuristics

§13 Combinatorial methods

§14 Hyperparameter search

7 Before you commit — always.

- » Run the simplest baseline first
- » If your sophisticated method cannot beat it

Logistic regression for classification, the mean for regression, random-restart hill climbing for search

The problem is usually your data or your objective, not your algorithm

1. Classification — predicting a category

Ordered roughly from simplest to most capable. Start at the top and stop when accuracy is sufficient.

Method	Use it when	Avoid it when
Logistic regression	You want a fast, well-calibrated baseline with interpretable coefficients. Almost always the right first model.	Relationships are strongly non-linear and you cannot engineer features to fix it.
Naive Bayes	Text or categorical features, small data, and you need speed. Surprisingly hard to beat on document classification.	You need the probabilities themselves to be trustworthy — it is confident far beyond what it knows.
k-Nearest Neighbours	Decision boundaries are irregular, data is small, and you want no training step at all.	Data is large or high-dimensional. Prediction cost grows with the dataset, and distance loses meaning in many dimensions.
Decision tree (CART)	You need a model a non-specialist can read off as rules, and interactions matter.	You need stability — small changes in data produce entirely different trees.

Method	Use it when	Avoid it when
Random forest	A strong general-purpose default. Handles mixed types, needs little tuning, resists overfitting.	You need a single interpretable rule set, or inference must be very fast.
Gradient boosting (XGBoost, LightGBM, CatBoost)	Tabular data where accuracy matters most. Usually the winner on structured problems.	You have little data, no time to tune, or you need calibrated probabilities without extra work.
AdaBoost	A simple, well-understood boosting baseline on clean data.	Your labels are noisy — it concentrates on hard cases, and mislabelled points are hard cases.
Support vector machine	Clear margin between classes, moderate sample size, high dimensions relative to samples.	You have hundreds of thousands of rows, or you need probability estimates directly.
Linear / quadratic discriminant analysis	Classes are roughly Gaussian with similar covariance. Very few parameters, very fast.	Distributions are far from Gaussian or strongly overlapping.
Multilayer perceptron	Non-linear structure that trees handle badly, and enough data to fit it.	Tabular data of moderate size — boosted trees usually win, with less effort.

2. Regression — predicting a number

Method	Use it when	Avoid it when
Ordinary least squares	The baseline. Interpretable, fast, and the reference every other model should beat.	Predictors are strongly collinear or the relationship is clearly non-linear.
Ridge regression	Many correlated predictors. Shrinks coefficients without dropping any.	You want automatic variable selection — ridge keeps everything.
Lasso	You want the model to pick a small subset of predictors for you.	Predictors are highly correlated — lasso picks one arbitrarily and zeroes its twins.
Elastic net	The common case: many predictors, some correlated, and you want selection anyway.	You have few predictors and no collinearity — plain OLS is simpler.
Polynomial regression	A known curved relationship in one or two variables.	Many variables — the term count explodes and extrapolation becomes wild.
Decision tree / random forest regressor	Non-linear, interaction-heavy tabular data.	You need to extrapolate beyond the training range — trees cannot.
Gradient boosting regressor	Best accuracy on tabular regression, given time to tune.	Small data or a need for smooth, monotone predictions without constraints.
Support vector regression	Moderate data, robustness to outliers within a tolerance band.	Large datasets — training scales poorly.
Gaussian process regression	Small data and you need honest uncertainty estimates with every prediction.	More than a few thousand points — cost grows cubically.
Quantile regression	You care about the tails, not the average — worst-case delivery time, not mean.	A single central estimate is genuinely what the decision needs.

3. Ensembles — combining models

Not algorithms so much as strategies applied to the ones above.

Method	Use it when	Avoid it when
Bagging	Your base model is high-variance and unstable, such as a deep tree.	The base model is already stable and biased — bagging will not fix bias.
Boosting	You want maximum accuracy and can afford sequential training.	Labels are noisy, or you need parallel training on huge data.
Stacking	Several genuinely different models each capture something the others miss.	Models are highly correlated — you add complexity for almost nothing.
Voting / averaging	The cheapest ensemble. A quick, reliable improvement over any single member.	One member is far better than the rest — averaging drags it down.

4. Clustering — finding groups

Method	Use it when	Avoid it when
k-Means	You can guess the number of clusters, and groups are roughly spherical and similar in size.	Clusters are elongated, nested, or wildly different in density.
k-Medoids (PAM)	Same as k-means but you need centres to be actual data points, or robustness to outliers.	Large datasets — considerably slower than k-means.
Hierarchical (agglomerative)	You want the whole tree of nestings and no fixed k in advance. The dendrogram is itself the output.	More than a few tens of thousands of points.
DBSCAN	Clusters have arbitrary shape, and you want outliers labelled as noise rather than forced into a group.	Cluster densities vary a lot — one epsilon cannot serve all of them.
OPTICS	DBSCAN's problem, fixed: varying densities in one dataset.	You want a simple result — the reachability plot needs interpretation.
Gaussian mixture model	Clusters overlap and you want soft membership probabilities rather than hard assignment.	Clusters are clearly non-Gaussian in shape.
Mean shift	You do not know k and clusters are blob-like. Finds modes of the density.	High dimensions, or large data — bandwidth selection is fiddly and cost is high.
Spectral clustering	Clusters are connected but not compact — rings, spirals, graph communities.	Very large data; it requires an affinity matrix over all pairs.

5. Dimensionality reduction — fewer variables

Method	Use it when	Avoid it when
Principal component analysis	The default. Linear compression, noise reduction, and a fast look at the structure.	Structure is non-linear, or you need components to mean something physically.

Method	Use it when	Avoid it when
Kernel PCA	Non-linear structure, but you still want a projection you can apply to new data.	You cannot justify a kernel choice, or the dataset is large.
Linear discriminant analysis	You have labels and want the projection that best separates classes, not the one with most variance.	No labels, or classes are far from Gaussian.
t-SNE	Visualization in two dimensions, to see whether clusters exist at all.	Anything downstream. Distances between clusters are not meaningful and it cannot transform new points.
UMAP	Like t-SNE but faster, preserves more global structure, and can map new points.	You need a mathematically interpretable projection.
Independent component analysis	Signals are mixed and you want to separate the sources — audio, EEG.	You want variance-ordered components. ICA finds independence, not importance.
Non-negative matrix factorization	Data is non-negative and you want additive, parts-based components — topics, spectra.	Data contains meaningful negative values.
Autoencoder	Non-linear compression with lots of data and a need to reconstruct.	Small data — a linear method will do better with far less effort.

6. Anomaly detection — finding the unusual

Method	Use it when	Avoid it when
Isolation forest	General-purpose default. Fast, scales well, few assumptions.	Anomalies are only anomalous in context — a normal value at an abnormal time.
One-class SVM	You have a clean sample of normal behaviour and want a tight boundary around it.	Data is large, or the training set is already contaminated.
Local outlier factor	Anomalies are local — normal for the dataset, abnormal for their neighbourhood.	You need to score new points without refitting.
Elliptic envelope / Mahalanobis distance	Data is roughly Gaussian and you want a principled statistical threshold.	Distributions are skewed or multi-modal.
Autoencoder reconstruction error	High-dimensional data — images, sensor streams — with plenty of normal examples.	Small data, or you need to explain why something was flagged.

7. Neural architectures

Method	Use it when	Avoid it when
Feedforward network (MLP)	Tabular or vector input with non-linear structure and sufficient data.	Modest tabular data. Boosted trees usually beat it for less effort.
Convolutional network (CNN)	Grid-structured data where position matters locally — images, spectrograms.	Features have no spatial relationship to exploit.

Method	Use it when	Avoid it when
Recurrent network (RNN)	Short sequences, and you want the simplest recurrent baseline.	Long dependencies — gradients vanish. Use LSTM/GRU or a transformer.
LSTM / GRU	Sequences with long-range dependencies and moderate data.	Very long sequences or large-scale training — transformers dominate here now.
Transformer	Long sequences, large data, and the compute to train or fine-tune.	Small datasets. They are data-hungry and overfit small corpora quickly.
Attention mechanism	You need the model to weigh which inputs matter for each output — and to show you.	Sequences are short and uniform; the added cost buys little.
Autoencoder	Compression, denoising, or pretraining without labels.	You need generation — a plain autoencoder's latent space is not smooth.
Variational autoencoder	Generation with a structured, continuous latent space you can interpolate through.	You need sharp, photorealistic samples.
Generative adversarial network	Sharp, realistic sample generation.	You need stable training or a likelihood. Diffusion models have largely superseded them.
Diffusion model	State-of-the-art generation quality for images and audio.	Inference must be fast — generation takes many sequential steps.
Graph neural network	Data is genuinely a graph — molecules, social networks, road systems.	You can flatten to a table without losing the relationships.

8. Time series and forecasting

Method	Use it when	Avoid it when
Exponential smoothing / Holt-Winters	Short series, clear trend and seasonality, and you need something defensible quickly.	Multiple interacting drivers you need to model explicitly.
ARIMA / SARIMA	A single stationary series with autocorrelation structure worth modelling properly.	Many series, or strong external drivers — the specification burden grows fast.
Decomposition-based methods (STL, Prophet-style)	Business series with holidays, multiple seasonalities and missing data.	High-frequency data where the additive assumption breaks down.
Gradient boosting on lag features	Often the strongest practical forecaster. Turn the series into a supervised table.	You need honest prediction intervals without extra machinery.
LSTM / temporal CNN	Many related series and enough data to learn shared structure.	One short series — classical methods will beat it comfortably.

9. Recommendation and ranking

Method	Use it when	Avoid it when
Content-based filtering	You have good item features and a cold-start problem for new users.	Items are hard to describe, or you want serendipitous recommendations.

Method	Use it when	Avoid it when
Collaborative filtering (neighbourhood)	Plenty of interaction data and you want explainable 'people like you' logic.	Sparse data or new items — the cold-start problem is severe.
Matrix factorization	Large, sparse interaction matrices. The workhorse of practical recommenders.	You need to explain a recommendation to a user.
Learning to rank	Ordering matters more than the score — search results, feeds.	You genuinely need calibrated scores rather than an ordering.

10. Gradient-based optimization

How models are actually fitted. If you can compute a gradient, use one of these — not a metaheuristic.

Method	Use it when	Avoid it when
Batch gradient descent	Small datasets and a smooth objective. Stable, predictable descent.	Large data — every step needs a full pass over everything.
Stochastic gradient descent	Large datasets. Noisy steps, but many more of them per unit of time.	You need a smooth, monotone decrease in loss.
Momentum / Nesterov	The objective has ravines — steep in one direction, shallow in another.	The surface is already well-conditioned; momentum can overshoot.
AdaGrad	Sparse features where rare ones deserve larger steps.	Long training runs — the effective learning rate decays towards zero.
RMSprop	AdaGrad's decay problem fixed. Solid for recurrent networks.	You want one optimizer for everything — Adam is the safer default.
Adam	The default for neural networks. Adaptive, forgiving of a poor initial learning rate.	You need best-in-class generalization — well-tuned SGD with momentum often edges it out.
Newton's method	Few parameters and a smooth objective. Converges in remarkably few iterations.	Many parameters — the Hessian is quadratic in size and expensive to invert.
BFGS / L-BFGS	Newton's speed without the full Hessian. Excellent for classical models.	Stochastic or noisy objectives; mini-batch training.
Conjugate gradient	Large sparse linear systems and quadratic objectives.	Strongly non-quadratic surfaces.
Coordinate descent	Objectives that decompose per variable — this is how lasso is actually solved.	Variables are strongly coupled.

11. Continuous metaheuristics

For objectives you cannot differentiate: simulations, black boxes, noisy measurements.

Method	Use it when	Avoid it when
Hill climbing	The baseline every other method must beat. With random restarts it is a serious competitor.	The landscape has many deep local optima and restarts are too expensive.

Method	Use it when	Avoid it when
Simulated annealing	One rugged landscape, cheap evaluations, and you want a simple method with one main dial.	Evaluations are expensive — it uses a lot of them, one at a time.
Particle swarm optimization	Continuous spaces, few parameters to set, fast convergence on smooth-ish landscapes.	The problem is combinatorial, or you need guaranteed diversity late in the run.
Differential evolution	Continuous problems where you do not want to tune a step size — it scales itself.	The population may converge prematurely; without restarts it cannot recover.
Grey wolf optimization	Few parameters, quick convergence, a straightforward implementation.	You need sustained exploration — the pack follows its three leaders closely.
Artificial bee colony	You want an explicit mechanism for abandoning exhausted solutions.	You need fast convergence; the scout phase costs evaluations.
Genetic algorithm	Solutions decompose into parts that can be recombined meaningfully.	Your encoding puts unrelated things next to each other — crossover then destroys structure.
CMA-ES	The strongest general-purpose choice on hard continuous problems up to a few hundred dimensions.	Very high dimensions — the covariance matrix becomes the bottleneck.
Bayesian optimization	Each evaluation costs minutes or hours. Builds a surrogate model to spend evaluations wisely.	Evaluations are cheap — the surrogate overhead is not worth it.

12. Reinforcement learning

Method	Use it when	Avoid it when
Multi-armed bandits	One decision repeated, no state. A/B testing that adapts as it learns.	Actions change the situation for later decisions.
Q-learning	Small discrete state and action spaces. Learns the optimal policy while following another.	Continuous or very large state spaces.
SARSA	You need the policy learned to be the policy followed — safety matters during learning.	You want the optimal policy regardless of exploration cost.
Deep Q-network (DQN)	Large or continuous states, discrete actions.	Continuous action spaces.
Policy gradient (REINFORCE)	Continuous actions, or you need a stochastic policy.	You cannot tolerate high variance and slow convergence.
Actor-critic (A2C, A3C)	Policy gradient with the variance reduced by a learned value function.	Simple problems where a value method suffices.
PPO	The practical default. Stable, forgiving, widely used including in language-model tuning.	You need maximum sample efficiency — off-policy methods reuse data better.

13. Combinatorial methods

When a solution is an arrangement: a route, a schedule, an assignment, a subset.

Method	Use it when	Avoid it when
Greedy construction	You need an answer immediately and a good-enough one will do. Also the starting point for local search.	Quality matters — greedy commits early and cannot reconsider.
Local search (2-opt, k-opt)	Improving an existing solution by small rearrangements. Cheap and very effective.	Used alone — it stops at the first local optimum.
Tabu search	Local search that keeps moving by forbidding recent moves. Strong on scheduling and routing.	You cannot define a sensible move attribute to forbid.
Ant colony optimization	Solutions are paths through a graph, and you want the memory to live in the problem, not the agents.	The solution is a point in continuous space.
Genetic algorithm (permutation-coded)	Ordering problems where specialised crossover operators exist.	You are using generic crossover on a permutation — it usually produces invalid children.
Branch and bound	You need a provably optimal answer and the instance is small enough.	Large instances — the search tree grows explosively.
Dynamic programming	The problem has optimal substructure and overlapping subproblems — shortest paths, knapsack, alignment.	The state space is too large to tabulate.
Integer / linear programming	Constraints and objective are linear and a solver can be applied. Often beats every heuristic.	The objective is non-linear or comes from a simulation.

14. Hyperparameter search

Method	Use it when	Avoid it when
Manual search	You understand the method and know which two parameters matter. Frequently the fastest route.	You are guessing — then it is just slow random search.
Grid search	Two or three parameters and you want exhaustive, reproducible coverage.	More than three — the cost is the product of every axis.
Random search	Many parameters, only a few of which matter. Reliably beats grid search per unit of compute.	You have a very small budget and strong prior knowledge.
Bayesian optimization	Each training run is expensive and you have a few dozen evaluations at most.	Training is fast — random search will find as much, sooner.
Successive halving / Hyperband	Many configurations, and poor ones can be identified early and killed.	Performance early in training does not predict performance at the end.

Eight rules that matter more than the choice

Everything above assumes these are already handled. They usually are not.

Rule	Why
Run the baseline first, always.	Logistic regression for classification, the mean for regression, random-restart hill climbing for search. If the sophisticated method cannot beat it, the problem is your data or your objective.
No algorithm is best in general.	The No Free Lunch theorem says that averaged over all possible problems, every method performs identically. Performance is a claim about a class of problems, not a property of an algorithm.
More data usually beats a better algorithm.	The gap between two good methods is typically smaller than the gap made by fixing your labels, your features or your sampling.
Decide what you are optimizing before you optimize it.	Accuracy, recall, cost-weighted error and calibration lead to different winners. An inverted or wrongly-weighted objective produces a beautiful convergence curve and a worthless answer.
Interpretability is a requirement, not a preference.	If a decision must be explained to a regulator, a clinician or a court, that constraint eliminates most of this document before you compare anything.
Match the representation to the problem before tuning anything.	Ant colony wants paths. Differential evolution wants continuous vectors. Crossover wants related things adjacent in the encoding. Most failures are representation failures.
One run of a stochastic method tells you nothing.	Run it thirty times with different seeds and plot the band. Half the published comparisons in this field would look considerably less impressive under that standard.
You cannot debug what you cannot compute by hand.	Whatever you choose, work one iteration on the smallest possible instance — three data points, two dimensions. It takes twenty minutes and it is the difference between engineering and guessing.

Free worked examples, hand calculations and a notation reference at computationallabs.com/resources