Machine Learning Notes:

Concepts, Models, and Modern Architectures

Dipanka Tanu Sarmah

January 21, 2026

> I prepared these notes primarily for myself while learning and revisiting machine learning concepts. If others find them helpful in any way, that would be a welcome bonus. The goal here is clarity and connection of ideas, not completeness.

1 Optimization in Linear Models

Why Linear Models Exist

Linear models are the first principled response to the problem of prediction under uncertainty. They assume that the response variable can be expressed as a weighted superposition of input features corrupted by noise. This assumption allows closed-form solutions, statistical guarantees, and interpretability.

Their continued relevance lies not in expressive power but in transparency, stability, and theoretical tractability.

Strengths and Limitations

Pros

- Convex optimization with unique global optima
- Closed-form solutions and fast convergence
- Strong statistical interpretability

Cons

- Limited expressiveness
- Sensitive to multicollinearity
- Linear decision boundaries only

1.1 Ordinary Least Squares (OLS)

The OLS estimator seeks parameters β minimizing squared residual error:

$$L(\beta) = \|y - X\beta\|_2^2$$

The solution arises from stationarity of the loss gradient:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Geometric View OLS computes the orthogonal projection of y onto the column space of X. Residuals lie in the null space orthogonal to that span.

Statistical View Under Gaussian noise assumptions, OLS coincides with the maximum likelihood estimator.

Pseudocode: Ordinary Least Squares

```
Input: Design matrix X, response vector y
Compute A = X^T X
Compute b = X^T y
Solve A = b
Return
```

1.1.1 Regularization: L1 and L2

Regularization addresses instability when predictors are correlated or data is high-dimensional.

Ridge Regression

$$L(\beta) = \|y - X\beta\|_2^2 + \lambda\|\beta\|_2^2$$

Lasso Regression

$$L(\beta) = \|y - X\beta\|_2^2 + \lambda\|\beta\|_1$$

Ridge improves conditioning. Lasso induces sparsity through non-differentiable corners in its constraint geometry.

2 Logistic Regression

Why Logistic Regression Exists

Linear regression is unsuitable for classification because its outputs are unbounded and cannot be interpreted as probabilities. Logistic Regression was introduced to model binary outcomes by explicitly connecting linear predictors to probability theory.

Rather than predicting class labels directly, Logistic Regression models the probability that a data point belongs to a particular class, given its features.

Strengths and Limitations

Pros

- Probabilistic interpretation of predictions
- Convex optimization with a unique global optimum
- Strong statistical grounding
- Interpretable coefficients

Cons

- Linear decision boundary in feature space
- Limited capacity to model complex interactions
- Sensitive to outliers and class imbalance

Core Concepts Underlying Logistic Regression

From Linear Scores to Probabilities

Logistic Regression assumes that the log-odds of class membership are a linear function of the input features:

$$\log\left(\frac{P(y=1|x)}{1-P(y=1|x)}\right) = w^T x + b$$

This transformation ensures that predicted probabilities lie strictly between 0 and 1.

The Sigmoid Function

The sigmoid function maps real-valued inputs to probabilities:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

It is monotonic and differentiable, which makes optimization tractable.

Decision Boundary in Logistic Regression

The decision boundary of Logistic Regression is defined by:

$$w^T x + b = 0$$

This equation represents a hyperplane that separates feature space into two regions.

Points on one side are assigned higher probability for class 1, while points on the other side favor class 0.

The boundary is linear, regardless of how confident the predictions become away from it.

Interpreting the Parameters

For the decision function

$$f(x) = w^T x + b$$

- $x \in \mathbb{R}^d$ represents the feature vector of a data point.
- $w \in \mathbb{R}^d$ is the weight vector controlling the orientation of the boundary.
- $b \in \mathbb{R}$ is the bias term shifting the boundary.

Each component of w quantifies the contribution of a feature to the log-odds of the positive class.

Optimization via Maximum Likelihood

Logistic Regression is derived from a Bernoulli likelihood model:

$$P(y_i|x_i) = p_i^{y_i}(1 - p_i)^{1-y_i}$$

Maximizing the likelihood is equivalent to minimizing the negative log-likelihood:

$$L(w) = -\sum_i [y_i \log p_i + (1 - y_i) \log(1 - p_i)]$$

This loss function is known as the cross-entropy loss.

What Does "Error" Mean Here?

Classification Error

Classification error measures incorrect label assignment:

$$\mathbb{I}(y \neq \hat{y})$$

Logistic Regression does not optimize this quantity directly.

Log-Loss or Cross-Entropy Error

The optimized error is probabilistic:

$$\ell(y, p) = -[y \log p + (1 - y) \log(1 - p)]$$

This loss penalizes confident wrong predictions more heavily than uncertain ones.

Which Error Is Being Minimized?

Logistic Regression minimizes expected log-loss, not misclassification rate. This encourages well-calibrated probability estimates.

> **Why Probabilistic Error Matters**
>
> Two classifiers with identical accuracy can have very different probability estimates.
>
> Logistic Regression distinguishes between confident and uncertain predictions, which is essential in risk-sensitive domains such as medicine and finance.

Redundant Data in Logistic Regression

In Logistic Regression, all data points contribute to the likelihood. However, points far from the decision boundary have diminishing influence on parameter updates.

Once a point is classified with high confidence, its gradient contribution becomes negligible.

> **What Does Redundant Data Mean Here?**
>
> Redundant data points are samples whose predicted probabilities are already close to 0 or 1.
>
> These points contribute little to gradient updates and do not significantly influence the learned parameters.
>
> Learning is driven primarily by points near the decision boundary.

Regularization in Logistic Regression

To prevent overfitting and control coefficient magnitude, regularization terms are added:

$$L(w) = \text{LogLoss} + \lambda \|w\|_p$$

- L2 regularization stabilizes estimates and reduces variance.
- L1 regularization induces sparsity and performs feature selection.

Regularization directly controls the bias-variance tradeoff.

Practical Interpretation

Logistic Regression is best understood as a probabilistic linear classifier whose strength lies in interpretability and calibration rather than raw predictive power.

> Logistic Regression transforms classification into a statistical inference problem, where uncertainty is modeled explicitly rather than hidden behind hard decisions.

3 Tree-Based Models

Why Tree-Based Models Exist

Many real-world relationships are non-linear, hierarchical, and interaction-driven. Linear models impose global constraints that fail to capture such structure. Tree-based models were developed to address this limitation by recursively partitioning feature space into simpler, locally homogeneous regions.

Instead of fitting a single global function, decision trees learn a sequence of conditional rules that resemble human decision-making.

Strengths and Limitations

Pros

- Naturally capture non-linear relationships
- Handle feature interactions automatically
- Require minimal preprocessing and no feature scaling
- Highly interpretable at the individual tree level

Cons

- High variance for single trees
- Greedy optimization with no global optimality guarantee
- Unstable to small perturbations in data

Core Concepts Underlying Tree Models

Before discussing specific algorithms, it is essential to understand the foundational ideas shared by all tree-based learners.

Recursive Partitioning

Tree models divide feature space using a sequence of binary splits. Each split defines a condition of the form:

$$x_j \leq t$$

which partitions the data into two subsets. This process is applied recursively until a stopping criterion is met.

Decision Boundary Geometry

Decision trees produce decision boundaries that are axis-aligned and piecewise constant.

Unlike linear models, tree boundaries can change abruptly and adapt locally, but they cannot form smooth oblique surfaces without ensembling.

> **Decision Boundaries in Tree Models**
>
> A decision boundary in a tree model is composed of axis-aligned hyper-rectangles.
>
> Each internal node introduces a vertical or horizontal cut in feature space. The final classifier assigns a constant prediction within each resulting region.
>
> As tree depth increases, the boundary becomes more fragmented and flexible, but also more sensitive to noise.

Node Purity and Homogeneity

At each node, the goal is to produce child nodes that are more homogeneous than the parent. This is quantified using impurity measures.

Impurity Measures

Gini Impurity

$$G = 1 - \sum_k p_k^2$$

Gini impurity measures the probability of incorrect classification if a label is randomly assigned according to class proportions.

Entropy

$$H = -\sum_k p_k \log p_k$$

Entropy measures uncertainty in class labels. Information gain is defined as the reduction in entropy achieved by a split.

Which Impurity Is Optimized?

Trees do not minimize classification error directly. They optimize a local impurity reduction criterion, which is a differentiable surrogate for classification accuracy.

What Does "Error" Mean for Trees?

Misclassification Error

This refers to incorrect label assignment at a leaf:

$$\mathbb{I}(y \neq \hat{y})$$

It is not used for split optimization because it is insensitive to class probability changes.

Impurity-Based Error

Tree learning minimizes impurity at each split, not global classification error. This local optimization is greedy and myopic.

Generalization Error

Overly deep trees perfectly fit training data but generalize poorly. This is the classical overfitting problem in trees.

Stopping Criteria and Tree Complexity

Tree growth is controlled by stopping rules such as:

- Maximum depth
- Minimum samples per node
- Minimum impurity decrease

Deeper trees reduce bias but increase variance.

Pruning: Controlling Overfitting

Post-pruning removes branches that do not improve validation performance.

Cost-complexity pruning introduces a penalty on tree size:

$$R_\alpha(T) = R(T) + \alpha|T|$$

This balances training fit and model simplicity.

Redundant Data in Tree Models

Redundant data points are samples that fall deep within a leaf node and do not affect any split decisions.

Only data points near split thresholds influence tree structure.

This explains why trees can ignore large portions of the data while remaining sensitive to small changes near decision boundaries.

3.1 Ensemble Tree Methods

Single trees are unstable learners. Ensembles reduce this instability by combining many trees.

Bagging and Random Forests

Why Bagging Works

Bagging reduces variance by training trees on bootstrap samples and averaging predictions.

Random Forests

Random Forests introduce additional randomness by selecting a subset of features at each split.

Key Effects

- Decorrelates trees
- Reduces variance without increasing bias significantly
- Improves robustness

Boosting and Gradient Boosting Trees

Why Boosting Exists

Bagging reduces variance but does not reduce bias. Boosting was developed to address systematic underfitting.

Core Idea

Trees are trained sequentially. Each tree focuses on correcting the errors made by the previous ensemble.

Gradient Boosting Interpretation

Boosting can be viewed as gradient descent in function space, where each tree fits the negative gradient of the loss function.

Bias-Variance Behavior

- Boosting reduces bias aggressively
- Overfitting can occur without regularization

Comparative Summary of Tree Models

- Single trees are interpretable but unstable
- Random forests emphasize variance reduction
- Gradient boosting emphasizes bias reduction
- Ensembles sacrifice interpretability for performance

Tree-based models transform learning into a sequence of local decisions. Their power comes from hierarchical partitioning, while their limitations arise from greedy optimization.

What Is a Kernel Function?

A *kernel* is a function

$$K(x, x') = \langle \phi(x), \phi(x') \rangle$$

that computes an inner product in a (possibly high-dimensional) feature space without explicitly constructing the mapping ϕ.

This allows algorithms to operate as if the data were embedded in a richer representation space, while all computations remain in the original input space.

Key Properties

- The kernel must be symmetric: $K(x, x') = K(x', x)$.
- The kernel must be positive semi-definite, meaning that the kernel matrix is non-negative for all finite datasets.

Why Kernels Matter

- Linear algorithms become non-linear in the original input space.
- Decision boundaries can curve, bend, and adapt to complex geometry.
- Computation scales with sample size, not feature space dimension.

Common Examples

$$\begin{aligned} \text{Linear:} \quad & K(x, x') = x^T x' \\ \text{Polynomial:} \quad & K(x, x') = (x^T x' + c)^d \\ \text{RBF:} \quad & K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right) \end{aligned}$$

The kernel trick replaces explicit feature engineering with implicit geometry.

4 Support Vector Machines

Why Support Vector Machines Exist

Most classifiers attempt to minimize training error. Support Vector Machines take a fundamentally different approach. They seek to maximize the geometric margin between classes, based on the principle that classifiers with larger margins generalize better to unseen data.

Rather than focusing on all data points equally, SVMs identify a small subset of critical samples that define the decision boundary. These points are known as *support vectors*.

What Is a Decision Boundary?

A decision boundary is the geometric surface that separates different predicted classes in feature space.

For binary classification, it is the set of all points x for which the classifier is indifferent between classes. Formally, it is defined by:

$$f(x) = 0$$

Points on one side of the boundary are assigned to class 0, while points on the other side are assigned to class 1.

The shape of the decision boundary reflects the assumptions of the model:

- Linear models produce flat hyperplanes.
- Tree-based models produce axis-aligned partitions.
- Kernel methods produce curved boundaries.
- Neural networks produce highly flexible surfaces.

Core Concepts Underlying All SVMs

Before distinguishing between different SVM variants, it is essential to understand the concepts shared by all of them.

Maximum Margin Principle

For a linear decision function

$$f(x) = w^T x + b$$

the margin between two classes is defined as

$$\text{Margin} = \frac{2}{\|w\|}$$

Interpreting the Linear Decision Function

The linear decision function

$$f(x) = w^T x + b$$

defines a hyperplane in feature space.

- $x \in \mathbb{R}^d$ is the input feature vector representing a data point.
- $w \in \mathbb{R}^d$ is the weight vector that determines the orientation of the decision boundary.
- $b \in \mathbb{R}$ is the bias or intercept term that shifts the boundary away from the origin.

The sign of $f(x)$ determines the predicted class, while the magnitude of $f(x)$ reflects confidence relative to the boundary.

Maximizing the margin is equivalent to minimizing $\|w\|^2$, which leads to a well-posed optimization problem with strong generalization guarantees.

Support Vectors

Only data points that lie on or within the margin constraints influence the final classifier. These points are called support vectors.

Removing non-support vectors does not change the learned decision boundary. This property makes SVMs robust to redundant data.

Primal and Dual Formulations

The primal optimization problem focuses on parameters w and b. The dual formulation rewrites the problem entirely in terms of pairwise inner products between samples.

The dual form is what enables kernel methods, allowing SVMs to operate in implicit high-dimensional feature spaces.

Hard-Margin SVM

When It Is Needed

Hard-margin SVMs are used when data is perfectly linearly separable with no overlap between classes.

Optimization Problem

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1 \quad \forall i$$

Characteristics

Pros

- Maximum geometric separation
- Strong theoretical guarantees

Cons

- Extremely sensitive to noise
- Rarely applicable to real data

Soft-Margin SVM

Why Soft Margins Are Necessary

Real-world data is noisy and often overlapping. Soft-margin SVMs allow controlled violations of margin constraints using slack variables.

Optimization Problem

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_i \xi_i \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0$$

The Role of the C Parameter

- Large C penalizes misclassification strongly and favors low bias.
- Small C allows margin violations and favors smoother boundaries.

Interpretation

The C parameter controls the bias-variance tradeoff explicitly. Soft-margin SVMs are the most commonly used form in practice.

Kernel SVMs

Why Kernels Are Introduced

Linear separation in input space is often impossible. Kernel SVMs overcome this limitation by mapping data into a higher-dimensional feature space where linear separation becomes feasible.

This mapping is performed implicitly using a kernel function.

Common Kernel Choices

- **Linear Kernel**: Suitable when data is approximately linearly separable.
- **Polynomial Kernel**: Captures feature interactions of fixed degree.
- **Radial Basis Function (RBF)**: Models local similarity and complex non-linear boundaries.

RBF Kernel Hyperparameters

- γ controls the radius of influence of each support vector.
- Large γ leads to complex boundaries and high variance.
- Small γ leads to smoother boundaries and higher bias.

> Support Vector Machines convert learning into a geometric optimization problem. Their strength lies not in flexibility alone, but in the tight coupling between margin, regularization, and generalization theory.

One-Class SVM

Why One-Class SVM Exists

In anomaly detection, labeled negative examples are often unavailable. One-Class SVMs learn the support of a single distribution and identify deviations from it.

Core Idea

The algorithm finds a boundary that encloses most of the data points while allowing a small fraction to lie outside.

Applications

- Outlier detection
- Novelty detection
- Quality control

Support Vector Regression (SVR)

Why Regression with SVM

SVR extends the margin concept to regression by introducing an ϵ-insensitive loss.

Objective

Errors smaller than ϵ are ignored. Only deviations beyond ϵ contribute to the loss.

Key Parameters

- ϵ controls the width of the insensitive tube.
- C controls penalty for points outside the tube.

Practical Summary of SVM Variants

- Hard-margin SVM is theoretically elegant but rarely practical.
- Soft-margin SVM is the default choice for classification.
- Kernel SVMs handle non-linear data efficiently.
- One-Class SVM addresses unsupervised anomaly detection.
- SVR applies margin-based reasoning to regression.

What Does "Error" Mean in This Context?

The word *error* is used differently across machine learning algorithms. It is important to distinguish these meanings clearly.

Classification Error

Classification error refers to incorrect label assignment:

$$\mathbb{I}(y \neq \hat{y})$$

This is a discrete, non-differentiable quantity. SVMs do not directly minimize classification error.

Margin Violation Error

In Support Vector Machines, the relevant notion of error is *margin violation*.

A data point incurs error if it violates the constraint:

$$y_i(w^T x_i + b) \geq 1$$

Violations are quantified using slack variables ξ_i.

Loss-Based Error

SVMs minimize a surrogate loss known as the *hinge loss*:

$$\ell(y, f(x)) = \max(0, 1 - yf(x))$$

This loss penalizes both misclassification and points that lie too close to the boundary.

Which Error Is Being Optimized Here?

Support Vector Machines optimize margin-based loss, not raw misclassification rate.

This distinction is crucial. By minimizing a convex surrogate loss, SVMs achieve better generalization even when some classification errors are tolerated.

What Does Redundant Data Mean?

Redundant data points are samples that do not influence the position of the decision boundary.

In Support Vector Machines, only points that lie on or within the margin constraints affect the learned model. These points are called *support vectors*.

All other points can be removed from the training set without changing the classifier.

Why This Matters

- The model complexity depends on the number of support vectors, not the dataset size.
- SVMs are robust to repeated or duplicated observations.
- The decision boundary is determined by the hardest-to-classify samples.

> This property reflects the principle that learning is governed by constraints, not averages.

What Does "Error" Mean in This Context?

The word *error* is used differently across machine learning algorithms. It is important to distinguish these meanings clearly.

Classification Error

Classification error refers to incorrect label assignment:

$$\mathbb{I}(y \neq \hat{y})$$

This is a discrete, non-differentiable quantity. SVMs do not directly minimize classification error.

Margin Violation Error

In Support Vector Machines, the relevant notion of error is *margin violation*.

A data point incurs error if it violates the constraint:

$$y_i(w^T x_i + b) \geq 1$$

Violations are quantified using slack variables ξ_i.

Loss-Based Error

SVMs minimize a surrogate loss known as the *hinge loss*:

$$\ell(y, f(x)) = \max(0, 1 - yf(x))$$

This loss penalizes both misclassification and points that lie too close to the boundary.

Which Error Is Being Optimized Here?

Support Vector Machines optimize margin-based loss, not raw misclassification rate.

This distinction is crucial. By minimizing a convex surrogate loss, SVMs achieve better generalization even when some classification errors are tolerated.

5 Neural Networks

Why Neural Networks Exist

Complex phenomena require hierarchical feature extraction. Neural networks learn representations rather than fixed transformations.

Strengths and Limitations

Pros

- Universal function approximation
- Automatic feature learning

Cons

- Non-convex optimization
- High data and compute demand

5.1 Backpropagation

Gradients are computed efficiently via the chain rule over computational graphs.

Pseudocode: Backpropagation

```
Initialize weights
For each epoch:
    Forward pass
    Compute loss
    Backward pass
    Update parameters
```

> Neural networks perform gradient flow on high-dimensional energy landscapes shaped by data, architecture, and regularization.

6 Mathematical Appendix

6.1 Convexity

A function f is convex if:

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$$

Convex losses guarantee global optima.

6.2 Bias-Variance Decomposition

$$\mathbb{E}[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

7 Feature Selection

Why Feature Selection Matters

In supervised learning, a *feature* is a measurable attribute used to represent data. Modern datasets often contain many features, some of which may be irrelevant, redundant, or noisy.

As the number of features increases, several challenges arise:

- Models become more complex and harder to interpret
- Estimation variance increases
- Optimization becomes less stable
- Generalization performance can degrade

Feature selection aims to address these challenges by identifying a subset of informative variables that are sufficient for prediction.

The Curse of Dimensionality

As dimensionality grows, data points become sparse in feature space. Distance measures lose meaning, and models require exponentially more data to learn reliable decision boundaries.

This phenomenon is known as the curse of dimensionality and motivates reducing the effective feature space.

Feature Selection and the Bias-Variance Tradeoff

Removing features simplifies the model and reduces variance, but excessive removal can increase bias.

Feature selection is therefore a balancing act between underfitting and overfitting, closely tied to the bias-variance tradeoff.

What Does "Important Feature" Mean?

Feature importance is not an absolute concept. A feature is important only relative to:

- A specific model class
- A specific prediction task
- A specific loss function

A feature that is important for a linear model may be irrelevant for a tree-based model, and vice versa.

Types of Feature Selection Methods

Feature selection methods are commonly categorized into three classes.

Filter Methods

Filter methods evaluate features independently of any learning algorithm.

Examples include correlation tests, mutual information, and statistical hypothesis testing.

Advantages

- Computationally efficient
- Model-agnostic

Limitations

- Ignore feature interactions
- Do not account for model behavior

Wrapper Methods

Wrapper methods evaluate subsets of features by training and testing a predictive model.

Examples include forward selection, backward elimination, and Recursive Feature Elimination.

Advantages

- Model-aware selection
- Can capture feature interactions

Limitations

- Computationally expensive
- Risk of overfitting

Embedded Methods

Embedded methods perform feature selection during model training.

Examples include LASSO, elastic net, and tree-based feature importance.

Advantages

- Efficient
- Integrated with optimization

Limitations

- Model-specific
- Less flexible

Why Wrapper Methods Are Special

Wrapper methods explicitly evaluate how feature subsets affect predictive performance.

This makes them powerful but also computationally intensive.

Recursive Feature Elimination is a structured and widely used wrapper method, making it a natural next topic.

> Feature selection is not about finding the smallest set of features. It is about finding a representation that supports stable and generalizable learning.

8 Recursive Feature Elimination

Why Feature Selection Is Necessary

In high-dimensional datasets, not all features contribute meaningfully to prediction. Irrelevant or redundant features increase variance, reduce interpretability, and can degrade generalization.

Feature selection aims to identify a compact subset of informative variables while preserving predictive performance.

Recursive Feature Elimination (RFE) is a wrapper-based method designed to perform feature selection in conjunction with a predictive model.

What Makes RFE Different

Unlike filter methods that rank features independently of a model, RFE evaluates features based on their contribution to a trained classifier or regressor.

Unlike embedded methods that select features during training, RFE explicitly removes features in an iterative and controlled manner.

Core Idea of Recursive Feature Elimination

RFE operates by repeatedly training a model, ranking features by importance, removing the least important features, and retraining the model on the reduced feature set.

This recursive process continues until a desired number of features remains.

Algorithmic Workflow

At a high level, RFE follows four steps:

1. Train a model using the current set of features.
2. Compute feature importance scores from the trained model.
3. Remove the least important feature or features.
4. Repeat until a stopping criterion is met.

What Does "Importance" Mean in RFE?

The definition of feature importance depends entirely on the underlying model.

- In Logistic Regression or Linear SVM, importance is given by the magnitude of the coefficients.
- In Tree-based models, importance is measured by impurity reduction.
- In kernel SVMs, importance is indirect and often approximated using linear surrogates.

RFE itself does not define importance. It delegates this responsibility to the base estimator.

RFE Is Model-Dependent

Recursive Feature Elimination does not discover universally important features.

It discovers features that are important for a specific model and objective.

Different base learners can produce different selected feature subsets on the same data.

Stopping Criteria

RFE can be stopped using one or more of the following criteria:

- A fixed number of remaining features
- Cross-validated performance degradation
- Exhaustive elimination down to a single feature

Decision Boundary Perspective

Feature elimination alters the geometry of the decision boundary.
Removing irrelevant features can simplify the boundary and improve generalization.
Removing informative features can distort the boundary and increase bias.

Feature Selection and Decision Geometry

Each feature corresponds to a dimension in feature space.

RFE progressively reduces the dimensionality of this space, forcing the classifier to construct boundaries using fewer directions.

Successful feature elimination preserves the geometry needed for separation while removing noisy axes.

Pseudocode: Recursive Feature Elimination

"'text Input: Dataset X, labels y, base model M Initialize feature set F = all features
While |F| > desired size: Train M on features F Compute feature importance scores Remove least important feature(s) from F
Return selected feature set F

Strengths and Limitations

Pros

- Model-aware feature selection
- Improves interpretability

- Can improve generalization

Cons

- Computationally expensive
- Sensitive to noise and correlations
- Dependent on base model choice

Relationship to Other Feature Selection Methods

- Filter methods rank features independently of models.
- Embedded methods perform selection during optimization.
- RFE is a wrapper method that iteratively retrains models.

When Should RFE Be Used?

RFE is particularly useful when:

- The number of features is large relative to samples
- Interpretability is important
- Model-based feature relevance is desired

> Recursive Feature Elimination reframes feature selection as an iterative hypothesis test between models of decreasing dimensionality.

9 Cross-Validation and RFECV

Why Cross-Validation Is Necessary

Evaluating a model on the same data used for training produces optimistic performance estimates. This issue becomes more severe when models are flexible or when feature selection is performed.

Cross-validation provides a principled framework for estimating generalization performance by repeatedly training models on subsets of data and evaluating them on held-out samples.

Core Idea of Cross-Validation

In K-fold cross-validation, the dataset is partitioned into K disjoint subsets called folds.

- The model is trained on $K - 1$ folds.
- Performance is evaluated on the remaining fold.
- The process is repeated so that each fold serves as validation once.

The final performance estimate is obtained by averaging across all folds.

What Cross-Validation Actually Estimates

Cross-validation approximates the expected test error under the assumption that samples are independent and identically distributed.

It reduces variance caused by a single train-test split and enables principled model comparison and hyperparameter tuning.

Why Feature Selection Requires Cross-Validation

When feature selection is performed using the entire dataset, information from the validation set leaks into the training process.

This phenomenon is known as data leakage and leads to inflated performance estimates.

Feature selection must therefore be nested inside the training process of each cross-validation split.

Data Leakage in Feature Selection

If features are selected before cross-validation, the validation data indirectly influences the model.

This violates the independence assumption required for honest performance evaluation.

Cross-validation prevents leakage by isolating feature selection within each training fold.

Recursive Feature Elimination with Cross-Validation (RFECV)

Recursive Feature Elimination with Cross-Validation extends RFE by automatically determining the optimal number of features using validation performance.

Rather than fixing the number of features in advance, RFECV evaluates model performance across different feature subset sizes under cross-validation.

Algorithmic Workflow of RFECV

RFECV proceeds through the following steps:

1. Choose a base learning algorithm.
2. Apply Recursive Feature Elimination to rank and remove features.
3. At each feature subset size, evaluate model performance using cross-validation.
4. Select the feature subset size that maximizes the chosen validation metric.

What Does RFECV Optimize?

RFECV optimizes a user-defined performance metric, such as:

- Accuracy
- ROC-AUC
- F1 score
- Mean squared error

The selected feature subset is the one that achieves the best average cross-validated performance.

Bias-Variance Perspective

As the number of selected features decreases:

- Model variance decreases due to reduced complexity.
- Model bias may increase if informative features are removed.

RFECV explicitly searches for the feature subset where this tradeoff yields optimal validation performance.

RFECV as a Bias-Variance Controller

RFECV does not aim to find the smallest feature set.

It aims to find the feature set that best balances bias and variance under cross-validation.

Pseudocode: RFECV

```
Input: Dataset X, labels y, base model M, number of folds K

For each feature subset size s:
    For each cross-validation fold:
        Perform RFE on training fold
        Train model M using selected features
        Evaluate performance on validation fold
    Compute mean validation score for s

Select feature subset size with best mean score
Return selected feature set and trained model
```

Computational Considerations

RFECV is computationally expensive because it repeatedly trains models across feature subsets and cross-validation folds.

Its use is best justified when interpretability is important or when feature dimensionality is moderate.

Schematic Illustration of RFECV

RFECV Workflow

Dataset → Split into K folds

Training folds → RFE → Train model

Validation fold → Evaluate performance

Repeat for all folds and feature subset sizes

Select feature set with best average validation score

Figure 1: Schematic representation of Recursive Feature Elimination with Cross-Validation. Feature selection is nested within cross-validation to prevent data leakage and to identify the feature subset that optimizes generalization performance.

When Should RFECV Be Used?

RFECV is particularly appropriate when:

- The number of features is large relative to sample size
- Interpretability is important
- Feature redundancy is suspected

Recursive Feature Elimination with Cross-Validation reframes feature selection as a model selection problem grounded in statistical validation.

10 From Feature Selection to Representation Learning

Limitations of Feature-Centric Learning

All models discussed so far assume that data is represented by a fixed set of features. Learning consists of assigning weights, splits, or margins over these predefined dimensions.

Feature selection methods such as RFE and RFECV attempt to identify the most informative subset of these features. However, the features themselves remain static.

The Shift Toward Learned Representations

Neural networks introduced the idea of learning intermediate representations through layered transformations. Instead of selecting features explicitly, networks learn combinations of features that are useful for prediction.

However, classical neural networks still rely on fixed connectivity patterns and shared transformations applied uniformly across inputs.

Why Attention Changes the Picture

Attention mechanisms allow models to dynamically weight different parts of the input depending on context.

Rather than selecting a subset of features once, attention assigns importance scores to inputs at each prediction step.

This marks a shift from feature selection to context-dependent feature weighting.

> Feature selection chooses what to keep. Attention learns what to focus on.

11 Transformers

Why Transformers Were Introduced

Many real-world data types are sequential or structured, such as text, time series, biological sequences, and graphs. Earlier models such as recurrent neural networks process sequences step by step, which makes learning long-range dependencies difficult and computationally inefficient.

Transformers were introduced to overcome these limitations by removing recurrence entirely and replacing it with attention mechanisms that operate on the full input simultaneously.

The Core Idea Behind Transformers

The central idea of a Transformer is simple but powerful.

Instead of processing inputs sequentially, the model looks at all elements of the input at once and learns which elements are important for each other.

This is achieved through a mechanism called *self-attention*.

From Features to Tokens

In Transformer models, the basic unit of input is a *token*.

A token can represent:

- A word or subword in text
- A position in a sequence
- An element in a structured set

Each token is mapped to a continuous vector representation called an *embedding*. Learning then operates on these embeddings rather than on handcrafted features.

> Transformers do not select features explicitly. They learn how to represent and weight inputs jointly through attention.

Self-Attention: The Fundamental Operation

Self-attention allows each token to interact with every other token in the input.

Given an input matrix $X \in \mathbb{R}^{n \times d}$, where n is the number of tokens and d is the embedding dimension, three linear transformations are applied:

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

Here:

- Q represents queries
- K represents keys
- V represents values

The attention output is computed as:

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

Interpreting Queries, Keys, and Values

Each token asks a question about the rest of the sequence through its query vector.

Keys determine how relevant other tokens are to that question.

Values carry the information that is aggregated based on relevance.

The result is a weighted combination of values that depends on context.

> Self-attention assigns importance dynamically. The same token can be important in one context and irrelevant in another.

Why Scaling Is Necessary

The dot product QK^T grows with dimensionality, which can lead to unstable gradients.

Dividing by $\sqrt{d_k}$ stabilizes optimization and ensures smooth gradient flow during training.

Multi-Head Attention

Instead of performing a single attention operation, Transformers use multiple attention heads.

Each head learns a different type of interaction:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(h_1, \ldots, h_H)W_O$$

This allows the model to capture diverse relationships such as syntax, semantics, or long-range dependencies.

Position Information and Order

Because Transformers process tokens in parallel, they require explicit information about order.

This is achieved through positional encodings that are added to token embeddings.

These encodings allow the model to distinguish between sequences such as:

$$(x_1, x_2, x_3) \neq (x_3, x_2, x_1)$$

The Transformer Block

A Transformer is built by stacking identical blocks, each consisting of:

- Multi-head self-attention
- A position-wise feedforward network
- Residual connections
- Layer normalization

Residual connections preserve gradient flow, while normalization stabilizes training.

Encoder and Decoder Roles

In the original formulation:

- The encoder builds contextual representations of the input.
- The decoder generates outputs while attending to encoder representations.

Some modern architectures use only encoders or only decoders, depending on the task.

What Does "Error" Mean in Transformers?

Transformers are trained by minimizing a task-specific loss, commonly cross-entropy for classification or language modeling.

The error reflects mismatch between predicted probability distributions and observed targets.

Unlike classical models, the error propagates through attention weights and representation layers, not just through a decision boundary.

Strengths and Limitations of Transformers

Strengths

- Learn representations automatically
- Capture long-range dependencies
- Highly parallelizable
- Scalable across domains

Limitations

- Computationally expensive
- Large data requirements
- Limited interpretability without analysis tools

Schematic View of a Transformer

Why Transformers Represent a Paradigm Shift

Earlier models relied on feature engineering, selection, or fixed interactions.

Transformers learn representations, interactions, and importance jointly.

This unifies feature selection, weighting, and transformation into a single differentiable framework.

> Transformers do not ask which features matter. They learn how meaning emerges from interactions.

Looking Ahead

With this foundation, we can now examine how large-scale Transformer models are trained on language data and how they produce coherent, context-aware responses in real applications.

This will serve as a concrete case study for understanding Transformer behavior in practice.

12 Transformers for Language Modeling

What Is Language Modeling?

Language modeling is the task of assigning probabilities to sequences of tokens.

Given a sequence $(x_1, x_2, \ldots, x_n)$, a language model estimates:

$$P(x_1, x_2, \ldots, x_n)$$

Using the chain rule of probability, this is decomposed as:

$$P(x_1, \ldots, x_n) = \prod_{t=1}^{n} P(x_t \mid x_1, \ldots, x_{t-1})$$

The model learns to predict the next token given all previous tokens.

Why Transformers Are Well-Suited for Language

Language exhibits long-range dependencies. The meaning of a word often depends on distant context.

Transformers handle this naturally because self-attention allows every token to attend to all previous tokens directly, without compression through recurrent states.

Autoregressive Transformer Models

In language modeling, Transformers are typically used in an autoregressive manner.

At training time:

- The full sequence is provided.
- Attention is masked so each token only attends to earlier tokens.
- The model predicts the next token at every position.

At inference time:

- Tokens are generated one at a time.
- Each generated token is appended to the context.
- The process repeats until a stopping condition is met.

Training Objective

The model is trained by minimizing cross-entropy loss:

$$\mathcal{L} = -\sum_{t} \log P(x_t \mid x_1, \ldots, x_{t-1})$$

This objective encourages the model to assign high probability to observed continuations.

> Language models do not learn grammar rules explicitly. They learn probability distributions over continuations.

13 How Conversational Language Models Work

From Language Models to Dialogue

A conversational language model is still a language model. The difference lies in how it is trained and prompted.

Conversation is represented as a sequence of tokens that includes:

- System instructions
- User inputs
- Assistant responses

The model learns to predict the next token conditioned on the entire dialogue history.

Role of Pretraining

During pretraining, the model is exposed to large amounts of text and learns general language structure, world knowledge, and patterns of reasoning.

This phase teaches syntax, semantics, and factual associations.

Instruction Following and Alignment

To make the model useful in conversation, additional training stages are applied:

- Supervised fine-tuning on instruction-response pairs
- Preference-based optimization using human feedback

These steps encourage helpful, safe, and coherent responses.

Inference in Conversation

At inference time:

- The prompt is tokenized.
- The Transformer computes next-token probabilities.
- A decoding strategy selects the next token.

Common decoding strategies include greedy decoding, sampling, and nucleus sampling.

> Conversational models do not retrieve answers. They generate responses token by token based on learned probability distributions.

14 Why They Can Reason, Fail, and Hallucinate

What Does "Reasoning" Mean Here?

Reasoning in language models emerges from the ability to combine patterns across tokens and contexts.

The model does not perform symbolic logic. Instead, it approximates reasoning through learned statistical regularities.

Why Reasoning Appears

Reasoning-like behavior arises because:

- Attention allows integration of distant context
- Large models learn compositional patterns
- Training data contains examples of reasoning chains

This enables multi-step inference without explicit rule execution.

Why Failures Occur

Failures occur when:

- The prompt is ambiguous
- Required information is missing
- The model extrapolates beyond its training distribution

The model always produces a fluent continuation, even when uncertain.

Hallucination Explained

Hallucination refers to confident but incorrect generation.

This occurs because the training objective rewards plausibility, not truth.

If a continuation is statistically likely given the prompt, the model may generate it even if it is factually wrong.

> Language models optimize likelihood, not correctness. Fluency does not imply truth.

Why Hallucinations Are Hard to Eliminate

Hallucinations are not simple bugs. They are a consequence of:

- Open-ended generation
- Incomplete grounding
- Probabilistic objectives

Mitigation requires external verification, retrieval, or structured constraints.

15 Connecting Attention to Feature Importance

Attention as Dynamic Feature Weighting

Earlier models rely on fixed feature importance. Transformers compute importance dynamically through attention.

Each attention weight represents how strongly one token influences another in a given context.

Comparison with Classical Feature Selection

- RFE selects features globally and permanently.
- Tree importance reflects average split utility.
- Attention assigns relevance conditionally and transiently.

A feature can be important in one context and irrelevant in another.

Attention Weights as Explanations

Attention weights can sometimes provide insight into model behavior, but they are not definitive explanations.

They indicate information flow, not causal influence.

Limitations of Attention Interpretability

High attention does not guarantee importance, and low attention does not imply irrelevance.

Interpretation must be combined with probing, ablation, and sensitivity analysis.

> Attention tells us where the model looks, not why it believes.

Unifying Perspective

Transformers unify feature representation, selection, and weighting into a single differentiable mechanism.

This marks a shift from static modeling assumptions to context-sensitive computation.

> Classical models choose features. Transformers learn interactions.

16 Transformers for Language Modeling

What Is Language Modeling?

Language modeling is the problem of learning a probability distribution over sequences of discrete symbols called tokens.

Given a sequence of tokens $(x_1, x_2, \ldots, x_n)$, a language model estimates the joint probability:

$$P(x_1, x_2, \ldots, x_n)$$

Using the chain rule of probability, this joint distribution is decomposed into conditional probabilities:

$$P(x_1, \ldots, x_n) = \prod_{t=1}^{n} P(x_t \mid x_1, \ldots, x_{t-1})$$

This formulation converts sequence modeling into a repeated next-token prediction task.

Why Next-Token Prediction Is Sufficient

Although the objective appears simple, predicting the next token requires understanding:

- Syntax and grammar
- Semantics and meaning
- Long-range dependencies
- World knowledge and discourse structure

Any error in these components reduces the probability of the correct next token.

> Language modeling compresses linguistic competence into the ability to predict what comes next.

Autoregressive Transformers

In Transformer-based language models, prediction is autoregressive.

During training:

- The entire sequence is available.
- A causal attention mask prevents access to future tokens.
- The model predicts every token using its preceding context.

During inference:

- Tokens are generated one at a time.
- Each generated token becomes part of the context.
- Generation continues until a termination condition is met.

Training Objective

The model parameters are optimized by minimizing the negative log-likelihood of observed sequences:

$$\mathcal{L} = -\sum_{t} \log P(x_t \mid x_1, \ldots, x_{t-1})$$

This is equivalent to minimizing cross-entropy loss between predicted and empirical token distributions.

Why Transformers Excel at Language Modeling

Transformers outperform earlier architectures because:

- Self-attention captures long-range dependencies directly.
- Parallel computation improves efficiency.
- Depth allows hierarchical abstraction.

> Transformers do not memorize sentences. They learn probability landscapes over language.

17 How Conversational Language Models Work

Conversation as a Sequence Modeling Problem

A conversation is represented as a single long sequence of tokens, including:

- Instructions defining assistant behavior
- User queries
- Assistant responses

There is no architectural distinction between user and assistant text. The distinction exists only through token patterns learned during training.

Pretraining Phase

In pretraining, the model is trained on massive corpora of text using the language modeling objective.
This phase teaches:

- Grammar and syntax
- Semantic relationships
- Factual associations
- Common reasoning patterns

Pretraining builds a general-purpose language representation.

Instruction and Alignment Training

To make the model conversational, additional training stages are applied.

- Supervised fine-tuning teaches how to respond to instructions.
- Preference-based optimization encourages helpful and safe responses.

These stages reshape the probability distribution learned during pretraining.

> Alignment does not add knowledge. It reshapes how existing knowledge is expressed.

Inference and Decoding

At inference time, the model outputs a probability distribution over the vocabulary.
A decoding algorithm selects the next token based on this distribution.
Common strategies include:

- Greedy decoding
- Temperature-controlled sampling
- Nucleus sampling

Different decoding strategies trade determinism for diversity.

Why Responses Appear Intentional

Conversational coherence arises because:

- The model conditions on the entire dialogue history.
- Attention integrates context dynamically.
- Training data contains conversational patterns.

There is no internal intent or goal. Coherence emerges from probabilistic conditioning.

18 Why They Can Reason, Fail, and Hallucinate

What Is Reasoning in Language Models?

Reasoning refers to the ability to produce multi-step, logically consistent outputs.
In language models, reasoning is not symbolic manipulation. It is the emergence of structured patterns through learned representations.

Why Reasoning Emerges

Reasoning-like behavior arises because:

- Attention enables integration across distant tokens.
- Depth enables composition of simpler patterns.
- Training data contains examples of reasoning chains.

The model interpolates between these examples.

> Language models do not execute logic. They approximate it statistically.

Why Errors Occur

Errors arise when:

- Context is insufficient or ambiguous
- The model extrapolates beyond its training distribution
- Multiple plausible continuations exist

The model must always choose a continuation, even when uncertain.

Hallucination Defined

Hallucination is the generation of fluent but incorrect information.
This occurs because the objective function rewards likelihood, not factual correctness.
If an incorrect continuation is statistically plausible, it may be generated confidently.

Why Hallucinations Are Structural

Hallucinations are not simple bugs.
They emerge from:

- Open-ended generation
- Lack of grounding
- Probabilistic objectives

Mitigation requires external signals such as retrieval, verification, or constraints.

> Fluency is optimized directly. Truth is not.

19 Connecting Attention to Feature Importance

From Static Features to Dynamic Relevance

Classical models assign fixed importance to features.
Transformers compute relevance dynamically through attention weights that depend on context.

Attention as Conditional Weighting

In self-attention, each token assigns weights to other tokens:

$$\alpha_{ij} = \text{softmax}\left(\frac{q_i k_j}{\sqrt{d_k}}\right)$$

These weights determine how information flows through the model.

Comparison with Classical Feature Selection

- RFE removes features permanently.
- Tree importance averages over many splits.
- Attention assigns relevance conditionally at each layer.

Attention operates at inference time, not only during training.

Can Attention Be Interpreted?

Attention weights provide partial insight into model behavior.
However:

- High attention does not imply causal importance.
- Low attention does not imply irrelevance.

Interpretation requires additional analysis such as ablation or probing.

> Attention shows information flow, not belief.

Unifying Perspective

Transformers unify:

- Feature representation
- Feature weighting
- Feature interaction

into a single differentiable mechanism.

> Classical models choose features. Transformers learn interactions.

20 ChatGPT as a Case Study of Transformer-Based Language Models

Why Use ChatGPT as an Example

Abstract descriptions of Transformers can obscure how these models actually behave in practice. ChatGPT provides a concrete and familiar example of a large-scale Transformer trained for conversational language modeling.

By analyzing ChatGPT mechanistically, we can connect theory to observable behavior without introducing new architectural concepts.

What ChatGPT Is and What It Is Not

ChatGPT is a Transformer-based autoregressive language model that generates text by predicting the next token given a context.

It is important to clarify what ChatGPT does not do:

- It does not search the internet during generation.
- It does not retrieve documents from a database by default.
- It does not store memories of past conversations.
- It does not reason symbolically.

All responses are generated by computing probability distributions over tokens conditioned on the prompt.

Step-by-Step: What Happens When You Ask a Question

Consider a user prompt:

> Explain why logistic regression uses cross-entropy loss.

The following steps occur internally.

Step 1: Tokenization

The input text is converted into a sequence of tokens. These tokens may correspond to whole words, subwords, or symbols.

Tokenization defines the atomic units on which the Transformer operates.

Step 2: Embedding and Position Encoding

Each token is mapped to a vector embedding.

Positional encodings are added so the model can distinguish between different token orders.

The result is a matrix representation of the input sequence.

Step 3: Contextualization via Self-Attention

Through stacked Transformer layers, each token embedding is updated by attending to all other tokens in the prompt.

Attention weights determine which parts of the prompt influence each token representation.

Key concepts such as *logistic regression*, *cross-entropy*, and *loss* reinforce one another through attention.

> The model does not locate an answer. It constructs a context-aware representation of the question.

Step 4: Next-Token Probability Computation

After contextualization, the model computes a probability distribution over the vocabulary for the next token.

For example, the model assigns probabilities to possible continuations such as:

- "Because"
- "Logistic"
- "Cross-entropy"

Step 5: Decoding

A decoding strategy selects one token from the probability distribution.

This token is appended to the context, and the process repeats until a complete response is generated.

Why the Answer Appears Structured

The generated explanation appears coherent because:

- Training data contains many similar explanations.
- Attention integrates definitions, motivations, and examples.
- The model has learned discourse patterns of teaching text.

There is no internal representation of correctness. Only likelihood governs generation.

Why ChatGPT Can Explain Concepts It Was Not Explicitly Taught

ChatGPT can explain new questions because it has learned compositional structure.
It can combine:

- Definitions of logistic regression
- Properties of likelihood functions
- Explanations of loss functions

into a coherent answer, even if that exact question never appeared in training.

> Generalization in language models emerges from composition, not memorization.

Where Reasoning Comes From in Practice

When ChatGPT produces multi-step explanations, it is not executing a reasoning algorithm.
Instead:

- Earlier tokens shape later tokens.
- Attention maintains consistency across steps.
- The model imitates reasoning patterns found in training data.

This creates the appearance of logical reasoning.

Why ChatGPT Sometimes Fails

Failures occur when:

- The prompt underspecifies the task.
- The model must extrapolate beyond training examples.
- Multiple plausible explanations exist.

In such cases, the model still generates fluent text, even if the content is incorrect.

Hallucination Through the Lens of This Example

If asked for a reference that does not exist, ChatGPT may generate a plausible-looking citation.
This happens because:

- The model has learned the structure of citations.
- It optimizes plausibility, not truth.
- There is no grounding signal to stop generation.

> Hallucination is the cost of fluent generation under uncertainty.

Relating ChatGPT Back to Feature Importance

In classical models, feature importance is static.
In ChatGPT:

- Attention weights change across layers.
- Importance is context-dependent.
- Different tokens matter at different generation steps.

This dynamic weighting replaces explicit feature selection.

What This Case Study Teaches Us

ChatGPT demonstrates that:

- Transformers unify representation, weighting, and interaction.
- Language understanding can emerge from next-token prediction.
- Reasoning-like behavior can arise without symbolic rules.

At the same time, it highlights fundamental limitations of probabilistic text generation.

> ChatGPT is best understood not as a knowledge engine, but as a conditional probability machine operating over language.

21 Vision–Language Models

Why Vision–Language Models Were Introduced

Many real-world tasks require understanding both visual and textual information. Images alone lack semantic precision, while text alone lacks grounding in the physical world.

Vision–Language Models were introduced to jointly model visual and linguistic modalities, enabling systems to reason about images using language and to ground language in visual content.

Typical tasks include image captioning, visual question answering, image retrieval using text, and multimodal dialogue.

What Makes Vision–Language Models Different

Unlike language-only models, Vision–Language Models operate on two distinct input modalities:

- Visual inputs such as images or video frames
- Textual inputs such as captions, questions, or instructions

The core challenge is to align these modalities into a shared representational space where joint reasoning becomes possible.

From Pixels and Words to Representations

Visual inputs are first processed by a vision encoder, which converts images into a sequence of visual embeddings.

Text inputs are processed by a language encoder, which converts tokens into textual embeddings.

These embeddings are then fused using attention-based mechanisms.

> Vision–Language Models do not see images or read text. They operate on numerical representations learned from data.

Core Architectural Paradigms

There are three dominant architectural strategies used in Vision–Language Models.

Dual-Encoder Models

In dual-encoder architectures, images and text are encoded separately.

Similarity between image and text embeddings is computed using a dot product or cosine similarity.

These models are efficient and scalable, but they have limited capacity for fine-grained reasoning.

Cross-Encoder Models

Cross-encoder architectures jointly process visual and textual embeddings through cross-attention layers.
Text tokens can attend to image regions, and image regions can attend to text tokens.
These models are more expressive but computationally expensive.

Encoder–Decoder Models

In encoder–decoder architectures, a multimodal encoder builds a joint representation, and a decoder generates text conditioned on visual input.
This paradigm is commonly used for image captioning and multimodal dialogue.

Cross-Attention: The Key Mechanism

Cross-attention allows tokens from one modality to attend to representations from another modality.
For example, a word such as *cat* can attend to visual regions corresponding to the animal in an image.
This mechanism enables grounding of language in visual evidence.

> Cross-attention replaces explicit alignment rules with learned correspondence.

Training Objectives

Vision–Language Models are trained using a combination of objectives, including:

- Contrastive learning to align image and text representations
- Masked language modeling conditioned on images
- Caption generation and reconstruction losses

Contrastive objectives encourage matching image–text pairs to be closer than mismatched pairs.

How Vision–Language Models Differ from Language Models

Compared to language-only Transformers:

- Vision–Language Models require an additional vision encoder
- Attention operates both within and across modalities
- Representations must be aligned across heterogeneous input spaces

Language models generate text based purely on linguistic context. Vision–Language Models condition generation on both visual and textual context.

Strengths of Vision–Language Models

Pros

- Ground language in visual evidence
- Enable multimodal reasoning
- Improve robustness in perceptual tasks
- Support new interaction paradigms such as visual dialogue

Limitations and Challenges

Cons

- High computational cost
- Dependence on large, carefully aligned datasets
- Sensitivity to visual bias and spurious correlations
- Limited interpretability of cross-modal attention

Failure Modes

Vision–Language Models can fail when:

- Visual cues are ambiguous or misleading
- Training data contains biased associations
- The model relies on language priors instead of visual evidence

These failures are often subtle and difficult to detect.

> Multimodal grounding reduces hallucination, but does not eliminate it.

Relation to Feature Importance and Attention

In Vision–Language Models:

- Attention dynamically selects relevant visual regions
- Importance depends on both visual and textual context
- Feature relevance changes across layers and modalities

This generalizes classical feature importance to multimodal, context-dependent relevance.

Why Vision–Language Models Matter

Vision–Language Models represent a step toward grounded intelligence.

They integrate perception and language, allowing models to connect abstract symbols to observable reality.

> Vision–Language Models move learning closer to how humans combine perception and language, while still operating as statistical systems.

22 Multimodal Hallucination and Grounding Limits

What Is Multimodal Hallucination?

Multimodal hallucination occurs when a Vision–Language Model generates text that is inconsistent with the visual input or fabricates visual details that are not present.

Unlike language-only hallucination, multimodal hallucination arises from a failure to correctly ground language in perception.

Examples include:

- Describing objects that do not exist in an image
- Misidentifying attributes such as color, count, or spatial relations
- Over-relying on linguistic priors instead of visual evidence

Why Grounding Is Hard

Grounding requires aligning abstract linguistic symbols with noisy, high-dimensional perceptual signals.

Several factors make this difficult:

- Visual representations are lossy and ambiguous
- Training datasets contain biased image–text correlations
- Language priors often dominate weak visual cues

Attention Does Not Guarantee Grounding

Although Vision–Language Models use cross-attention, high attention weights do not ensure correct grounding.

The model may attend to visually irrelevant regions while still producing fluent descriptions.

> Attention aligns representations. It does not enforce truth.

Structural Limits of Multimodal Grounding

Grounding failures are structural rather than accidental.

They arise because:

- The training objective rewards plausibility
- There is no explicit verification of visual claims
- Perceptual uncertainty is not modeled symbolically

As a result, Vision–Language Models reduce hallucination relative to language-only models but cannot eliminate it entirely.

23 Vision–Language Models vs Symbolic Perception

What Is Symbolic Perception?

Symbolic perception systems explicitly represent visual scenes using structured symbols.

Typical representations include:

- Object identities
- Attributes such as color and size
- Spatial relationships
- Logical constraints

These representations are often generated using pipelines that include object detection, segmentation, and rule-based reasoning.

Key Differences in Representation

Vision–Language Models learn continuous vector representations.

Symbolic systems use discrete, human-interpretable symbols.

This leads to fundamental differences:

- VLMs are flexible but opaque
- Symbolic systems are interpretable but brittle

Reasoning Capabilities

Symbolic systems support explicit reasoning through logic and rules.

Vision–Language Models approximate reasoning through statistical pattern matching.

This allows VLMs to generalize broadly but limits their ability to enforce strict logical consistency.

Error Profiles

Symbolic perception fails catastrophically when symbols are misdetected.

Vision–Language Models fail softly by producing plausible but incorrect outputs.

> Symbolic systems fail loudly. Neural systems fail fluently.

Hybrid Approaches

Recent research explores combining symbolic structure with neural representations.

These hybrid systems aim to:

- Retain flexibility of learned representations
- Introduce constraints for reasoning and verification

Such approaches reflect an ongoing tension between expressiveness and reliability.

24 Biological Analogies to Multimodal Integration

Multisensory Integration in the Brain

Biological intelligence integrates information from multiple sensory modalities such as vision, audition, and touch.

Neuroscience studies show that this integration occurs hierarchically and contextually.

Early sensory processing is modality-specific, while higher cortical areas integrate signals across modalities.

Attention as a Biological Principle

Attention in biological systems dynamically allocates processing resources based on task demands and context.

This resembles attention mechanisms in Transformers, where relevance is computed dynamically rather than fixed.

> Attention is not unique to Transformers. It is a general principle of intelligent systems.

Limits of the Analogy

Despite surface similarities, Vision–Language Models differ fundamentally from biological systems.

- Biological systems are embodied
- Learning is grounded in action and feedback
- Perception is tightly coupled to survival objectives

Vision–Language Models lack embodiment and operate purely on static data.

Why the Analogy Still Matters

Biological analogies provide intuition for:

- Hierarchical representation learning
- Context-dependent relevance
- Robust integration of noisy signals

However, they should not be mistaken for equivalence.

25 Practical Multimodal Example Workflow

Overview of a Typical Multimodal Pipeline

A practical Vision–Language workflow follows a sequence of steps that mirror the architecture.

1. Input acquisition
2. Representation encoding
3. Cross-modal interaction
4. Output generation

Step 1: Input Acquisition

The system receives:

- An image or set of images
- A textual prompt or question

Both inputs are preprocessed independently.

Step 2: Encoding

The vision encoder converts images into visual embeddings.
The language encoder converts text into token embeddings.
These embeddings form the basis for multimodal reasoning.

Step 3: Cross-Modal Fusion

Cross-attention layers allow text tokens to attend to visual embeddings.
This enables the model to associate words with visual regions dynamically.

Step 4: Generation or Prediction

The fused representation is used to:

- Generate a textual response
- Select an answer
- Retrieve relevant images

The output is produced autoregressively or through classification heads.

Where Errors Enter the Pipeline

Errors can arise at multiple stages:

- Ambiguous visual encoding
- Misalignment during cross-attention
- Language priors overriding perception

Understanding the pipeline helps diagnose failures.

> Multimodal systems fail at interfaces, not at single components.

Practical Implications

Effective use of Vision–Language Models requires:

- Careful prompt design
- Awareness of grounding limitations
- Human verification for critical tasks

These systems are powerful assistants, not autonomous perceptual agents.

26 Retrieval-Augmented Multimodal Models

Why Retrieval Is Necessary

Pure Vision–Language Models rely entirely on internal parameters learned during training. As a result, they suffer from limited factual grounding, outdated knowledge, and hallucination when uncertainty is high.

Retrieval-augmented models address these issues by incorporating external information sources during inference.

Core Idea of Retrieval Augmentation

Instead of generating outputs solely from internal representations, retrieval-augmented models perform two coupled operations:

- Retrieve relevant external information
- Condition generation on both input and retrieved evidence

The retrieved content may include documents, images, structured databases, or multimodal records.

Multimodal Retrieval

In multimodal settings, retrieval can involve:

- Text-to-text retrieval
- Image-to-text retrieval
- Text-to-image retrieval

A shared embedding space is often used to compute similarity across modalities.

Generation with Evidence

Once retrieved, external information is injected into the model context.

The Transformer then attends jointly to:

- Original input
- Retrieved evidence

This improves factual accuracy and reduces hallucination, but does not eliminate it.

> Retrieval augments memory. It does not guarantee understanding.

Limitations of Retrieval-Augmented Models

- Retrieval errors propagate downstream
- Retrieved content may be inconsistent or incomplete
- Increased system complexity and latency

Retrieval improves grounding, but reasoning remains probabilistic.

27 Causality and Counterfactual Reasoning Limits

Correlation vs Causation

Transformers and Vision–Language Models learn statistical associations from data.

They do not learn causal mechanisms in the interventionist sense.

As a result, they excel at prediction but struggle with causal inference.

What Counterfactual Reasoning Requires

Counterfactual reasoning asks questions of the form:

> What would have happened if some aspect of the input were different?

This requires:

- Explicit causal variables
- Intervention semantics
- Structural causal relationships

None of these are explicitly represented in standard Transformer architectures.

Why Language Models Appear Causal

Language models can generate plausible counterfactual explanations because training data contains many such narratives.

This creates the appearance of causal reasoning without true causal modeling.

> Language models describe causality. They do not manipulate it.

Implications for Multimodal Systems

Vision–Language Models may incorrectly infer causes from visual correlations, such as assuming intent or agency from appearance.

These errors are subtle and difficult to detect without explicit causal constraints.

28 Evaluation Metrics for Multimodal Systems

Why Evaluation Is Hard

Evaluating multimodal models is challenging because outputs are:

- High-dimensional
- Context-dependent
- Often subjective

Simple accuracy metrics fail to capture nuanced failures.

Task-Specific Metrics

Common evaluation approaches include:

- Caption similarity metrics for image captioning
- Exact match or accuracy for visual question answering
- Retrieval recall for multimodal retrieval tasks

These metrics often correlate weakly with human judgment.

Human Evaluation

Human evaluation remains essential for assessing:

- Factual correctness
- Grounding
- Safety and bias

However, it is expensive and difficult to scale.

Failure-Oriented Evaluation

Modern evaluation increasingly focuses on:

- Stress testing
- Adversarial prompts
- Out-of-distribution scenarios

This shifts emphasis from average performance to worst-case behavior.

> A model that performs well on average can still fail catastrophically.

29 Responsible Deployment and Epistemic Uncertainty

Understanding Epistemic Uncertainty

Epistemic uncertainty arises from lack of knowledge rather than noise.
In large models, this uncertainty is hidden behind fluent generation.
The model does not know when it does not know.

Why Uncertainty Is Hard to Represent

Transformers output point estimates over tokens.

They do not maintain explicit belief states or confidence bounds.

As a result, uncertainty is not communicated reliably to users.

Risks of Overconfidence

Overconfident outputs can lead to:

- Misinformation
- Automation bias
- Unsafe decision-making

These risks are amplified in multimodal systems where visual evidence appears authoritative.

Principles for Responsible Deployment

Responsible use of multimodal models requires:

- Clear communication of limitations
- Human oversight for critical tasks
- Monitoring and auditing of failures

Designing for Uncertainty

Effective systems incorporate:

- Retrieval and verification
- Conservative generation strategies
- Explicit refusal or deferral mechanisms

> Intelligence without humility is dangerous. Uncertainty awareness is a design requirement.

30 Conclusion: From Fixed Features to Learned Interactions

This document has traced the evolution of machine learning models from classical statistical methods to modern multimodal Transformer-based systems.

At the foundation lie linear and probabilistic models, which frame learning as parameter estimation over fixed features. These models offer interpretability, convex optimization, and strong theoretical guarantees, but they rely heavily on feature engineering and impose global constraints on data geometry.

Tree-based models relax linearity by introducing hierarchical partitioning. They capture non-linear interactions naturally, but at the cost of stability and global optimality. Ensemble methods mitigate these weaknesses through variance and bias reduction, trading interpretability for performance.

Support Vector Machines reformulate learning as a geometric optimization problem. By maximizing margins and leveraging kernel methods, they extend linear classifiers into rich function spaces while retaining strong generalization principles.

Feature selection methods such as RFE and RFECV reflect the limits of feature-centric learning. They attempt to control complexity by identifying relevant dimensions, but they assume that importance is static and global.

Transformers represent a fundamental shift. Rather than selecting features, they learn representations. Rather than assigning fixed importance, they compute relevance dynamically through attention. Representation, weighting, and interaction are unified into a single differentiable framework.

Language models operationalize this idea at scale. Conversational systems demonstrate how reasoning-like behavior, explanation, and abstraction can emerge from next-token prediction, while also revealing structural limitations such as hallucination and lack of grounding.

Vision–Language Models extend this paradigm by integrating perception and language. They ground symbols in visual input, reduce hallucination relative to text-only systems, and enable new forms of interaction. At the same time, they expose new failure modes related to multimodal bias and alignment.

Retrieval-augmented systems, causal analysis, and responsible deployment practices acknowledge an important reality. Statistical learning alone is insufficient for truth, safety, and reliability. External grounding, verification, and human oversight are necessary components of real-world systems.

Across this progression, a consistent theme emerges. Machine learning models do not become intelligent by accumulating rules. They become powerful by learning how information interacts.

> The history of machine learning is the gradual replacement of fixed assumptions with learned structure.

Understanding this trajectory is essential not only for using modern models effectively, but also for recognizing their limits and designing systems that are robust, interpretable, and responsible.

31 Summary Comparison of Model Classes

Model Class	Core Assumption	Strengths	Key Limitations
Linear Models	Linear relationship between features and output	Convex optimization, interpretability, statistical guarantees	Limited expressiveness, heavy reliance on feature engineering
Logistic Regression	Linear decision boundary in log-odds space	Probabilistic outputs, calibration, stability	Cannot model complex interactions
Tree Models	Recursive partitioning of feature space	Non-linear modeling, interpretability (single trees)	High variance, greedy optimization
Ensemble Trees	Aggregation of multiple weak learners	Strong performance, robustness	Reduced interpretability, higher computation
Support Vector Machines	Maximum margin separation	Strong generalization, kernel flexibility	Sensitive to hyperparameters, limited scalability
Feature Selection (RFE, RFECV)	Static feature relevance	Improved interpretability, variance control	Model-dependent, computationally expensive
Neural Networks	Hierarchical representation learning	Expressive power, automatic feature learning	Non-convex optimization, data-hungry
Transformers	Attention-based interaction learning	Long-range dependency modeling, scalability	High compute cost, limited interpretability
Language Models	Next-token probability modeling	Generalization, reasoning-like behavior	Hallucination, lack of grounding
Vision–Language Models	Joint visual and linguistic representations	Multimodal grounding, richer interaction	Visual bias, complex failure modes
Retrieval-Augmented Models	External evidence integration	Improved factual accuracy, reduced hallucination	System complexity, retrieval errors
Symbolic Systems	Explicit structured representations	Interpretability, logical consistency	Brittleness, poor generalization

Table 1: Comparative overview of major machine learning model classes, highlighting their core assumptions, strengths, and limitations.

> No single model class is universally superior. Each reflects a different tradeoff between structure, flexibility, and reliability.

32 Why don't you try yourself?

Easy

1. Why are linear models considered foundational in machine learning?
2. In one sentence, what problem does Logistic Regression solve?
3. What is a decision boundary?
4. What is the sigmoid function used for in Logistic Regression?
5. What does a decision tree do at each split?
6. Name two impurity metrics used in trees.
7. What is the maximum-margin idea in SVM?
8. What is a kernel, at a high level?
9. What does cross-validation try to estimate?
10. Why do we need positional encodings in Transformers?
11. In one sentence, what is self-attention?
12. What is a token in language modeling?
13. What does it mean for a language model to be autoregressive?
14. Give one example task for a Vision–Language Model.

Medium

1. Explain why Logistic Regression uses cross-entropy loss instead of squared error.
2. For $f(x) = w^T x + b$, explain what x, w, and b represent.
3. Define the bias–variance tradeoff and give one example of each extreme.
4. Why do trees optimize impurity reduction instead of classification error at splits?
5. Compare entropy and Gini impurity. What do they measure conceptually?
6. Why are single decision trees high-variance learners?
7. Explain bagging and how it reduces variance.
8. Explain boosting and why it often reduces bias.
9. What does the C parameter control in soft-margin SVM?
10. What is the intuition behind the RBF kernel and the role of γ?
11. What is "data leakage" in feature selection, and why is it harmful?
12. Compare filter, wrapper, and embedded feature selection methods.
13. Describe the algorithmic loop of RFE in your own words.
14. What extra idea does RFECV add beyond RFE?
15. Explain what queries, keys, and values represent in self-attention.
16. Why is the attention score scaled by $\sqrt{d_k}$?
17. What is multi-head attention, and why is it useful?
18. Why can a language model generate fluent text that is factually wrong?
19. Explain the difference between a language-only model and a Vision–Language Model.
20. Contrast dual-encoder and cross-encoder Vision–Language architectures.

Hard

1. Derive the chain rule factorization of $P(x_1, \ldots, x_n)$ used in language modeling and explain its implication for training objectives.
2. Show why the Logistic Regression objective is convex, and explain what convexity implies for optimization.
3. Explain, using margin geometry, why minimizing $\|w\|$ in SVM corresponds to maximizing the margin.
4. Give a rigorous definition of hinge loss and explain how it differs from classification error and log-loss.
5. Provide a concrete example where RFE fails due to feature correlation or interaction effects, and explain why.
6. Explain why RFECV should be nested within cross-validation for honest performance estimates, and identify the leakage pathway if it is not.
7. Discuss why attention weights are not guaranteed to be faithful explanations of model decisions. Propose two alternative analysis methods.
8. Formally describe causal masking in autoregressive Transformers and explain what goes wrong if it is removed.
9. Explain, mechanistically, why hallucination is a structural consequence of likelihood-based training. Identify at least two mitigation strategies and their limitations.
10. Compare Vision–Language Models with symbolic perception systems in terms of representation, reasoning, and failure modes. When would you prefer each?
11. Define multimodal hallucination and propose an evaluation protocol that isolates grounding failures from language priors.
12. Explain why Transformers are correlation-based learners and why counterfactual reasoning requires additional structure. Provide an example counterfactual query and discuss what a model would need to answer it reliably.
13. Describe a retrieval-augmented multimodal pipeline end to end. Identify two failure points in retrieval and two failure points in generation.
14. Propose an evaluation suite for multimodal systems that includes both average-case metrics and stress tests. Justify the inclusion of worst-case evaluation.
15. Define epistemic uncertainty in the context of generative models and propose a system design pattern that encourages appropriate deferral under uncertainty.

Easy: definitions and basic intuition. **Medium**: mechanism-level explanations and comparisons. **Hard**: derivations, failure analysis, and system-level reasoning.