Machine Learning Algorithms — Cheat Sheet

> **Comprehensive reference for ML engineers:** Core concepts, hyperparameters, evaluation metrics, pros/cons & practical use cases.
>
> *Last updated: August 10, 2025*

🔄 Quick ML Workflow

1. Data Collection → 2. Preprocessing → 3. Feature Engineering → 4. Model Selection → 5. Training → 6. Evaluation → 7. Hyperparameter Tuning → 8. Deployment → 9. Monitoring

> 💡 **Pro Tip:** Always start with a simple baseline model to set performance expectations and validate your pipeline before moving to complex algorithms.

🛠️ Data Preprocessing Essentials

Step	Best Practices
Missing Data	Impute with mean/median/mode for numerical, mode for categorical. Consider model-based imputation (KNN) for complex patterns. Always watch for data leakage.
Encoding	One-hot encoding for nominal categories, ordinal encoding when order matters, target/mean encoding for high-cardinality features.
Scaling	StandardScaler (z-score) for most ML algorithms, MinMaxScaler for neural networks, RobustScaler when outliers are present.
Outliers	Detect using IQR method or z-score. Handle by clipping, log transformation, or using robust algorithms.
Feature Selection	Filter methods (correlation), wrapper methods (RFE), embedded methods (L1 regularization), mutual information for relevance.
Data Splitting	Train/validation/test splits. Use stratified sampling for imbalanced classes, time-based splits for temporal data.

🎯 Model Selection & Tuning Guidelines

- **Start Simple:** Begin with interpretable baselines (linear regression, logistic regression, decision trees) to understand data behavior
- **Cross-Validation:** Use k-fold or stratified k-fold for robust performance estimates. Time series requires temporal splits
- **Search Strategy:** Random search for initial exploration → Bayesian optimization for expensive hyperparameter spaces. Grid search only for small discrete spaces
- **Learning Curves:** Monitor training vs validation performance to diagnose bias-variance tradeoff and detect overfitting

- **Production Preference:** Choose simpler models in production environments when performance is comparable to complex alternatives

---

Supervised Learning Algorithms

Linear Regression

Type: Regression | **Core:** Least squares fitting
Hyperparameters: `fit_intercept`, `regularization (alpha)`

- **Metrics:** MSE, RMSE, MAE, R^2
- **Pros:** Simple, fast, highly interpretable, no hyperparameters
- **Cons:** Assumes linear relationships, sensitive to outliers
- **Use Cases:** Price prediction, trend forecasting, baseline models

Logistic Regression

Type: Classification | **Core:** Sigmoid/softmax probability modeling
Hyperparameters: `penalty`, `C (inverse regularization)`

- **Metrics:** Accuracy, Precision, Recall, F1-Score, ROC-AUC
- **Pros:** Interpretable coefficients, works well with limited data, probabilistic output
- **Cons:** Linear decision boundary, limited for complex non-linear patterns
- **Use Cases:** Credit scoring, medical diagnosis, binary classification tasks

Decision Tree

Type: Both Tasks | **Core:** Recursive binary partitioning
Hyperparameters: `max_depth`, `min_samples_split`, `min_samples_leaf`

- **Metrics:** Accuracy, F1-Score, RMSE (regression)
- **Pros:** Highly interpretable, handles non-linear patterns, no data scaling needed
- **Cons:** Prone to overfitting, unstable (small data changes = different tree)
- **Use Cases:** Medical decision making, rule extraction, feature importance analysis

Random Forest

Type: Both Tasks | **Core:** Bootstrap aggregated trees (bagging)
Hyperparameters: `n_estimators`, `max_features`, `max_depth`

- **Metrics:** Accuracy, AUC-ROC, RMSE, feature importance
- **Pros:** Robust to overfitting, handles mixed data types, built-in feature importance
- **Cons:** Less interpretable than single tree, can be slower for inference
- **Use Cases:** Fraud detection, general-purpose classification, feature selection

Support Vector Machine (SVM)

Type: Both Tasks | **Core:** Maximum margin classifier with kernel trick
Hyperparameters: `C`, `kernel`, `gamma`

- **Metrics:** Accuracy, AUC-ROC, precision-recall
- **Pros:** Effective in high-dimensional spaces, memory efficient, versatile kernels
- **Cons:** Slow on large datasets, sensitive to feature scaling, no probabilistic output
- **Use Cases:** Text classification, bioinformatics, image classification

K-Nearest Neighbors (KNN)

Type: Both Tasks | **Core:** Instance-based lazy learning
Hyperparameters: `n_neighbors`, `weights`, `distance_metric`

- **Metrics:** Accuracy, F1-Score, MAE (regression)
- **Pros:** Simple concept, no training phase, works with non-linear data
- **Cons:** Computationally expensive at prediction time, sensitive to irrelevant features
- **Use Cases:** Recommendation systems, anomaly detection, imputation

Gradient Boosting (XGBoost/LightGBM)

Type: Both Tasks | **Core:** Sequential ensemble of weak learners
Hyperparameters: `learning_rate`, `n_estimators`, `max_depth`

- **Metrics:** AUC-ROC, RMSE, log-loss
- **Pros:** High predictive accuracy, handles missing values (some variants), feature importance
- **Cons:** Many hyperparameters to tune, prone to overfitting, sensitive to noise
- **Use Cases:** Kaggle competitions, ranking systems, tabular data prediction

---

🔍 Unsupervised Learning Algorithms

K-Means Clustering

Type: Clustering | **Core:** Centroid-based partitioning
Hyperparameters: `n_clusters`, `init`, `max_iter`

- **Metrics:** Inertia (WCSS), Silhouette Score, Elbow Method
- **Pros:** Fast, simple, works well with spherical clusters
- **Cons:** Assumes spherical clusters, requires pre-specified k, sensitive to initialization
- **Use Cases:** Customer segmentation, image segmentation, data compression

DBSCAN

Type: Clustering | **Core:** Density-based spatial clustering
Hyperparameters: `eps`, `min_samples`

- **Metrics:** Silhouette Score, Adjusted Rand Index
- **Pros:** Finds arbitrary shaped clusters, identifies outliers, no need to specify cluster count
- **Cons:** Sensitive to hyperparameters, struggles with varying densities
- **Use Cases:** Geospatial clustering, anomaly detection, image processing

Principal Component Analysis (PCA)

Type: Dimensionality Reduction | **Core:** Orthogonal linear projection
Hyperparameters: `n_components`

- **Metrics:** Explained variance ratio, reconstruction error
- **Pros:** Reduces dimensionality, removes multicollinearity, fast computation
- **Cons:** Linear transformation only, components less interpretable
- **Use Cases:** Data visualization, noise reduction, feature extraction

Association Rules (Apriori/FP-Growth)

Type: Association Mining | **Core:** Frequent itemset and rule discovery
Hyperparameters: `min_support`, `min_confidence`, `min_lift`

- **Metrics:** Support, Confidence, Lift, Conviction
- **Pros:** Discovers actionable business rules, interpretable results
- **Cons:** Computationally expensive for large itemsets, many redundant rules
- **Use Cases:** Market basket analysis, recommendation engines, web usage mining

---

🧠 Deep Learning Algorithms

Convolutional Neural Networks (CNN)

Type: Deep Learning | **Core:** Convolutional feature extraction
Hyperparameters: `filters`, `kernel_size`, `learning_rate`

- **Metrics:** Accuracy, Top-k accuracy, IoU (segmentation)
- **Pros:** Excellent for spatial data, translation invariant, hierarchical features
- **Cons:** Requires large datasets, computationally intensive, many hyperparameters
- **Use Cases:** Image classification, object detection, medical imaging

Recurrent Neural Networks (RNN/LSTM/GRU)

Type: Deep Learning | **Core:** Sequential processing with memory
Hyperparameters: `hidden_size`, `num_layers`, `dropout`

- **Metrics:** Accuracy, Perplexity, BLEU score (NLP tasks)
- **Pros:** Handles sequential dependencies, variable input length
- **Cons:** Vanishing gradient problem, slower training, sequential processing
- **Use Cases:** Time series forecasting, language modeling, speech recognition

Transformers

Type: Deep Learning | **Core:** Self-attention mechanism
Hyperparameters: `num_heads`, `d_model`, `num_layers`

- **Metrics:** BLEU, ROUGE, Accuracy, Perplexity
- **Pros:** Parallelizable training, captures long-range dependencies, state-of-the-art performance
- **Cons:** Extremely large computational and data requirements
- **Use Cases:** Natural language processing, machine translation, vision transformers

Reinforcement Learning (Q-Learning/DQN)

Type: Reinforcement Learning | **Core:** Learn optimal policy through interaction
Hyperparameters: `learning_rate`, `gamma (discount)`, `epsilon`

- **Metrics:** Cumulative reward, episode length, success rate
- **Pros:** Learns from trial and error, no labeled data needed, handles sequential decisions
- **Cons:** Sample inefficient, unstable training, requires simulation environment
- **Use Cases:** Game AI, robotics, autonomous vehicles, resource allocation

---

⚖️ Bias-Variance Tradeoff Quick Reference

Problem	Symptoms	Solutions
High Bias (Underfitting)	Poor performance on both training and validation sets	Increase model complexity, add features, reduce regularization
High Variance (Overfitting)	Good training performance, poor validation performance	Add regularization, get more data, reduce model complexity, early stopping

---

📊 Evaluation Metrics Quick Reference

Classification Metrics

Accuracy | **Precision** | **Recall** | **F1-Score** | **ROC-AUC** | **PR-AUC**

Regression Metrics

MSE | **RMSE** | **MAE** | **R^2** | **MAPE**

Clustering Metrics

Silhouette Score | **Davies-Bouldin Index** | **Calinski-Harabasz**

Ranking/RecSys Metrics

NDCG | **MAP@K** | **Precision@K** | **Recall@K**

NLP Metrics

BLEU | **ROUGE** | **Perplexity** | **METEOR**

🎯 Quick Model Selection Flow

1. **Data Size Assessment:** Small datasets (<1000 samples) → simple models (linear, logistic regression)
2. **Relationship Type:** Non-linear patterns → tree-based models, SVMs with RBF kernel
3. **Data Type:** Images → CNNs, Text → Transformers/RNNs, Tabular → Gradient boosting
4. **Performance Plateau:** Single models plateau → try ensemble methods
5. **Interpretability:** High requirement → linear models, decision trees, SHAP values

🔧 Regularization & Advanced Techniques

- **L1 Regularization (Lasso):** Feature selection, sparsity
- **L2 Regularization (Ridge):** Weight decay, handles multicollinearity
- **Elastic Net:** Combines L1 + L2 benefits
- **Dropout:** Prevents overfitting in neural networks
- **Batch Normalization:** Stabilizes deep network training
- **Early Stopping:** Prevents overfitting during training
- **Data Augmentation:** Artificially expand training data

🔍 Hyperparameter Optimization

Grid Search: Exhaustive search over parameter grid. Best for small, discrete spaces.

Random Search: Random sampling from parameter distributions. More efficient than grid search.

Bayesian Optimization: Uses previous evaluations to guide search. Best for expensive function evaluations.

Hyperband: Multi-armed bandit approach. Good for neural network architectures.

⚙️ Feature Engineering Tips

- **Domain Knowledge:** Often more impactful than algorithm choice
- **Polynomial Features:** Create interaction terms for linear models
- **Binning:** Convert continuous variables to categorical
- **Time-based Features:** Hour, day, season for temporal data
- **Text Features:** TF-IDF, word embeddings, n-grams
- **Image Features:** Edge detection, texture, color histograms

🚀 Model Deployment Considerations

- **Latency Requirements:** Real-time vs batch prediction needs
- **Model Size:** Memory and storage constraints
- **Scalability:** Horizontal vs vertical scaling capabilities
- **Monitoring:** Data drift, model performance degradation
- **A/B Testing:** Gradual rollout and performance comparison
- **Model Versioning:** Reproducibility and rollback capabilities

⚠️ Common Pitfalls to Avoid

- **Data Leakage:** Future information in features
- **Selection Bias:** Non-representative training data
- **Overfitting:** Model memorizes training data
- **Improper Cross-validation:** Using future data for past predictions
- **Ignoring Class Imbalance:** Biased toward majority class

- **Feature Scaling:** Forgetting to scale features for distance-based algorithms

---

Converting This Markdown to PDF

Method 1: Pandoc (Recommended)

```
pandoc ml-cheat-sheet.md -o ml-cheat-sheet.pdf --pdf-engine=wkhtmltopdf
```

Method 2: Online Converters

- **Markdown to PDF:** Use tools like MarkdownToPDF, Dillinger, or GitPrint
- **GitHub:** Push to GitHub and use GitPrint service

Method 3: VS Code Extension

- Install "Markdown PDF" extension
- Right-click → "Markdown PDF: Export (pdf)"

Method 4: Obsidian/Notion

- Import into Obsidian or Notion
- Export as PDF with custom styling

> **Final Note:** Feature engineering and data quality often have more impact on model performance than algorithm selection. Always start with clean, well-understood data.

Comprehensive ML Reference • Ready for PDF Export