In-Depth Guide to Core Machine Learning Algorithms

This detailed report delves deeper into the theory, implementation, hyperparameter tuning, and real-world applicability of the most important machine learning algorithms. It is organized by algorithm family and includes mathematical foundations, practical tips, and code examples.

1. Supervised Learning

1.1 Linear and Regularized Regression

Theory

- **Ordinary Least Squares (OLS)** minimizes
  $ J(\boldsymbol\beta) = \frac{1}{n}\sum_{i=1}^n (y_i - \boldsymbol\beta^\top \mathbf{x}_i)^2. $
- **Ridge Regression** adds ℓ_2 penalty:
  $ J(\boldsymbol\beta) + \lambda \|\boldsymbol\beta\|_2^2. $
- **Lasso** adds ℓ_1 penalty:
  $ J(\boldsymbol\beta) + \lambda \|\boldsymbol\beta\|_1. $

Hyperparameters

- **α (lambda)** in Ridge/Lasso: controls bias–variance trade-off.
- **Fit intercept**: include model bias term or assume zero-mean targets.

Code Example

```
from sklearn.linear_model import Ridge, Lasso
ridge = Ridge(alpha=1.0).fit(X_train, y_train)
lasso = Lasso(alpha=0.1).fit(X_train, y_train)
```

Applicability

- **Use when**: Relationship roughly linear; interpretability important; small-to-medium datasets.
- **Avoid when**: Highly nonlinear relationships prevail.

1.2 Logistic Regression

Theory

- Models $\Pr(y=1|\mathbf{x}) = \sigma(\boldsymbol{\beta}^\top \mathbf{x})$.
- **Loss:** Negative log-likelihood (cross-entropy):
  $ -\sum_i\bigl[y_i\log p_i + (1-y_i)\log(1-p_i)\bigr]. $

Hyperparameters

- **C = 1/λ**: inverse regularization strength.
- **Penalty**: 'l1', 'l2', or elastic net.
- **Solver**: 'liblinear' (small data), 'saga' (sparse), 'lbfgs' (dense).

Code Example

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(C=1.0, penalty='l2', solver='lbfgs', max_iter=200)
clf.fit(X_train, y_train)
```

Applicability

- **Use when**: Binary classification; feature interpretability; moderate feature count.
- **Watch out**: Poor performance on highly nonlinear separable data.

1.3 Tree-Based Models

1.3.1 Decision Trees

- **Criterion**: Gini impurity or entropy for splits.
- **Overfitting controls**: `max_depth`, `min_samples_leaf`, `min_samples_split`.

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier(max_depth=7, min_samples_leaf=5)
dt.fit(X_train, y_train)
```

1.3.2 Random Forests

- **Ensemble of trees** via bagging and feature randomness.
- **Key hyperparameters**: `n_estimators`, `max_features`, `max_depth`.

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=200, max_features='sqrt', max_depth=10)
rf.fit(X_train, y_train)
```

1.3.3 Gradient Boosting (XGBoost / LightGBM)

- **Boosting** sequentially corrects previous errors.
- **Hyperparameters**:
  - `learning_rate` (0.01–0.3),
  - `n_estimators` (100–1000),
  - `max_depth` (3–10),
  - `subsample` (0.6–1.0).

```
import xgboost as xgb
xgb_clf = xgb.XGBClassifier(
    n_estimators=500, learning_rate=0.05, max_depth=6, subsample=0.8)
xgb_clf.fit(X_train, y_train, eval_set=[(X_val,y_val)], early_stopping_rounds=50)
```

Applicability

- **Use when**: Complex nonlinear relationships; high accuracy required.
- **Trade-offs**: Longer training and careful tuning; less interpretable.

2. Unsupervised Learning

2.1 Clustering

2.1.1 K-Means

- **Objective**: Minimize within-cluster variance.
- **Elbow method**: Plot inertia vs. K to choose optimal number.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=5, init='k-means++', n_init=10)
labels = kmeans.fit_predict(X)
```

2.1.2 DBSCAN

- **Density-based** clusters with `eps`, `min_samples`.
- Good for arbitrary-shape clusters and noise detection.

```
from sklearn.cluster import DBSCAN
db = DBSCAN(eps=0.5, min_samples=5)
labels = db.fit_predict(X)
```

2.2 Dimensionality Reduction

2.2.1 PCA

- **Transforms** data to orthogonal principal components capturing variance.

```
from sklearn.decomposition import PCA
pca = PCA(n_components=0.95)
X_reduced = pca.fit_transform(X)
```

2.2.2 t-SNE / UMAP

- **Nonlinear** methods for visualization (2D/3D embeddings).

```
from sklearn.manifold import TSNE
X_embedded = TSNE(n_components=2, perplexity=30).fit_transform(X)
```

3. Neural Networks

3.1 Feedforward MLP

```
import torch.nn as nn
class MLP(nn.Module):
    def __init__(self, in_dim, h_dim, out_dim):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(in_dim, h_dim),
            nn.ReLU(),
            nn.Linear(h_dim, out_dim)
        )
    def forward(self, x): return self.net(x)
```

- **Hyperparameters**: Learning rate (1e-3–1e-5), batch size (32–256), hidden units, layers.
- **Tips**: Use dropout and batch normalization to improve generalization.

3.2 CNNs for Vision

```
from torchvision import models
model = models.resnet50(pretrained=True)
model.fc = nn.Linear(model.fc.in_features, num_classes)
```

- **Transfer learning**: Freeze early layers, fine-tune later.
- **Data augmentation**: Random crops, flips, color jitter.

3.3 Sequence Models & Transformers

```
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained("t5-base")
model = AutoModelForSeq2SeqLM.from_pretrained("t5-base")
```

- **Hyperparameters**:
  - Learning rate (1e-5–5e-5),
  - batch size (4–32),
  - warmup steps, weight decay.
- **Tips**: Use mixed precision (`fp16`) to speed up training.

4. Model Selection & Tuning Workflow

1. **Baseline**: Start with simple models (linear/logistic).
2. **Feature Engineering**: Create informative features, interactions, embeddings.
3. **Cross-Validation**: Use k-fold CV to estimate performance robustly.
4. **Hyperparameter Search**:
   - Grid or random search for small spaces.
   - Bayesian optimization (Optuna) for larger spaces.
5. **Ensembling**: Combine diverse models to reduce variance (stacking, voting).
6. **Interpretability**: Use SHAP or LIME to explain predictions.
7. **Monitoring**: Track metrics in production; retrain on drifted data.

5. Real-World Applicability Examples

Application	Recommended Algorithm	Notes
Credit Scoring	Logistic Regression, RF	Simple, interpretable, handles categorical features
Recommendation	Matrix Factorization, GBDT	Use user/item features + collaborative filtering
Anomaly Detection	Isolation Forest, DBSCAN	Unsupervised detection of outliers

Application	Recommended Algorithm	Notes
Sentiment Analysis	Transformers (BERT), SVM	Fine-tune BERT for best accuracy; SVM for speed
Time-Series Forecast	ARIMA, LSTM	ARIMA for linear trends, LSTM for complex sequences
Image Classification	CNN (ResNet, EfficientNet)	Transfer learning for small datasets

This deep dive equips you with the **theoretical grounding**, **implementation patterns**, **hyperparameter guidance**, and **use-case mapping** necessary to apply these algorithms in production settings.