Complete Machine Learning Algorithms & Libraries Cheat Sheet

Table of Contents

---

1. Core ML Libraries & Frameworks {#core-libraries}

Python Libraries

Scikit-learn

- **Purpose**: General-purpose ML library
- **Key Features**: Classification, regression, clustering, dimensionality reduction
- **Common Classes**:

```python
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

TensorFlow/Keras

- **Purpose**: Deep learning framework
- **Architecture Types**: Sequential, Functional API, Subclassing
- **Key Components**:
  - Layers: Dense, Conv2D, LSTM, GRU, Dropout, BatchNormalization
  - Optimizers: Adam, SGD, RMSprop, Adagrad
  - Loss Functions: MSE, CrossEntropy, Huber, Custom losses

PyTorch

- **Purpose**: Dynamic deep learning framework
- **Key Features**: Autograd, dynamic computation graphs
- **Core Components**:
  - torch.nn: Neural network modules
  - torch.optim: Optimization algorithms
  - torch.utils.data: DataLoader, Dataset classes

XGBoost/LightGBM/CatBoost

- **Purpose**: Gradient boosting frameworks
- **Key Parameters**:
  - learning_rate: Step size shrinkage
  - max_depth: Maximum tree depth
  - n_estimators: Number of boosting rounds
  - subsample: Fraction of samples per tree

---

2. Supervised Learning Algorithms {#supervised-learning}

Linear Models

Linear Regression

- **Equation**: $y = \beta_0 + \beta_1x_1 + \beta_2x_2 + ... + \beta_nx_n + \varepsilon$
- **Objective**: Minimize MSE = $\Sigma(y_i - \hat{y}_i)^2/n$
- **Methods**:
  - **Ordinary Least Squares (OLS)**: $\beta = (X'X)^{-1}X'y$
  - **Gradient Descent**: $\theta = \theta - \alpha\nabla J(\theta)$
- **Assumptions**: Linearity, independence, homoscedasticity, normality
- **Regularization**:
  - **Ridge (L2)**: $J(\theta) = MSE + \lambda\Sigma\theta_i^2$
  - **Lasso (L1)**: $J(\theta) = MSE + \lambda\Sigma|\theta_i|$
  - **Elastic Net**: $J(\theta) = MSE + \lambda_1\Sigma|\theta_i| + \lambda_2\Sigma\theta_i^2$

Logistic Regression

- **Function**: p(y=1|x) = 1/(1 + e^(-z)), where $z = \beta_0 + \beta_1x_1 + ... + \beta_nx_n$
- **Loss Function**: Binary Cross-Entropy = $-\Sigma[y_i\log(p_i) + (1-y_i)\log(1-p_i)]$

- **Optimization**: Maximum Likelihood Estimation via gradient descent
- **Multiclass Extensions**:
  - **One-vs-Rest (OvR)**: Train k binary classifiers
  - **Softmax Regression**: $p(y=k|x) = e^{(z_k)}/\Sigma e^{(z_j)}$

Tree-Based Models

Decision Trees

- **Splitting Criteria**:
  - **Gini Impurity**: $G = 1 - \Sigma p_i^2$
  - **Entropy**: $H = -\Sigma p_i \log_2(p_i)$
  - **Information Gain**: $IG = H(parent) - \Sigma(n_j/n)H(child_j)$
- **Algorithms**:
  - **CART**: Binary splits, supports regression
  - **ID3**: Uses entropy and information gain
  - **C4.5**: Handles continuous attributes, missing values
- **Pruning Methods**:
  - **Pre-pruning**: Stop growing based on criteria
  - **Post-pruning**: Remove branches that don't improve validation performance

Random Forests

- **Concept**: Ensemble of decision trees with bagging
- **Key Features**:
  - Bootstrap sampling for each tree
  - Random feature selection at each split
  - Out-of-bag (OOB) error estimation
- **Hyperparameters**:
  - n_estimators: Number of trees
  - max_features: Features to consider at each split
  - min_samples_split: Minimum samples to split node

Support Vector Machines (SVM)

Linear SVM

- **Objective**: Maximize margin = $2/||w||$
- **Constraints**: $y_i(w \cdot x_i + b) \geq 1$ for all i

- **Soft Margin**: Allows misclassification with penalty C
- **Dual Form**: Optimize α subject to $\Sigma\alpha_i y_i = 0$

Kernel SVM

- **Kernel Trick**: $K(x,y) = \varphi(x) \cdot \varphi(y)$
- **Common Kernels**:
  - **RBF**: $K(x,y) = \exp(-\gamma||x-y||^2)$
  - **Polynomial**: $K(x,y) = (\gamma x \cdot y + r)^d$
  - **Sigmoid**: $K(x,y) = \tanh(\gamma x \cdot y + r)$

Naive Bayes

Gaussian Naive Bayes

- **Assumption**: Features follow Gaussian distribution
- **Likelihood**: $P(x_i|y) = (1/\sqrt{(2\pi\sigma^2_y)})\exp(-(x_i-\mu_y)^2/2\sigma^2_y)$

Multinomial Naive Bayes

- **Use Case**: Text classification, discrete features
- **Likelihood**: $P(x_i|y) = (N_{yi} + \alpha)/(N_y + \alpha|V|)$
- **Smoothing**: Laplace smoothing with parameter α

k-Nearest Neighbors (k-NN)

- **Algorithm**: Find k nearest points, vote/average
- **Distance Metrics**:
  - **Euclidean**: $d = \sqrt{\Sigma(x_i-y_i)^2}$
  - **Manhattan**: $d = \Sigma|x_i-y_i|$
  - **Minkowski**: $d = (\Sigma|x_i-y_i|^p)^(1/p)$
- **Weighting Schemes**:
  - Uniform: Equal weights
  - Distance: Weight = 1/distance

---

3. Unsupervised Learning Algorithms {#unsupervised-learning}

Clustering Algorithms

K-Means

- **Objective**: Minimize within-cluster sum of squares (WCSS)

- **Algorithm**:
  1. Initialize k centroids randomly
  2. Assign points to nearest centroid
  3. Update centroids as mean of assigned points
  4. Repeat until convergence
- **Variants**:
  - **K-Means++**: Smart initialization
  - **Mini-Batch K-Means**: Subsample for efficiency

Hierarchical Clustering

- **Types**:
  - **Agglomerative**: Bottom-up approach
  - **Divisive**: Top-down approach
- **Linkage Criteria**:
  - **Single**: $\min\{d(a,b): a \in A, b \in B\}$
  - **Complete**: $\max\{d(a,b): a \in A, b \in B\}$
  - **Average**: $\Sigma d(a,b)/(|A||B|)$
  - **Ward**: Minimize within-cluster variance

DBSCAN

- **Parameters**:
  - eps: Maximum distance between points
  - min_samples: Minimum points to form dense region
- **Concepts**:
  - Core points: Have min_samples within eps
  - Border points: Within eps of core point
  - Noise points: Neither core nor border

Gaussian Mixture Models (GMM)

- **Model**: $p(x) = \Sigma \pi_k \mathcal{N}(x|\mu_k,\Sigma_k)$
- **EM Algorithm**:
  - **E-step**: Calculate responsibilities
  - **M-step**: Update parameters π, μ, Σ
- **Covariance Types**: Full, diagonal, spherical, tied

Association Rules

Apriori Algorithm

- **Metrics**:
  - **Support**: supp(X) = |t∈T: X⊆t|/|T|
  - **Confidence**: conf(X→Y) = supp(X∪Y)/supp(X)
  - **Lift**: lift(X→Y) = supp(X∪Y)/(supp(X)×supp(Y))

---

4. Deep Learning Architectures {#deep-learning}

Feedforward Neural Networks

Multi-Layer Perceptron (MLP)

- **Architecture**: Input → Hidden layers → Output
- **Forward Pass**: z = Wx + b, a = g(z)
- **Activation Functions**:
  - **ReLU**: f(x) = max(0,x)
  - **Sigmoid**: f(x) = 1/(1+e^(-x))
  - **Tanh**: f(x) = (e^x-e^(-x))/(e^x+e^(-x))
  - **Leaky ReLU**: f(x) = max(αx,x)
  - **GELU**: f(x) = x·Φ(x)

Convolutional Neural Networks (CNN)

Core Components

- **Convolution Layer**:
  - Operation: (f*g)(t) = Σf(τ)g(t-τ)
  - Parameters: filters, kernel_size, stride, padding
- **Pooling Layers**:
  - Max Pooling: Select maximum value
  - Average Pooling: Calculate mean
  - Global Pooling: Reduce to single value per channel

Popular Architectures

- **LeNet-5**: Early CNN for digit recognition
- **AlexNet**: Deep CNN with ReLU, dropout
- **VGGNet**: Small filters, deep architecture

- **ResNet**: Skip connections, residual blocks
- **Inception**: Multiple filter sizes in parallel
- **EfficientNet**: Compound scaling of depth/width/resolution

Recurrent Neural Networks (RNN)

Vanilla RNN

- **Hidden State**: $h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t + b)$
- **Output**: $y_t = W_{hy}h_t + b_y$
- **Problem**: Vanishing/exploding gradients

Long Short-Term Memory (LSTM)

- **Gates**:
  - **Forget Gate**: $f_t = \sigma(Wf[h_{t-1},x_t] + bf)$
  - **Input Gate**: $i_t = \sigma(Wi[h_{t-1},x_t] + bi)$
  - **Output Gate**: $o_t = \sigma(Wo[h_{t-1},x_t] + bo)$
- **Cell State**: $C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t$

Gated Recurrent Unit (GRU)

- **Gates**:
  - **Reset Gate**: $r_t = \sigma(Wr[h_{t-1},x_t])$
  - **Update Gate**: $z_t = \sigma(Wz[h_{t-1},x_t])$
- **Hidden State**: $h_t = (1-z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t$

Transformer Architecture

Self-Attention Mechanism

- **Query, Key, Value**: $Q = XW^{\varphi}$, $K = XW_k$, $V = XW_v$
- **Attention**: $\text{Attention}(Q,K,V) = \text{softmax}(QK'/\sqrt{d_k})V$
- **Multi-Head**: $\text{Concat}(\text{head}_1,...,\text{head}_h)W_o$

Positional Encoding

- **Sinusoidal**: PE(pos,2i) = sin(pos/10000^(2i/d))
- **Learned**: Trainable embedding vectors

Autoencoders

Types

- **Vanilla Autoencoder**: Encoder → Bottleneck → Decoder
- **Variational Autoencoder (VAE)**:
  - Encoder outputs μ and σ
  - Sample $z \sim N(\mu,\sigma^2)$
  - Loss = Reconstruction + KL divergence
- **Denoising Autoencoder**: Input corrupted data, reconstruct clean

Generative Adversarial Networks (GAN)

Architecture

- **Generator**: G(z) generates fake samples from noise z
- **Discriminator**: D(x) classifies real vs fake
- **Objective**: min_G max_D V(D,G) = E[log D(x)] + E[log(1-D(G(z)))]

Variants

- **DCGAN**: Convolutional architecture
- **WGAN**: Wasserstein distance loss
- **CycleGAN**: Unpaired image translation
- **StyleGAN**: Style-based generation

---

5. Reinforcement Learning {#reinforcement-learning}

Core Concepts

Markov Decision Process (MDP)

- **Components**: (S, A, P, R, γ)
  - S: State space
  - A: Action space
  - P: Transition probability
  - R: Reward function
  - γ: Discount factor

Value Functions

- **State Value**: $V^\pi(s) = E[\Sigma\gamma^t r_t|s_0=s]$
- **Action Value**: $Q^\pi(s,a) = E[\Sigma\gamma^t r_t|s_0=s,a_0=a]$
- **Bellman Equations**:
  - $V^\pi(s) = \Sigma_a\pi(a|s)\Sigma_{s'}P(s'|s,a)[R(s,a,s') + \gamma V^\pi(s')]$

Algorithms

Dynamic Programming

- **Policy Iteration**: Evaluate → Improve → Repeat
- **Value Iteration**: $V(s) = \max_a \Sigma_{s'} P(s'|s,a)[R + \gamma V(s')]$

Monte Carlo Methods

- **First-Visit MC**: Average returns from first visit
- **Every-Visit MC**: Average all returns
- **MC Control**: Use ε-greedy for exploration

Temporal Difference Learning

- **TD(0)**: $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
- **SARSA**: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
- **Q-Learning**: $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)]$

Deep RL

- **DQN**: Neural network approximates Q-function
  - Experience replay buffer
  - Target network for stability
- **Policy Gradient**:
  - **REINFORCE**: $\nabla J(\theta) = E[\nabla \log \pi(a|s) G]$
  - **Actor-Critic**: Separate value and policy networks
- **PPO**: Proximal Policy Optimization with clipped objective
- **A3C**: Asynchronous Advantage Actor-Critic

---

6. Ensemble Methods {#ensemble-methods}

Bagging (Bootstrap Aggregating)

Random Forest (covered above)

Extra Trees

- **Difference from RF**: Random thresholds for splitting
- **Advantages**: Faster training, more randomness

Boosting

AdaBoost

- **Weight Update**: $w_i = w_i \times \exp(\alpha_t \times I(y_i \neq h_t(x_i)))$
- **Classifier Weight**: $\alpha_t = 0.5 \times \log((1-\varepsilon_t)/\varepsilon_t)$
- **Final Prediction**: $H(x) = \text{sign}(\Sigma\alpha_t h_t(x))$

Gradient Boosting

- **Algorithm**:
  1. Initialize with constant prediction
  2. For m = 1 to M:
     - Compute residuals $r_i = y_i - F_{m-1}(x_i)$
     - Fit tree h_m to residuals
     - Update $F_m = F_{m-1} + \nu h_m$
- **Loss Functions**: MSE, MAE, Huber, Quantile

XGBoost

- **Objective**: $\text{Obj} = \Sigma l(y_i, \hat{y}_i) + \Sigma\Omega(f_k)$
- **Regularization**: $\Omega(f) = \gamma T + 0.5\lambda\Sigma w_j^2$
- **Features**:
  - Parallel tree construction
  - Built-in cross-validation
  - Missing value handling

Stacking

- **Level-0 Models**: Base learners (diverse algorithms)
- **Level-1 Model**: Meta-learner combines base predictions
- **Cross-Validation**: Avoid overfitting with CV predictions

---

7. Dimensionality Reduction {#dimensionality-reduction}

Linear Methods

Principal Component Analysis (PCA)

- **Objective**: Find directions of maximum variance
- **Algorithm**:
  1. Standardize data
  2. Compute covariance matrix C

3. Find eigenvectors/eigenvalues of C
4. Select top k components

- **Transformation**: Z = XW, where W contains eigenvectors

Linear Discriminant Analysis (LDA)

- **Goal**: Maximize between-class/within-class variance ratio
- **Steps**:
  1. Compute class means and scatter matrices
  2. Solve generalized eigenvalue problem
  3. Project data onto discriminant vectors

Independent Component Analysis (ICA)

- **Assumption**: Sources are statistically independent
- **Methods**: FastICA, Infomax, JADE
- **Applications**: Blind source separation

Non-Linear Methods

t-SNE

- **Objective**: Preserve local neighborhoods
- **Perplexity**: Controls effective number of neighbors
- **Algorithm**:
  1. Compute pairwise similarities in high-D
  2. Initialize low-D representation
  3. Minimize KL divergence between distributions

UMAP

- **Advantages over t-SNE**: Preserves global structure, faster
- **Parameters**: n_neighbors, min_dist
- **Theory**: Based on Riemannian geometry

Autoencoders (covered above)

Matrix Factorization

Non-negative Matrix Factorization (NMF)

- **Constraint**: All values ≥ 0

- **Objective**: $||X - WH||^2$ subject to $W,H \geq 0$
- **Applications**: Topic modeling, image analysis

Singular Value Decomposition (SVD)

- **Decomposition**: $X = U\Sigma V'$
- **Truncated SVD**: Keep top k singular values
- **Applications**: Recommender systems, LSA

---

8. Evaluation Metrics {#evaluation-metrics}

Classification Metrics

Binary Classification

- **Accuracy**: $(TP + TN)/(TP + TN + FP + FN)$
- **Precision**: $TP/(TP + FP)$
- **Recall/Sensitivity**: $TP/(TP + FN)$
- **F1-Score**: $2 \times (Precision \times Recall)/(Precision + Recall)$
- **Specificity**: $TN/(TN + FP)$
- **AUC-ROC**: Area under ROC curve
- **AUC-PR**: Area under Precision-Recall curve

Multi-class Classification

- **Macro Average**: Average metric across classes
- **Weighted Average**: Weight by class frequency
- **Cohen's Kappa**: Agreement corrected for chance
- **Matthews Correlation Coefficient**: Balanced measure

Regression Metrics

- **Mean Absolute Error (MAE)**: $\Sigma|y_i - \hat{y}_i|/n$
- **Mean Squared Error (MSE)**: $\Sigma(y_i - \hat{y}_i)^2/n$
- **Root Mean Squared Error (RMSE)**: $\sqrt{MSE}$
- **R^2 Score**: $1 - \Sigma(y_i - \hat{y}_i)^2/\Sigma(y_i - \bar{y})^2$
- **Mean Absolute Percentage Error (MAPE)**: $\Sigma|y_i - \hat{y}_i|/|y_i| \times 100/n$

Clustering Metrics

- **Silhouette Score**: $(b - a)/\max(a, b)$

- **Davies-Bouldin Index**: Average similarity ratio
- **Calinski-Harabasz Index**: Between/within cluster variance
- **Adjusted Rand Index**: Corrected for chance

Cross-Validation Strategies

- **k-Fold**: Split data into k equal parts
- **Stratified k-Fold**: Preserve class distribution
- **Leave-One-Out**: k = n samples
- **Time Series Split**: Respect temporal order

---

9. Feature Engineering & Selection {#feature-engineering}

Feature Creation

Numerical Features

- **Binning**: Discretize continuous values
- **Polynomial Features**: x_1^2, x_1x_2, etc.
- **Log Transform**: Handle skewed distributions
- **Scaling Methods**:
  - StandardScaler: $(x - \mu)/\sigma$
  - MinMaxScaler: $(x - min)/(max - min)$
  - RobustScaler: Use median and IQR

Categorical Features

- **One-Hot Encoding**: Binary columns for each category
- **Label Encoding**: Integer representation
- **Target Encoding**: Replace with target mean
- **Ordinal Encoding**: For ordered categories

Text Features

- **Bag of Words**: Word frequency vectors
- **TF-IDF**: Term frequency × inverse document frequency
- **Word Embeddings**: Word2Vec, GloVe, FastText
- **N-grams**: Sequences of n words

Feature Selection

Filter Methods

- **Variance Threshold**: Remove low-variance features
- **Correlation**: Remove highly correlated features
- **Chi-Square Test**: For categorical targets
- **ANOVA F-test**: For continuous targets
- **Mutual Information**: Non-linear relationships

Wrapper Methods

- **Recursive Feature Elimination (RFE)**: Iteratively remove features
- **Forward Selection**: Start empty, add best
- **Backward Elimination**: Start full, remove worst

Embedded Methods

- **L1 Regularization**: Lasso induces sparsity
- **Tree-based Importance**: Gini/entropy reduction
- **Permutation Importance**: Shuffle and measure impact

---

10. Model Optimization Techniques {#optimization}

Hyperparameter Tuning

Grid Search

- **Method**: Exhaustive search over parameter grid
- **Pros**: Guaranteed to find best in grid
- **Cons**: Computationally expensive

Random Search

- **Method**: Sample random combinations
- **Pros**: More efficient than grid search
- **Theory**: Better coverage of important parameters

Bayesian Optimization

- **Method**: Build probabilistic model of objective
- **Acquisition Functions**:
  - Expected Improvement
  - Probability of Improvement

- Upper Confidence Bound

Genetic Algorithms

- **Operations**: Selection, crossover, mutation
- **Applications**: Complex search spaces

Gradient Descent Variants

Batch Gradient Descent

- **Update**: $\theta = \theta - \alpha \nabla J(\theta)$
- **Pros**: Stable convergence
- **Cons**: Slow for large datasets

Stochastic Gradient Descent (SGD)

- **Update**: $\theta = \theta - \alpha \nabla J(\theta; x_i, y_i)$
- **Pros**: Faster updates, can escape local minima
- **Cons**: Noisy updates

Mini-batch Gradient Descent

- **Balance**: Between batch and stochastic
- **Typical sizes**: 32, 64, 128, 256

Advanced Optimizers

Momentum

- **Update**: $v_t = \beta v_{t-1} + \alpha \nabla J(\theta)$
- **θ Update**: $\theta = \theta - v_t$
- **Effect**: Accelerates in consistent directions

RMSprop

- **Accumulate**: $E_t = \beta E_{t-1} + (1-\beta)(\nabla J)^2$
- **Update**: $\theta = \theta - \alpha \nabla J / \sqrt{(E_t + \varepsilon)}$

Adam (Adaptive Moment Estimation)

- **First moment**: $m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla J$
- **Second moment**: $v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla J)^2$
- **Bias correction**: $\hat{m}_t = m_t/(1-\beta_1^t)$, $\hat{v}_t = v_t/(1-\beta_2^t)$

- **Update**: $\theta = \theta - \alpha\hat{m}_t/(\sqrt{\hat{v}_t} + \varepsilon)$

AdamW

- **Modification**: Decoupled weight decay
- **Update**: $\theta = \theta - \alpha(\hat{m}_t/(\sqrt{\hat{v}_t} + \varepsilon) + \lambda\theta)$

Regularization Techniques

L1/L2 Regularization (covered above)

Dropout

- **Training**: Randomly zero out neurons with probability p
- **Inference**: Scale activations by (1-p)
- **Effect**: Ensemble of sub-networks

Batch Normalization

- **Normalize**: $\hat{x} = (x - \mu)/\sqrt{(\sigma^2 + \varepsilon)}$
- **Scale and Shift**: $y = \gamma\hat{x} + \beta$
- **Benefits**: Faster training, higher learning rates

Early Stopping

- **Monitor**: Validation loss
- **Patience**: Number of epochs without improvement
- **Restore**: Best weights when stopped

Data Augmentation

- **Images**: Rotation, flip, crop, color jitter
- **Text**: Synonym replacement, back-translation
- **Time Series**: Window slicing, noise injection

Advanced Techniques

Transfer Learning

- **Pre-trained Models**: Use existing trained models
- **Fine-tuning**: Adapt to new task
- **Feature Extraction**: Use as fixed feature extractor

Multi-task Learning

- **Shared Layers**: Common representations
- **Task-specific Heads**: Separate outputs
- **Loss**: Weighted sum of task losses

Meta-Learning

- **Goal**: Learn to learn
- **Approaches**: MAML, Prototypical Networks
- **Applications**: Few-shot learning

Federated Learning

- **Concept**: Train on distributed data
- **Privacy**: Data stays on device
- **Aggregation**: Average model updates

Quick Reference - Common Scikit-learn Code Patterns

python

```python
# Data Preprocessing
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Model Training & Evaluation
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train_scaled, y_train)
y_pred = model.predict(X_test_scaled)
print(classification_report(y_test, y_pred))

# Hyperparameter Tuning
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10]
}
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)
best_model = grid_search.best_estimator_

# Pipeline
from sklearn.pipeline import Pipeline
from sklearn.decomposition import PCA

pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=0.95)),
    ('classifier', RandomForestClassifier())
])
pipeline.fit(X_train, y_train)
```

This cheat sheet provides a comprehensive overview of machine learning algorithms, libraries, and techniques. Each section includes the mathematical foundations, key parameters, and practical

considerations for implementation. Use this as a reference guide for understanding and implementing various ML approaches in your projects.