Linear & Logistic Regression

Linear and logistic regression are the foundational algorithms for supervised learning. Despite their simplicity, they remain useful in machine learning applications, serving as strong baselines and interpretable models especially when data relationships are approximately linear. This guide provides applied, actionable insights for practitioners at all levels.

1. Model Structure

Linear models learn a set of weights (coefficients) for each input feature to produce predictions. The core principle is that the output is a weighted sum of the inputs, optionally transformed through a non-linear function in the case of classification.

1.1 Linear Regression

Linear regression predicts a continuous target variable by fitting a linear equation to the observed data. The model minimises the difference between the predicted values and the actual target values.

Mathematical Form

The model learns coefficients $w_1, w_2, ..., w_n$ and an intercept b such that:

$$\hat{y} = w_1x_1 + w_2x_2 + ... + w_nx_n + b$$

Key Characteristics

- Assumes a linear relationship between features and the target
- Output is unbounded — can take any real value
- Coefficients directly indicate feature importance and direction of influence

Best Practice

Use linear regression when your goal is to predict continuous outcomes like revenue, temperature, or sales volume. If you observe strong non-linearity in feature scatter plots, consider polynomial features or move to tree-based models.

1.2 Logistic Regression

Logistic regression is used for classification tasks where the target is categorical. Unlike linear regression, it outputs probabilities bounded between 0 and 1 using a non-linear transformation function.

Binary Logistic Regression

Binary logistic regression is employed when the target has exactly two classes (e.g., Yes/No, True/False, Approved/Rejected). It applies the sigmoid function to transform a linear combination of features into a probability.

Mathematical Form

The linear combination $z = w_1x_1 + w_2x_2 + ... + w_nx_n + b$ is transformed via the sigmoid function:

$$P(y=1 \mid x) = 1 / (1 + e^{-z})$$

The output probability ranges from 0 to 1. A decision threshold (commonly 0.5) is applied: probabilities above the threshold are classified as Class 1, below as Class 0.

Key Characteristics

- Outputs calibrated probabilities suitable for risk scoring
- Decision boundary is linear in feature space
- Sensitive to class imbalance — may require threshold tuning or resampling
- Coefficients indicate log-odds contribution of each feature

Multinomial Logistic Regression (Softmax)

Multinomial logistic regression extends binary classification to handle multiple classes. Instead of a single sigmoid output, the model uses the softmax function to produce a probability distribution across all classes.

Mathematical Form

For each class k, compute a score z_k, then normalise using softmax:

$$P(y=k \mid x) = e^{z_k} / \Sigma_j e^{z_j}$$

The class with the highest probability is selected as the prediction. Softmax ensures all class probabilities sum to 1.

Key Characteristics

- Handles any number of classes (3+)
- Produces a full probability distribution over all classes
- Uses one-hot encoded labels during training
- More parameters than binary classification — each class has its own weight vector

2. Loss Functions

The loss function quantifies how far the model's predictions are from the actual target values. Choosing the right loss function is critical as it directly influences what the optimiser minimises during training.

2.1 Regression Loss Functions

Mean Squared Error (MSE)

MSE is the default loss function for regression. It squares the difference between predicted and actual values, heavily penalising large errors.

- Suitable when large deviations are undesirable
- Sensitive to outliers (extreme values disproportionately inflate the loss)
- Smooth gradient makes optimisation straightforward

Mean Absolute Error (MAE)

MAE computes the absolute difference between predictions and targets, treating all errors equally regardless of magnitude.

- Robust to outliers (extreme values have linear impact)
- Preferred when the dataset contains noise or anomalous observations
- Gradient is constant, which can slow convergence near the optimum

Best Practice

Start with MSE as your default regression loss. If residual analysis reveals significant outliers that are not errors but legitimate edge cases, switch to MAE to reduce their influence.

2.2 Classification Loss Functions

Log Loss (Binary Cross-Entropy)

Log loss measures the accuracy of probability predictions for binary classification. It penalises confident incorrect predictions exponentially.

- Encourages the model to output calibrated probabilities
- Heavily penalises predictions far from the true label
- Works on the principle of maximum likelihood estimation

Categorical Cross-Entropy

Categorical cross-entropy is the natural extension of log loss for multi-class problems. It compares the predicted probability distribution against the true one-hot encoded label.

- Suitable for all multi-class classification tasks
- Requires one-hot encoding of target labels
- Compatible with softmax output layer

Comparison Summary

Loss Function	Task Type	Strengths	Limitations
MSE	Regression	Default choice; penalizes large errors quadratically	Sensitive to outliers; magnifies extreme errors
MAE	Regression	Robust to outliers; treats all errors equally	Less aggressive penalty for large errors; slower convergence
Log Loss (Binary Cross-Entropy)	Binary Classification	Penalizes confident wrong predictions heavily; probabilistic output	Requires calibrated probabilities
Categorical Cross-Entropy	Multi-class Classification	Natural extension for multiple classes; works with softmax	Requires one-hot encoded labels

3. Regularization

Regularization prevents overfitting by adding a penalty term to the loss function that discourages overly complex models. This ensures that the model generalises well to unseen data rather than memorising the training set.

Business Context

In production environments, regularization is important for model reliability. A model that overfits training data may perform well in testing but fail catastrophically when deployed on new customers, market conditions, or seasonal patterns. Regularization acts as insurance against this risk.

3.1 L1 Regularization (Lasso)

L1 regularization adds the sum of absolute values of the model's weights to the loss function. This encourages sparsity, many coefficients are driven to exactly zero.

Key Properties

- Performs automatic feature selection by zeroing out less important features
- Produces interpretable, compressed models with fewer active predictors
- Useful when you suspect many features are irrelevant or redundant

Lambda (Regularization Strength)

Typical range: 0.001 to 1.0

- Small lambda (≈ 0.001): Minimal regularization; close to unregularised model
- Medium lambda (≈ 0.01 – 0.1): Balanced sparsity and performance
- Large lambda (≈ 1.0): Aggressive sparsity; many features eliminated

⚠ Warning

L1 regularization may arbitrarily select one feature from a group of highly correlated predictors and drop the others. If all correlated features are important for interpretation, consider L2 or Elastic Net instead.

3.2 L2 Regularization (Ridge)

L2 regularization adds the sum of squared weights to the loss function. This penalises large coefficient values without forcing any to zero.

Key Properties

- Shrinks all coefficients toward zero but keeps all features in the model
- Stabilises training by preventing any single feature from dominating
- Most commonly used regularization technique due to smooth, reliable behaviour

Lambda (Regularization Strength)

Typical range: 0.0001 to 0.1

- Small lambda (≈ 0.0001): Light regularization for mildly complex models
- Medium lambda (≈ 0.001 – 0.01): Standard choice for most applications
- Large lambda (≈ 0.1): Strong shrinkage for high-dimensional data

3.3 Elastic Net (L1 + L2 Combined)

Elastic Net combines L1 and L2 penalties, providing both feature selection (from L1) and coefficient stability (from L2). It is especially effective when dealing with groups of correlated features.

Key Properties

- Balances sparsity and stability
- Does not arbitrarily drop correlated features like pure L1
- Requires tuning two hyperparameters (L1 and L2 lambda values)

Comparison Summary

Method	Lambda Range	Strengths	Limitations
L1 (Lasso)	0.001 – 1.0	Feature selection; sparse models	May arbitrarily drop correlated features
L2 (Ridge)	0.0001 – 0.1	Stability; prevents large weights	Keeps all features; no automatic selection
Elastic Net	L1: 0.001 – 1.0 L2: 0.0001 – 0.1	Balances sparsity and stability	Requires tuning two hyperparameters

Best Practice

Start with L2 regularization as the default. If feature selection is critical or you have hundreds of features, switch to L1 or Elastic Net. Use cross-validation to tune the lambda values rather than guessing.

4. Optimization Techniques

Optimization methods determine how the model's coefficients are updated during training to minimise the loss function. The choice of optimizer affects training speed, stability, and final model quality.

4.1 Gradient Descent

Gradient descent is the fundamental optimization algorithm. At each iteration, it computes the gradient of the loss function with respect to the model parameters and moves in the direction of steepest descent.

Learning Rate (Alpha)

Typical range: 0.001 to 0.1

- Controls the step size for each parameter update
- Small alpha (≈ 0.001): Slow, stable convergence; suitable for noisy gradients
- Large alpha (≈ 0.1): Fast initial progress; risk of overshooting the minimum

Key Characteristics

- Simple and predictable on convex loss surfaces (like linear models)
- Sensitive to learning rate, requires tuning for optimal performance
- Can be slow on large datasets without mini-batch or stochastic variants

4.2 Gradient Descent with Momentum

Momentum improves gradient descent by incorporating information from previous update steps. This accelerates convergence in consistent directions and reduces oscillations.

Hyperparameters

- **Alpha (Learning Rate):** 0.001 to 0.1 — same as standard gradient descent
- **Beta (Momentum Coefficient):** 0.8 to 0.99
  - Higher beta (≈ 0.99): Smooth, long-term trajectory; slower reaction to changes
  - Lower beta (≈ 0.8): Faster response to new gradient directions

Key Characteristics

- Faster convergence than vanilla gradient descent on most problems
- Helps escape shallow local minima and plateaus (less relevant for convex linear models)
- Adds one additional hyperparameter to tune

4.3 Adam Optimizer

Adam (Adaptive Moment Estimation) computes adaptive learning rates for each parameter using moving averages of both gradients and squared gradients.

Hyperparameters

- Alpha (Learning Rate): 0.001 to 0.1
- Beta1 (First Moment): 0.9 — controls momentum
- Beta2 (Second Moment): 0.999 — stabilises updates using gradient variance

Key Characteristics

- Adaptive per-parameter learning rates — no manual tuning needed for most problems
- Robust default choice across a wide range of datasets and architectures
- Slight computational overhead compared to basic gradient descent
- Often overkill for simple linear models but useful for complex neural networks

Comparison Summary

Optimizer	Hyperparameters	Strengths	When to Use
Gradient Descent	Alpha: 0.001 – 0.1	Simple; predictable convergence on convex loss	Can be slow; sensitive to learning rate

Optimizer	Hyperparameters	Strengths	When to Use
Momentum	Alpha: 0.001 – 0.1 Beta: 0.8 – 0.99	Faster convergence; smooths oscillations	Adds one extra hyperparameter
Adam	Alpha: 0.001 – 0.1 Beta1: 0.9 Beta2: 0.999	Adaptive per-parameter rates; robust default	Slight overhead; overkill for simple linear models

Best Practice

For linear and logistic regression, start with standard gradient descent or momentum. Adam is more beneficial when training neural networks or handling complex, non-convex loss surfaces. Always monitor training loss to verify convergence regardless of optimizer choice.

5. Learning Rate Decay

Learning rate decay gradually reduces the learning rate during training. For linear models, this is optional because the loss surface is convex and smooth. However, decay can help achieve more precise convergence near the optimum.

5.1 Exponential Decay

Exponential decay reduces the learning rate by a constant percentage after each epoch or iteration.

- Formula: alpha_new = alpha_initial × decay_rate^epoch
- Useful for long training runs requiring gradual refinement
- Provides smooth, continuous reduction

Note: *Users should input γ = 0.95 to mean "reduce by 5% each epoch", not enter 0.05 expecting a 5% reduction*

5.2 Step-wise Decay

Step-wise decay reduces the learning rate by a fixed factor at predetermined intervals.

- Simple and predictable — often drops by 50% or 10× at set epochs
- Suitable when training duration is known in advance
- Less smooth than exponential decay but easier to configure

5.3 Loss-Deviation Decay (Adaptive)

Loss-deviation decay monitors validation loss and reduces the learning rate only when progress stalls.

- Data-driven approach — adapts to actual training dynamics
- Ideal when plateau or stagnation indicates the need for finer adjustments
- Requires validation set and adds slight complexity

> **Note**
> For most linear regression and logistic regression tasks, learning rate decay is not necessary. The convex loss surface ensures reliable convergence with a fixed learning rate. Use decay primarily when training neural networks or when you observe oscillations near convergence.

6. Convergence Criteria

Convergence criteria determine when to stop training. Proper stopping conditions prevent both underfitting (stopping too early) and wasting compute resources (continuing after the model has converged).

Available Criteria

Criterion	Mechanism	Advantages	Considerations
Loss Threshold	Stop when loss < target	Clear goal; useful for fixed tolerance	May stop prematurely if target is too loose
Iteration Count	Fixed number of epochs/steps	Predictable training time	No guarantee of convergence
Gradient Threshold	Stop when gradient magnitude < ε	Reliable for convex problems	Requires choosing appropriate ε
Validation-Based Early Stopping	Stop when validation loss stops improving	Prevents overfitting; data-driven	Requires held-out validation set

6.1 Loss Threshold

Training stops once the loss falls below a predefined target value.

- Practical when the goal is to meet a specific error tolerance
- Reliable for linear models due to convex loss surfaces
- Risk: May stop prematurely if the threshold is set too loosely

6.2 Iteration Count

Training runs for a fixed number of epochs or passes through the data.

- Simplest approach — guarantees predictable training time
- Does not guarantee convergence on its own
- Often combined with other criteria for safety

6.3 Gradient Threshold

Training stops when the magnitude of the gradient (or the change in loss between iterations) drops below a tolerance level.

- Highly reliable for convex problems like linear regression

- Indicates that coefficients have stabilised
- Requires choosing an appropriate epsilon value (e.g., 1e-6 to 1e-4)

6.4 Validation-Based Early Stopping

Training halts when validation performance stops improving over a specified number of iterations (patience).

- Prevents overfitting to training noise
- Requires a held-out validation set
- Most useful for regularised models or high-dimensional feature spaces

Best Practice

For linear models, combine iteration count with gradient threshold or validation-based early stopping. This ensures you stop training once convergence is reached without risking premature termination or excessive compute waste.

7. Status Notification & Monitoring

During training, displaying loss values at regular intervals provides visibility into the optimization process. This helps diagnose issues like divergence, slow convergence, or overfitting.

7.1 What to Monitor

Training Loss Only

Displays the loss computed on the training data at each logging interval.

- Suitable when no validation set is available
- Lower computational overhead — no need to evaluate on validation data
- Provides basic confirmation that optimization is progressing

Training + Validation Loss

Displays both training loss and validation loss at each interval.

- Provides deeper insight into model generalisation
- Helps diagnose overfitting (training loss decreases, validation loss increases)
- Essential for validation-based early stopping

Best Practice

Always monitor validation loss when using regularization, early stopping, or working with high-dimensional data. The slight computational cost is usually worth the visibility into generalisation behaviour.

7.2 Logging Frequency

The frequency parameter controls how often loss values are displayed.

- Small frequency (e.g., every 10 iterations): More granular updates; useful for debugging convergence issues
- Large frequency (e.g., every 100 iterations): Reduces logging overhead; suitable for long training runs

Trade-offs

- **High frequency:** Better visibility but slightly slower training due to logging overhead
- **Low frequency:** Minimal overhead but less insight into iteration-by-iteration progress

Note

For production pipelines, log at a moderate frequency (e.g., every 50-100 iterations) and save the full training history to disk for post-hoc analysis. This balances real-time visibility with performance.

8. Practical Recommendations

The following guidelines synthesize best practices from the sections above into a concise decision framework.

Model Selection

- Use linear regression for continuous targets; logistic regression for categorical targets
- Start with binary logistic for two-class problems; use multinomial (softmax) for 3+ classes
- Validate that linear assumptions hold — check residual plots for systematic patterns

Loss Function

- Default to MSE for regression unless outliers are present, then use MAE
- Always use log loss (binary cross-entropy) for binary classification
- Always use categorical cross-entropy with softmax for multi-class classification

Regularization

- Apply L2 regularization by default with lambda ≈ 0.001 – 0.01
- Use L1 when you need feature selection or suspect many irrelevant predictors
- Use Elastic Net for correlated feature groups where Lasso may be too aggressive

- Tune lambda using cross-validation on a held-out validation set

Optimization

- Start with gradient descent (alpha ≈ 0.01) or momentum (beta ≈ 0.9)
- Reserve Adam for neural networks or when gradient descent converges slowly
- Monitor training loss — if it plateaus or oscillates, reduce the learning rate

Convergence

- Combine iteration count (e.g., 1000 epochs) with gradient threshold or early stopping
- Use validation-based early stopping when working with regularized or high-dimensional models
- Set a reasonable maximum iteration count to prevent runaway training

Monitoring

- Always log both training and validation loss if a validation set is available
- Save training history to disk for post-training analysis and reproducibility

9. Underfitting – Overfitting Tradeoff

Achieving strong model performance is a critical stage in any machine learning pipeline, where the goal is not just to learn patterns from training data, but to ensure those patterns hold on unseen data. While high training accuracy may appear promising, real-world success depends on balancing underfitting and overfitting.

9.1 Linear Regression

Because the model form is fixed and simple, it is naturally prone to underfitting when the true relationship is non-linear or involves interactions between variables. Overfitting becomes a risk when many features are added relative to the number of training examples, causing the model to chase noise rather than signal.

Aspect	▽ UNDERFITTING		△ OVERFITTING	
	Causes	Actions to Fix	Causes	Actions to Fix
Model Complexity	Model assumes a purely linear relationship; cannot capture curves or interactions in the data	Introduce polynomial or interaction terms to allow the model to bend and capture non-linear patterns	Adding too many engineered features inflates model complexity beyond what the data supports	Apply regularization (Ridge or Lasso) to penalise unnecessary complexity and shrink redundant coefficients

Feature Quality	Important predictors are absent; the model lacks the information needed to explain variance in the target	Perform feature engineering to surface domain-relevant signals; include interaction or ratio features	Too many correlated or irrelevant features dilute the signal and allow the model to fit noise	Remove highly correlated features; use feature selection techniques to retain only informative predictors
Regularization Strength	Penalty on coefficients is too strong, shrinking them towards zero and suppressing genuine signals	Reduce the regularization strength gradually; validate using cross-validation to find the right balance	No regularization is applied, leaving coefficients free to grow large and overfit noise	Introduce L2 (Ridge) to shrink all coefficients, or L1 (Lasso) to zero out irrelevant ones entirely
Training Data Volume	Too few examples to reliably estimate even a simple linear relationship	Collect more data; clean outliers and label errors that prevent the model from finding the true trend	Very small dataset with many predictors — the model has more degrees of freedom than data points	Obtain more training samples; use cross-validation to get a reliable estimate of generalisation performance
Bias / Variance Trade-off	High bias — the linearity assumption is too rigid for the actual data structure	Relax the linearity assumption by transforming features or moving to a more flexible model class	High variance — the model is sensitive to the exact training sample and memorises its noise	Constrain model flexibility through regularization; increase training data to reduce variance

9.2 Logistic Regression

Logistic Regression tends to underfit when classes cannot be separated linearly, and overfits when the feature space is high-dimensional relative to available labelled examples. Its performance is therefore tightly coupled to how well the features represent the underlying class structure.

Aspect	▽ UNDERFITTING		△ OVERFITTING	
	Causes	**Actions to Fix**	**Causes**	**Actions to Fix**
Decision Boundary Complexity	A linear boundary cannot separate classes that have curved or non-linear separation in the input space	Engineer polynomial or interaction features to give the linear boundary more expressive power	High-dimensional feature space allows the boundary to perfectly separate	Apply regularization (L1 or L2) to constrain the decision boundary and prevent it from contorting around noise

			training points, including noise	
Regularization Strength	Over-penalising the model shrinks coefficients too aggressively, preventing the boundary from fitting real patterns	Reduce regularization strength; use cross-validation to find the penalty level that balances bias and variance	Absent or very weak regularization lets coefficients grow large, fitting class-specific noise in the training set	Increase regularization strength; L1 regularization additionally forces irrelevant coefficients to exactly zero
Feature Quality	Key discriminative features are missing or poorly encoded, depriving the model of the signals it needs	Create interaction features; ensure categorical variables are meaningfully encoded (e.g., target encoding)	Redundant or highly correlated features effectively give the model multiple copies of the same noise	Use dimensionality reduction or feature selection to retain only features that add independent information
Class Imbalance	The model is dominated by the majority class and learns to largely ignore the minority class	Re-weight classes so minority examples carry more influence during training; oversample the minority class	After resampling or re-weighting, the model can overfit the artificially amplified minority class patterns	Use stratified cross-validation; calibrate predicted probabilities; evaluate with metrics beyond accuracy
Training Data Volume	Too few labelled examples per class to reliably estimate the decision boundary	Collect more labelled data; use active learning to prioritise the most informative examples to label	Small training set with many features means the model is estimated on insufficient evidence	Increase data volume; apply stronger regularization; use cross-validation rather than a single train-test split