MACHINE LEARNING & ANALYTICS

Visual Cheat Sheet Guide

Complete Reference for Model Selection, Feature Engineering, Statistical Tests, and Best Practices

SUPERVISED LEARNING | UNSUPERVISED | DEEP LEARNING | NLP | STATISTICS

Print-Ready • Laminate-Friendly • Color-Coded

Updated October 2025

📋 TABLE OF CONTENTS

🎯 How to Use This Guide

- **Color Coding:** Each topic has a distinct color for quick navigation
- **Decision Trees:** Follow flowcharts for model selection
- **Comparison Tables:** Side-by-side comparison of techniques
- **Practical Tips:** Colored boxes highlight best practices and warnings
- **Cross-References:** Page numbers link related topics

✅ Best Practices

Start Simple: Always baseline with simple models (Logistic Regression, Random Forest) before complex ones.

XGBoost Usually Wins: For tabular data, XGBoost/LightGBM beats neural networks 90% of the time.

Cross-Validate Everything: Never rely on a single train/test split.

🔄 MACHINE LEARNING WORKFLOW

1️⃣ UNDERSTAND THE PROBLEM

↓

- What are you predicting? (Classification / Regression / Generation)
- What's the success metric? (Accuracy / F1 / RMSE / Business KPI)
- What constraints exist? (Time / Compute / Interpretability)

2️⃣ EXPLORE YOUR DATA (EDA)

↓

- Sample size? (100 / 1K / 10K / 100K / 1M+)
- Feature count? (10 / 100 / 1000+)
- Data types? (Numerical / Categorical / Text / Images)
- Missing values? Outliers? Class imbalance?

3️⃣ FEATURE ENGINEERING

↓

Categorical Features

- Tree models → Label/Target encoding
- Linear models → One-hot encoding

Numerical Features

- Linear models → Scale (StandardScaler)
- Tree models → No scaling needed

4️⃣ MODEL SELECTION → See Page 4

↓

Based on Data Type

- Tabular → XGBoost/RF
- Images → CNN
- Text → Transformers
- Time Series → ARIMA/LSTM

Based on Sample Size

- <1K → Simple models
- 1K-10K → XGBoost
- 10K+ → DL possible
- 100K+ → DL recommended

5️⃣ TRAINING & VALIDATION

↓

- Split: 70-80% train, 10-15% validation, 10-15% test
- Use stratified split for imbalanced data
- Cross-validation (5-fold minimum)
- Hyperparameter tuning on validation set

6️⃣ EVALUATION → See Pages 16-17

🎯 MODEL SELECTION MASTER DECISION TREE

Step 1: What Type of Data?

📊 TABULAR / STRUCTURED DATA (Most Common)

Sample Size	Recommended Models	Why
< 1,000 samples	Logistic Regression OR Random Forest	Simple models generalize better with limited data
1K - 10K	**XGBoost (best)** OR Random Forest	XGBoost handles mixed types, missing values excellently
10K - 100K	**XGBoost / LightGBM** (winner 90%)	Beats neural networks on tabular data consistently
100K+	**LightGBM (fastest)** OR Deep Learning	LightGBM optimized for large datasets; DL if complex interactions

🖼️ IMAGES

Dataset Size	Recommended Approach	Models
< 10K images	**Transfer Learning**	ResNet, EfficientNet, Vision Transformers
10K - 100K	Fine-tune pretrained CNN	ResNet50, EfficientNet-B0, MobileNet
100K+	Train from scratch OR fine-tune	EfficientNet-B7, Custom CNNs, ViT

📝 TEXT / NLP

Sample Size	Recommended Models	Notes
< 1K samples	TF-IDF + Logistic Reg OR Few-shot LLM	Simple baseline works well; LLMs for complex tasks
1K - 100K	**Fine-tune BERT/RoBERTa**	Pretrained language models + task-specific tuning
100K+	Fine-tune large Transformer (GPT, T5)	Enough data for larger models

📈 TIME SERIES

Data Points	Recommended Models	Why

🗂️ DATA TYPE → MODEL MAPPING

Quick Reference Table

Problem Type	Data Type	Sample Size	Best Model	Alternative	Avoid
Classification	Tabular	<1K	Random Forest	Logistic Reg	Deep Learning
	Tabular	1K-100K	**XGBoost**	LightGBM	One-hot + Trees
	Tabular	100K+	**LightGBM**	XGBoost	-
Regression	Tabular (Linear)	Any	Ridge Regression	Lasso	-
	Tabular (Non-linear)	1K+	**XGBoost**	Random Forest	-
Images	Classification	<10K	Transfer Learning	ResNet/EfficientNet	Train from scratch
	Classification	100K+	**Fine-tune CNN**	Train from scratch	-
Text / NLP	Classification	<1K	TF-IDF + LogReg	Few-shot GPT	BERT (too little data)
	Classification	1K-10K	**Fine-tune BERT**	RoBERTa	-
	Generation	Any	GPT-4 / GPT-3.5	Llama 3 / Mistral	Train from scratch
Time Series	Univariate	<100	ARIMA	Exp. Smoothing	Deep Learning
	Univariate	100-10K	**Prophet**	SARIMA	-
	Multivariate	10K+	**LSTM**	XGBoost + lags	-

Problem-Specific Recommendations

📊 E-Commerce / Business

Customer Churn: XGBoost
Product Recommendations: Collaborative Filtering
Price Prediction: XGBoost / LightGBM
Sentiment Analysis: BERT

🏥 Healthcare / Life Sciences

Disease Prediction: Random Forest (interpretable)
Medical Image Analysis: CNN (ResNet)
Clinical Text: BioBERT
Drug Discovery: Graph Neural Networks

💰 Finance / Risk

Fraud Detection: XGBoost + SMOTE
Credit Scoring: Logistic Reg (interpretable)
Stock Prediction: LSTM / XGBoost
Anomaly Detection: Isolation Forest

🎯 Marketing / Growth

Customer Segmentation: K-Means
LTV Prediction: XGBoost
A/B Test Analysis: Statistical Tests (p9-10)
Ad Click Prediction: XGBoost

🏷️ CATEGORICAL ENCODING GUIDE

> ⚠️ **CRITICAL: Encoding Choice Dramatically Affects Performance!**
>
> Wrong encoding can reduce accuracy by 10-20%. **Tree models MUST NOT use one-hot encoding.**

Encoding Selection by Model Type

Encoding	Tree Models (RF, XGBoost)	Linear Models (LogReg, SVM)	Neural Nets	Cardinality	When to Use
Label	✓ Excellent	✗ Bad	△ OK	< 50	Default for trees, fast, no dimensions added
One-Hot	✗ Bad	✓ Required	△ OK	< 10	Linear models with low cardinality
Target	✓✓ Best	✓ Good	✓ Good	50+	High cardinality, use with CV to prevent leakage
Frequency	✓ Good	✓ Good	✓ Good	Any	Safe, no leakage, simple
Binary	✓ Good	✓ Good	✓ Good	10-100	Balance between one-hot and label
Ordinal	✓ Good	✓ Good	✓ Good	Any	Natural ordering exists (Small/Med/Large)
Hash	✓ Good	△ OK	✓ Good	1000+	Very high cardinality, memory constrained
Embedding	✗ N/A	✗ N/A	✓✓ Best	Any	Neural nets, learns representations

Decision Tree: Which Encoding?

What model are you using? TREE-BASED (Random Forest, XGBoost, LightGBM) | ├─ Cardinality < 50? | └─ Use: **Label Encoding** (default) | └─ Cardinality ≥ 50 (high)? ├─ Use: **Target Encoding** (with CV!) └─ Alternative: Frequency Encoding ❌ NEVER use One-Hot with trees (slower, worse performance) LINEAR MODELS (Logistic, Linear Reg, SVM) | ├─ Cardinality < 10? | └─ Use: **One-Hot Encoding** | ├─ Cardinality 10-50? | ├─ Use: Target Encoding | └─ OR: One-Hot + dimensionality reduction | └─ Cardinality > 50? └─ Use: **Target Encoding** OR Hash NEURAL NETWORKS | └─ Use: **Embedding Layer** (learns best representation)

🚫 Target Encoding Leakage Prevention


```
# ❌ WRONG - Leakage!
df['city_encoded'] = df.groupby('city')['target'].transform('mean')
```

📐 NUMERICAL TRANSFORMATIONS

Scaling: When and How?

Model Type	Scaling Needed?	Best Scaler	Why
Tree-Based (RF, XGBoost, DT)	✗ NO	None needed	Trees split on thresholds, scale doesn't matter
Linear Models (LogReg, Linear Reg)	✓ YES	StandardScaler	Coefficients depend on scale, gradient descent sensitive
SVM	✓ YES	StandardScaler	Distance-based, very sensitive to scale
Neural Networks	✓ YES	StandardScaler OR MinMaxScaler	Helps training stability and convergence
K-NN, K-Means	✓ CRITICAL	StandardScaler	Distance-based, large features dominate otherwise
Naive Bayes	✗ NO	None needed	Works on probabilities, not distances

Scaler Types

StandardScaler

Formula: z = (x - μ) / σ
Result: mean=0, std=1
Use for: Default choice for linear/neural
Sensitive to: Outliers

MinMaxScaler

Formula: x' = (x - min) / (max - min)
Result: range [0, 1]
Use for: Neural nets with sigmoid, images
Sensitive to: Outliers (very)

RobustScaler

Formula: x' = (x - median) / IQR
Result: median=0
Use for: Data with outliers
Robust to: Outliers ✓

Distribution Transformations

Log Transform

When: Right-skewed data (long tail)
Formula: log(x) or log(x+1) for zeros
Best for: Linear models
Example: Income, prices, counts

```
# For positive values
df['log_income'] = np.log(df['income'])

# For values with zeros
df['log_sales'] = np.log1p(df['sales'])
```

Square Root

When: Moderate skewness
Formula: √x
Best for: Count data
Less aggressive than log

```
df['sqrt_count'] = np.sqrt(df['count'])

# Box-Cox (optimal transform)
from scipy.stats import boxcox
df['transformed'], lambda = boxcox(df['col'])
```

🔧 FEATURE ENGINEERING TIPS

DateTime Features

Basic Extraction

```
df['year'] = df['date'].dt.year
df['month'] = df['date'].dt.month
df['day'] = df['date'].dt.day
df['hour'] = df['date'].dt.hour
df['day_of_week'] = df['date'].dt.dayofweek
df['quarter'] = df['date'].dt.quarter
df['week_of_year'] = df['date'].dt.isocalendar().week
```

Binary Indicators

```
# Weekend
df['is_weekend'] = df['day_of_week'].isin([5,6])

# Business hours
df['is_business_hours'] = df['hour'].between(9,17)

# Month end
df['is_month_end'] = df['date'].dt.is_month_end

# Holiday
import holidays
us_holidays = holidays.US()
df['is_holiday'] = df['date'].dt.date.isin(us_holidays)
```

🔄 Cyclical Encoding (for Linear Models & Neural Nets)

Why? Hour 23 is close to hour 0, but numerically they're far (23 vs 0). Cyclical encoding fixes this.

```
# Hour (24-hour cycle)
df['hour_sin'] = np.sin(2 * np.pi * df['hour'] / 24)
df['hour_cos'] = np.cos(2 * np.pi * df['hour'] / 24)

# Day of week (7-day cycle)
df['dow_sin'] = np.sin(2 * np.pi * df['day_of_week'] / 7)
df['dow_cos'] = np.cos(2 * np.pi * df['day_of_week'] / 7)

# Month (12-month cycle)
df['month_sin'] = np.sin(2 * np.pi * df['month'] / 12)
df['month_cos'] = np.cos(2 * np.pi * df['month'] / 12)
```

Domain-Specific Features

Domain	Feature Ideas	Example Code
E-Commerce	• Avg order value • Purchase frequency • Days since last purchase • Discount %	`df['avg_order'] = df['total_spent'] / df['num_orders']` `df['discount_pct'] = (df['orig_price'] - df['sale_price']) / df['orig_price']`
Finance	• Debt-to-income ratio • Credit utilization • Payment history • Income volatility	`df['debt_to_income'] = df['debt'] / df['income']` `df['credit_util'] = df['balance'] / df['limit']`

📊 STATISTICAL TESTS SELECTION

Test Selection Flowchart

```
What are you comparing? TWO CATEGORICAL VARIABLES └─ Use: Chi-Square Test Example: Gender vs Product Preference Check: Expected count ≥ 5 in 80% of cells ONE CATEGORICAL + ONE NUMERICAL ├─ 2 Groups? │ ├─ Independent samples? │ │ ├─ Normal? → Independent t-test │ │ └─ NOT normal? → Mann-Whitney U │ │ └─ Paired (before/after)? │ ├─ Normal? → Paired t-test │ └─ NOT normal? → Wilcoxon │ └─ 3+ Groups? ├─ Normal? → One-Way ANOVA └─ NOT normal? → Kruskal-Wallis TWO NUMERICAL VARIABLES ├─ Linear relationship? → Pearson Correlation └─ Monotonic? → Spearman Correlation
```

Test Comparison Matrix

Test	# Groups	Data Type	Paired?	Parametric?	When to Use
Chi-Square	2+	Categorical	No	Non-param	Test independence of 2 categorical variables
t-test	2	Continuous	No	Yes	Compare means, normal data
Paired t-test	2	Continuous	Yes	Yes	Before/after, normal data
Mann-Whitney U	2	Ordinal/Continuous	No	Non-param	Compare 2 groups, non-normal
Wilcoxon	2	Ordinal/Continuous	Yes	Non-param	Paired data, non-normal
ANOVA	3+	Continuous	No	Yes	Compare 3+ groups, normal
Kruskal-Wallis	3+	Ordinal/Continuous	No	Non-param	Compare 3+ groups, non-normal

Chi-Square vs T-test vs ANOVA

Chi-Square

Use when:
- Both variables categorical
- Testing independence
- Have contingency table

Example:
Smoking (Yes/No) vs
Lung Cancer (Yes/No)

Don't use if:
- Expected count < 5 in many cells
→ Use Fisher's Exact

T-test

Use when:
- Comparing 2 groups only
- Continuous outcome
- Normal distribution

Example:
Male vs Female salaries

Don't use if:
- Non-normal data
→ Use Mann-Whitney U
- 3+ groups → Use ANOVA

ANOVA

Use when:
- Comparing 3+ groups
- Continuous outcome
- Normal distribution

Example:
Drug A vs B vs C vs Placebo

Follow-up:
If significant → Run Tukey HSD to find which groups differ

🧪 A/B TESTING BEST PRACTICES

Sample Size Calculation (BEFORE Running Test!)

```
import scipy.stats as stats
import numpy as np

def calculate_sample_size(baseline_rate, mde, alpha=0.05, power=0.80):
    """
    baseline_rate: Current conversion rate (e.g., 0.05 for 5%)
    mde: Minimum detectable effect (e.g., 0.10 for 10% relative lift)
    alpha: Significance level (typically 0.05)
    power: Statistical power (typically 0.80)
    """
    p1 = baseline_rate
    p2 = baseline_rate * (1 + mde)
    p_pooled = (p1 + p2) / 2

    z_alpha = stats.norm.ppf(1 - alpha / 2)
    z_beta = stats.norm.ppf(power)

    n = ((z_alpha * np.sqrt(2*p_pooled*(1-p_pooled)) +
          z_beta * np.sqrt(p1*(1-p1) + p2*(1-p2)))**2) / ((p2-p1)**2)

    return int(np.ceil(n))

# Example
sample_size = calculate_sample_size(baseline_rate=0.05, mde=0.10)
print(f"Need {sample_size:,} per variant (total: {sample_size*2:,})")
```

Test Duration Guidelines

Factor	Minimum	Recommended	Why
Duration	1 week	2-4 weeks	Capture weekly patterns, avoid day-of-week bias
Sample Size	Calculate first!	Use power analysis	Ensure enough power to detect effect
Traffic Split	50/50	50/50 or 90/10	50/50 for equal power, 90/10 for low risk
Business Cycles	1 full cycle	2+ cycles	Account for weekend/weekday differences

Analyzing Results

```
def analyze_ab_test(conv_a, visit_a, conv_b, visit_b):
    """Analyze A/B test with statistical significance"""
    rate_a = conv_a / visit_a
    rate_b = conv_b / visit_b

    # Pooled proportion
    p_pooled = (conv_a + conv_b) / (visit_a + visit_b)

    # Standard error
    se = np.sqrt(p_pooled * (1-p_pooled) * (1/visit_a + 1/visit_b))
```

🧠 DEEP LEARNING: WHEN TO USE

Should You Use Deep Learning?

```
Data Type? ├─ TABULAR/STRUCTURED │ └─ ❌ Use XGBoost/Random Forest instead │ Exception: 1M+ rows with complex interactions │ ├─ IMAGES → ✅ Use CNNs ├─ TEXT → ✅ Use Transformers ├─ AUDIO/VIDEO → ✅ Use specialized architectures └─ TIME SERIES → ⚠️ Try traditional methods first (ARIMA, Prophet) Sample Size? ├─ < 10K → ❌ Too small, use Traditional ML ├─ 10K-100K → ⚠️ Transfer learning OR Traditional ML └─ > 100K → ✅ Deep Learning viable Resources? ├─ No GPU → ❌ Use Traditional ML (10-100x faster) ├─ Limited time → ❌ XGBoost trains faster, less tuning └─ Have GPU + time → ✅ Deep Learning possible Need Interpretability? ├─ YES → ❌ Use Linear/Tree models └─ NO → ✅ Deep Learning OK
```

When Deep Learning Wins

✅ Use Deep Learning

- **Unstructured data:** Images, audio, video
- **NLP tasks:** Translation, generation, Q&A
- **Large datasets:** 100K+ samples
- **Complex patterns:** Non-linear relationships
- **Automatic features:** Don't want manual engineering
- **Have GPU:** 10-100x faster training

❌ Don't Use Deep Learning

- **Tabular data:** XGBoost wins 90% of time
- **Small datasets:** <10K samples
- **Need interpretability:** Healthcare, finance, legal
- **No GPU:** Training too slow
- **Quick prototyping:** XGBoost faster to iterate
- **Production constraints:** Models are large, slow inference

Architecture Selection

Architecture	Data Type	Use Cases	Key Strength
MLP (Fully Connected)	Tabular	Classification, Regression on structured data	Simple baseline, but XGBoost usually better
CNN (Convolutional)	Images, Spatial	Image classification, Object detection	Captures spatial relationships
RNN/LSTM	Sequences	Time series, Text (older approach)	Memory of past inputs
Transformer	Text, Sequences	NLP (BERT, GPT), Translation	Long-range dependencies, parallelizable
Autoencoder	Any	Dimensionality reduction, Denoising	Unsupervised learning
GAN	Images	Image generation, Style transfer	Generative modeling

⚙️ NEURAL NETWORK TRAINING GUIDE

Key Hyperparameters

Hyperparameter	Typical Values	Impact	How to Choose
Learning Rate	0.001 (Adam) 0.01 (SGD)	**Most important!** Too high: diverges Too low: slow	Use learning rate finder, start with 0.001
Batch Size	32, 64, 128	Small: better generalization Large: faster, stable	Start with 32-64, use powers of 2
Optimizer	Adam (default) SGD+momentum AdamW (NLP)	Affects convergence speed	Adam for most tasks, AdamW for Transformers
Dropout	0.2-0.5	Regularization, prevents overfitting	Start with 0.5 for FC layers, 0.2-0.3 for CNN
Weight Decay	1e-5 to 1e-4	L2 regularization	Start with 1e-5, increase if overfitting

Training Best Practices

Data Preparation

- **Normalize inputs:** Images to [0,1] or standardize
- **Use data augmentation:** For images (flip, rotate, crop)
- **Balance classes:** Oversample minority or use class weights
- **Split properly:** Train/val/test before any processing

Regularization

- **Dropout:** Add between layers (0.5 for dense)
- **Batch Normalization:** Stabilizes training
- **Early Stopping:** Stop when val loss stops improving
- **Weight Decay:** L2 regularization (1e-5)

Monitoring

- **Track both:** Training AND validation loss
- **Plot curves:** Visualize loss over epochs
- **Watch for overfitting:** Val loss increasing while train decreasing
- **Use TensorBoard:** For experiment tracking

Optimization

- **Use GPU:** 10-100x faster than CPU
- **Learning rate scheduler:** Reduce on plateau
- **Gradient clipping:** Prevent exploding gradients
- **Mixed precision:** Faster training on modern GPUs

🚫 Common Neural Network Mistakes

❌ **Not normalizing inputs**

❌ **Forgetting model.eval()**

📝 NLP & TRANSFORMERS QUICK REFERENCE

NLP Task Selection

Task	Sample Size	Best Model	Notes
Text Classification (Sentiment, Topic)	<1K 1K-10K 10K+	TF-IDF + LogReg Fine-tune BERT/RoBERTa Fine-tune RoBERTa-large	Simple baseline works well for small data
Text Generation (Stories, Articles)	Any	GPT-4 (best quality) GPT-3.5 (cost-effective) Llama 3 / Mistral (open)	Use API or self-host open models
Question Answering	1K+	Fine-tune BERT RoBERTa DeBERTa	Extractive: answer in passage Generative: create answer
Named Entity Recognition	1K+	Fine-tune BERT RoBERTa SpaCy (fast baseline)	Tag entities: person, location, org
Summarization	Any	BART T5 GPT-4	T5 good for open-source, GPT-4 for quality
Translation	-	MarianMT (specific pairs) T5 GPT-4	MarianMT fast, GPT-4 multi-language
Semantic Similarity	-	Sentence-BERT (SBERT) Universal Sentence Encoder	Embed sentences, compute cosine similarity

Popular Transformers (2025)

BERT Family

BERT: Encoder-only, bidirectional
Use for: Classification, NER, Q&A
Size: 110M params (base)
Fine-tune on: 1K+ samples

Variants:
• RoBERTa (optimized BERT)
• DistilBERT (faster, smaller)
• DeBERTa (improved)

GPT Family

GPT: Decoder-only, autoregressive
Use for: Generation, completion
Size: 1.5B - 175B params
Access: OpenAI API

Versions:
• GPT-3.5: Cost-effective
• GPT-4: Highest quality
• Llama 3: Open-source alt

T5 / BART

T5: Encoder-decoder, multi-task
Use for: Translation, summarization
Size: 60M - 11B params
Framework: "text-to-text"

BART:
Similar to T5, good for summarization

🎯 Quick Decision: Which Transformer?

📊 EVALUATION METRICS: CLASSIFICATION

Confusion Matrix

		Predicted	
		Positive	**Negative**
Actual	**Positive**	**TP** **True Positive**	FN False Negative
	Negative	FP False Positive	**TN** **True Negative**

Derived Metrics:

Accuracy = (TP + TN) / Total
Precision = TP / (TP + FP)
Recall = TP / (TP + FN)
F1-Score = 2 × (P × R) / (P + R)

When to Use Each Metric

Metric	Formula	When to Use	Example
Accuracy	(TP+TN) / Total	**Balanced classes only** Equal importance for both classes	Email: spam vs not spam (50/50 split)
Precision	TP / (TP+FP)	**False Positives costly** Want to be sure when predicting positive	Spam filter: Don't mark real email as spam
Recall	TP / (TP+FN)	**False Negatives costly** Must catch all positives	Cancer detection: Don't miss any cases
F1-Score	2×(P×R)/(P+R)	**Imbalanced classes** Balance precision & recall	Fraud detection (1% fraud rate)
ROC-AUC	Area under ROC curve	**Binary classification** Threshold-agnostic Balanced classes	Credit scoring: Good/Bad credit
PR-AUC	Area under P-R curve	**Imbalanced classes** Better than ROC-AUC	Rare disease detection (0.1% prevalence)
Log Loss	-Σ y log(p)	**Probabilistic predictions** Penalizes wrong confidence	When need calibrated probabilities

🚫 Common Metric Mistakes

❌ Using Accuracy for Imbalanced Data

Example: 99% negative class
Model predicting "all negative" gets 99% accuracy!

❌ Ignoring Business Context

High precision means few false alarms, but might miss true cases.
High recall means catch everything, but many false alarms.

📈 EVALUATION METRICS: REGRESSION

Regression Metrics Comparison

Metric	Formula	When to Use	Units	Range
MAE Mean Absolute Error	Σ|y - ŷ| / n	**Outliers should not dominate** Linear penalty for errors	Same as target	[0, ∞) Lower better
MSE Mean Squared Error	Σ(y - ŷ)² / n	**Large errors very bad** Penalizes large errors heavily	Target²	[0, ∞) Lower better
RMSE Root MSE	√(MSE)	**Same units as target** Interpretable, penalizes large errors	Same as target	[0, ∞) Lower better
R² R-squared	1 - (SS_res/SS_tot)	**% variance explained** Model comparison	None (ratio)	(-∞, 1] 1 = perfect
MAPE Mean Abs % Error	Σ|y-ŷ|/|y| / n × 100	**Relative errors matter** % error more meaningful	Percentage	[0, ∞) Lower better

Choosing the Right Metric

MAE (Mean Absolute Error)

Use when:
- Outliers present in data
- All errors equally important
- Want robust metric

Example: Predicting house prices
MAE = $15,000 means avg error is $15K

Pros: Easy to interpret, robust
Cons: Doesn't penalize large errors

RMSE (Root Mean Squared Error)

Use when:
- Large errors are very costly
- Want same units as target
- Standard regression metric

Example: Predicting sales
RMSE = 100 units means typical error ~100

Pros: Penalizes large errors, standard
Cons: Sensitive to outliers

R² (R-squared)

Use when:
- Want % variance explained
- Comparing multiple models
- Need interpretable metric

Example: R² = 0.85
Model explains 85% of variance

MAPE (Mean Abs % Error)

Use when:
- Relative errors more meaningful
- Scale varies widely
- Business cares about %

Example: MAPE = 5%
Predictions typically 5% off

⚖️ MODEL COMPARISON MATRIX

Performance vs Complexity Tradeoff

Model	Accuracy	Train Time	Inference	Interpretability	Tuning	Handles Missing	Scales to	Best For
Logistic Reg	⭐⭐⭐	⚡⚡⚡ Fast	<1ms	✅✅✅ High	⚙️ Minimal	Need imputation	100M rows	Baseline, interpretable
Decision Tree	⭐⭐	⚡⚡⚡ Fast	<1ms	✅✅✅ High	⚙️ Minimal	✅ Native	1M rows	Interpretable, fast
Random Forest	⭐⭐⭐⭐	⚡⚡ Medium	~10ms	✅✅ Medium	⚙️ Minimal	✅ Native	10M rows	Robust, good defaults
XGBoost	⭐⭐⭐⭐⭐	⚡⚡ Medium	~10ms	✅ Low	⚙️⚙️⚙️ High	✅ Native	100M rows	**Winner for tabular**
LightGBM	⭐⭐⭐⭐⭐	⚡⚡⚡ Fast	<5ms	✅ Low	⚙️⚙️⚙️ High	✅ Native	1B rows	**Fastest, scales best**
SVM	⭐⭐⭐	⚡ Slow	~5ms	✅ Low	⚙️⚙️ Medium	Need imputation	100K rows	Small datasets, kernels
K-NN	⭐⭐⭐	⚡⚡⚡ Fast	⚡ Slow*	✅✅ Medium	⚙️ Minimal	Need imputation	1M rows	Baseline, no training
Neural Net	⭐⭐⭐⭐	⚡ Slow†	~10ms	❌ Very Low	⚙️⚙️⚙️ High	Need imputation	1B rows	Images, text, audio
Naive Bayes	⭐⭐	⚡⚡⚡ Fast	<1ms	✅✅ Medium	⚙️ Minimal	✅ Handles well	10M rows	Text classification

*K-NN slow at inference (must compute distance to all points)
†Neural nets fast with GPU, slow on CPU

When to Use Each Model

Small Data (<1K)

1st: Logistic Regression
2nd: Random Forest
3rd: SVM

Avoid: Deep Learning (overfits)
XGBoost (needs tuning)

Medium (1K-100K)

1st: XGBoost ⭐
2nd: LightGBM
3rd: Random Forest

XGBoost wins 90%
of Kaggle competitions

Large (100K+)

1st: LightGBM ⭐
2nd: XGBoost
3rd: Neural Nets

LightGBM fastest
for tabular data

⚠️ Model Selection Reality Check

For Tabular Data:

- XGBoost/LightGBM beat neural nets 90% of time
- Start with Random Forest (good defaults)

For Unstructured Data:

- Images: Always use CNNs (transfer learning)
- Text: Use Transformers (BERT, GPT)

⚙️ HYPERPARAMETER TUNING GUIDE

Tuning Priority by Model

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=100-500,    # More trees = better
    max_depth=None,          # Or 10-50
    min_samples_leaf=1-10,   # Higher = regularize
    max_features='sqrt',     # For classification
    n_jobs=-1                # Use all cores
)
```

Priority: n_estimators, max_depth, min_samples_leaf

XGBoost

```
from xgboost import XGBClassifier

xgb = XGBClassifier(
    n_estimators=100-1000,    # Tune with learning_rate
    learning_rate=0.01-0.3,   # Lower = more trees
    max_depth=3-10,           # Deeper = more complex
    subsample=0.5-1.0,        # Row sampling
    colsample_bytree=0.5-1.0, # Column sampling
    min_child_weight=1-10,    # Regularization
    gamma=0-5                 # Min loss reduction
)
```

Priority: learning_rate + n_estimators, max_depth, subsample

Neural Networks

```
import torch.nn as nn
from torch.optim import Adam

model = nn.Sequential(
    nn.Linear(input_dim, 128),
    nn.ReLU(),
    nn.Dropout(0.2-0.5),     # Regularization
    nn.Linear(128, 64),
    nn.ReLU(),
    nn.Dropout(0.2-0.5),
    nn.Linear(64, output_dim)
)

optimizer = Adam(
    model.parameters(),
    lr=0.001,                # MOST IMPORTANT!
    weight_decay=1e-5        # L2 regularization
)
```

Priority: learning_rate, batch_size, dropout

LightGBM

```
from lightgbm import LGBMClassifier

lgbm = LGBMClassifier(
    n_estimators=100-1000,
    learning_rate=0.01-0.3,
    max_depth=-1,             # No limit (or 3-10)
    num_leaves=31-255,        # Key parameter!
    subsample=0.5-1.0,
    colsample_bytree=0.5-1.0,
    min_child_samples=20      # Regularization
)
```

Priority: num_leaves, learning_rate, max_depth

Tuning Methods Comparison

Method	When to Use	Pros	Cons

💻 COMMON ALGORITHMS: CODE REFERENCE

Classification

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Scale features (REQUIRED for linear models)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train model
lr = LogisticRegression(
    C=1.0,              # Inverse regularization (lower = more reg)
    penalty='l2',       # or 'l1' for feature selection
    max_iter=1000
)
lr.fit(X_train_scaled, y_train)

# Predict
y_pred = lr.predict(X_test_scaled)
y_prob = lr.predict_proba(X_test_scaled)
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

# NO SCALING NEEDED for trees!
rf = RandomForestClassifier(
    n_estimators=100,
    max_depth=None,
    min_samples_split=2,
    random_state=42,
    n_jobs=-1
)
rf.fit(X_train, y_train)

# Predict
y_pred = rf.predict(X_test)

# Feature importance
import pandas as pd
importances = pd.DataFrame({
    'feature': X.columns,
    'importance': rf.feature_importances_
}).sort_values('importance', ascending=False)
```

XGBoost

```
from xgboost import XGBClassifier

# NO SCALING NEEDED
xgb = XGBClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=6,
    random_state=42,
    eval_metric='logloss'  # or 'auc', 'error'
)

# Train with validation for early stopping
xgb.fit(
    X_train, y_train,
    eval_set=[(X_val, y_val)],
    early_stopping_rounds=10,
    verbose=False
)

# Predict
y_pred = xgb.predict(X_test)
```

LightGBM

```
from lightgbm import LGBMClassifier

# NO SCALING NEEDED
lgbm = LGBMClassifier(
    n_estimators=100,
    learning_rate=0.1,
    num_leaves=31,
    random_state=42
)

lgbm.fit(
    X_train, y_train,
    eval_set=[(X_val, y_val)],
    callbacks=[lgbm.early_stopping(10)]
)

y_pred = lgbm.predict(X_test)
```

Regression

📋 QUICK REFERENCE SUMMARY

The ML Cheat Sheet

Scenario	Data Type	Sample Size	Best Model	Encoding	Metric
Customer churn	Tabular	10K-100K	XGBoost	Label/Target	F1-Score
House price	Tabular	1K-10K	XGBoost	Label/Target	RMSE
Image classification	Images	<10K	Transfer Learning (ResNet)	N/A	Accuracy
Sentiment analysis	Text	1K-10K	Fine-tune BERT	Tokenization	F1-Score
Fraud detection	Tabular	100K+	XGBoost + SMOTE	Target	PR-AUC
Credit scoring	Tabular	10K+	Logistic Reg (interpretable)	One-hot	ROC-AUC
Time series forecasting	Sequential	<1K	ARIMA / Prophet	N/A	MAE
Recommendation	User-item	100K+	Collaborative Filtering	Matrix factorization	NDCG

Golden Rules

✅ DO This

- **Start simple:** Logistic Reg → Random Forest → XGBoost
- **Split first:** Before any feature engineering
- **Cross-validate:** Always use CV, not just train/test
- **Check assumptions:** Normality, independence, etc.
- **Use right metric:** F1 for imbalanced, RMSE for regression
- **Feature importance:** Remove low-value features
- **Test significance:** Ensure improvements are real
- **Document everything:** Experiments, hyperparameters, results

❌ Don't Do This

- **Data leakage:** Never use test data during training
- **One-hot with trees:** Use Label/Target encoding
- **Accuracy for imbalanced:** Use F1 or PR-AUC
- **DL for tabular:** XGBoost beats it 90% of time
- **Ignore overfitting:** Monitor train vs val loss
- **Scale tree models:** They don't need it!
- **Too many features:** More features ≠ better
- **No baseline:** Always compare to simple model

Decision Framework Summary

Step 1: Understand Problem → Classification? Regression? Generation?

Step 2: Check Data Type → Tabular? Images? Text? Time Series?

Step 3: Check Sample Size → <1K? 1K-10K? 10K-100K? 100K+?

Step 4: Choose Model → Tabular→XGBoost, Images→CNN, Text→BERT

📚 RESOURCES & FINAL TIPS

Learning Resources

📖 Books

- **Hands-On Machine Learning** (Aurélien Géron)
  Best practical guide
- **The Elements of Statistical Learning**
  Theory & foundations
- **Deep Learning** (Goodfellow et al.)
  DL bible
- **Pattern Recognition and ML** (Bishop)
  Comprehensive theory

🎓 Courses

- **Fast.ai**
  Practical deep learning
- **Coursera ML (Andrew Ng)**
  Foundations
- **Stanford CS229**
  ML theory
- **HuggingFace Course**
  NLP & Transformers

🛠️ Tools & Libraries

- **scikit-learn:** Classical ML
- **XGBoost/LightGBM:** Gradient boosting
- **PyTorch/TensorFlow:** Deep learning
- **HuggingFace:** Transformers
- **Optuna:** Hyperparameter tuning

🏆 Practice

- **Kaggle:** Competitions & datasets
- **UCI ML Repository:** Classic datasets
- **Papers with Code:** Latest research
- **GitHub:** Open-source projects

Debugging Checklist

🐛 Model Not Learning?

- Check learning rate (too high/low?)
- Verify data preprocessing (normalized?)
- Check for data leakage
- Try simpler model first
- Visualize predictions vs actual
- Check for class imbalance

⚠️ Overfitting?

- Train loss << Val loss?
- Add regularization (dropout, L2)
- Get more data
- Reduce model complexity
- Use cross-validation
- Early stopping

📉 Underfitting?

- Train & val loss both high?
- Try more complex model
- Add more features
- Reduce regularization
- Train longer

✅ Good Performance?

- Test on holdout set
- Check statistical significance
- Look at confusion matrix
- Analyze errors
- Test on edge cases

- Check feature engineering

- A/B test in production

Final Words of Wisdom

🎯 Remember:

- **Simple is better than complex.** Start with simple models, add complexity only if needed.
- **Data > Algorithms.** 100K samples with Random Forest beats 1K samples with deep learning.
- **Feature engineering matters.** Good features with simple model beats bad features with complex model.
- **Always validate properly.** Cross-validation is your friend. Test set is sacred.
- **XGBoost is king for tabular.** Don't fight it. Use it.
- **Deep learning for unstructured.** Images, text, audio → use DL. Tabular → use trees.
- **Metrics matter.** Choose the right one. Accuracy lies on imbalanced data.
- **Iterate and experiment.** ML is empirical. Try things, measure, improve.
- **Document everything.** Future you will thank present you.
- **Keep learning.** ML evolves fast. Stay curious! 🚀

🎓 You're Ready!

Keep this guide handy for your ML journey.
Laminate it, reference it, and keep building amazing models!

Created October 2025 • Print-Ready • 8.5" x 11"