Machine Learning Fundamentals

A Comprehensive Guide

Contents

1 Introduction to Machine Learning

Machine learning (ML) is a subset of artificial intelligence that enables systems to learn and improve from experience without being explicitly programmed. The fundamental goal is to develop algorithms that can learn patterns from data and make predictions or decisions.

1.1 Types of Machine Learning

- **Supervised Learning**: Learning from labeled data
- **Unsupervised Learning**: Finding patterns in unlabeled data
- **Reinforcement Learning**: Learning through interaction and feedback

2 Supervised Learning

Supervised learning involves training a model on labeled data, where each input $\mathbf{x}$ is associated with an output y. The goal is to learn a function $f : \mathcal{X} \to \mathcal{Y}$ that maps inputs to outputs.

2.1 Linear Regression

Linear regression models the relationship between input features and a continuous output variable.

2.1.1 Mathematical Formulation

Given a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$, we seek to find weights $\mathbf{w} \in \mathbb{R}^d$ and bias $b \in \mathbb{R}$ such that:

$$\hat{y} = \mathbf{w}^T \mathbf{x} + b = \sum_{j=1}^{d} w_j x_j + b \tag{1}$$

Cost Function (Mean Squared Error):

$$J(\mathbf{w}, b) = \frac{1}{2n} \sum_{i=1}^{n} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 \tag{2}$$

Gradient Descent Update Rules:

$$w_j := w_j - \alpha \frac{\partial J}{\partial w_j} = w_j - \alpha \frac{1}{n} \sum_{i=1}^{n} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i) x_{ij} \tag{3}$$

$$b := b - \alpha \frac{\partial J}{\partial b} = b - \alpha \frac{1}{n} \sum_{i=1}^{n} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i) \tag{4}$$

where α is the learning rate.

2.1.2 Python Implementation

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
```

```python

# Generate synthetic data
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X + np.random.randn(100, 1)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train model
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Coefficients: {model.coef_[0][0]:.2f}")
print(f"Intercept: {model.intercept_[0]:.2f}")
print(f"MSE: {mse:.2f}")
print(f"R   Score: {r2:.2f}")

# Visualization
plt.scatter(X_test, y_test, color='blue', label='Actual')
plt.plot(X_test, y_pred, color='red', linewidth=2, label='Predicted')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Linear Regression')
plt.show()
```

2.2 Logistic Regression

Logistic regression is used for binary classification problems where $y \in \{0, 1\}$.

2.2.1 Mathematical Formulation

Sigmoid Function:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{5}$$

Hypothesis:

$$h_{\mathbf{w}}(\mathbf{x}) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} \tag{6}$$

Cost Function (Cross-Entropy Loss):

$$J(\mathbf{w}) = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log(h_{\mathbf{w}}(\mathbf{x}_i)) + (1 - y_i) \log(1 - h_{\mathbf{w}}(\mathbf{x}_i)) \right] \tag{7}$$

2.2.2 Python Implementation

```python
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,
    classification_report

# Generate binary classification data
X, y = make_classification(n_samples=1000, n_features=20,
                           n_informative=15, n_redundant=5,
                           random_state=42)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train logistic regression
log_reg = LogisticRegression(random_state=42, max_iter=1000)
log_reg.fit(X_train, y_train)

# Predictions
y_pred = log_reg.predict(X_test)
y_pred_proba = log_reg.predict_proba(X_test)[:, 1]

# Evaluation
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f"Accuracy: {accuracy:.4f}")
print("\nConfusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

2.3 Decision Trees

Decision trees partition the feature space into regions and make predictions based on the majority class or average value in each region.

2.3.1 Mathematical Concepts

Gini Impurity (for classification):

$$G = 1 - \sum_{k=1}^{K} p_k^2 \tag{8}$$

where p_k is the proportion of samples belonging to class k.

Entropy (alternative splitting criterion):

$$H = -\sum_{k=1}^{K} p_k \log_2(p_k) \tag{9}$$

Information Gain:

$$IG = H(\text{parent}) - \sum_{\text{child}} \frac{n_{\text{child}}}{n_{\text{parent}}} H(\text{child}) \tag{10}$$

2.3.2 Python Implementation

```python
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.datasets import load_iris

# Load iris dataset
iris = load_iris()
X, y = iris.data, iris.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train decision tree
dt_classifier = DecisionTreeClassifier(
    max_depth=3,
    criterion='gini',
    random_state=42
)
dt_classifier.fit(X_train, y_train)

# Predictions
y_pred = dt_classifier.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

print(f"Decision Tree Accuracy: {accuracy:.4f}")

# Visualize tree
plt.figure(figsize=(15, 10))
plot_tree(dt_classifier, feature_names=iris.feature_names,
          class_names=iris.target_names, filled=True)
plt.title('Decision Tree Visualization')
plt.show()
```

3 Unsupervised Learning

Unsupervised learning discovers hidden patterns in data without labeled outputs.

3.1 K-Means Clustering

K-means partitions data into K clusters by minimizing within-cluster variance.

3.1.1 Mathematical Formulation

Objective Function:

$$J = \sum_{k=1}^{K} \sum_{\mathbf{x}_i \in C_k} \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 \tag{11}$$

where C_k is the set of points in cluster k and $\boldsymbol{\mu}_k$ is the centroid of cluster k.

Algorithm:

1. Initialize K centroids randomly
2. **Assignment step**: Assign each point to nearest centroid

$$c_i = \arg\min_k \|\mathbf{x}_i - \boldsymbol{\mu}_k\|^2 \tag{12}$$

3. **Update step**: Recompute centroids

$$\boldsymbol{\mu}_k = \frac{1}{|C_k|} \sum_{\mathbf{x}_i \in C_k} \mathbf{x}_i \tag{13}$$

4. Repeat steps 2-3 until convergence

3.1.2 Python Implementation

```
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Generate clustered data
X, y_true = make_blobs(n_samples=300, centers=4,
                       cluster_std=0.60, random_state=42)

# Apply K-means
kmeans = KMeans(n_clusters=4, random_state=42, n_init=10)
y_kmeans = kmeans.fit_predict(X)

# Visualization
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='red',
            s=200, alpha=0.75, marker='X')
plt.title('K-Means Clustering')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.show()

# Elbow method for optimal K
inertias = []
K_range = range(1, 11)
for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    inertias.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(K_range, inertias, 'bo-')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.grid(True)
plt.show()
```

3.2 Principal Component Analysis (PCA)

PCA reduces dimensionality by projecting data onto principal components that capture maximum variance.

3.2.1 Mathematical Formulation

Given data matrix $\mathbf{X} \in \mathbb{R}^{n \times d}$, PCA finds orthogonal directions of maximum variance.

Covariance Matrix:

$$\mathbf{\Sigma} = \frac{1}{n}\mathbf{X}^T\mathbf{X} \tag{14}$$

Eigenvalue Decomposition:

$$\mathbf{\Sigma}\mathbf{v}_i = \lambda_i \mathbf{v}_i \tag{15}$$

The principal components are the eigenvectors $\mathbf{v}_i$ corresponding to the largest eigenvalues λ_i.

Projection:

$$\mathbf{Z} = \mathbf{X}\mathbf{V}_k \tag{16}$$

where $\mathbf{V}_k$ contains the top k eigenvectors.

3.2.2 Python Implementation

```python
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# Load and standardize iris data
iris = load_iris()
X = iris.data
y = iris.target

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Apply PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Explained variance
print(f"Explained variance ratio: {pca.explained_variance_ratio_}")
print(f"Total variance explained: {sum(pca.explained_variance_ratio_)
    :.4f}")

# Visualization
plt.figure(figsize=(10, 6))
colors = ['red', 'green', 'blue']
for i, color in enumerate(colors):
    plt.scatter(X_pca[y == i, 0], X_pca[y == i, 1],
                color=color, label=iris.target_names[i], alpha=0.7)
plt.xlabel(f'PC1 ({pca.explained_variance_ratio_[0]:.2%} variance)')
plt.ylabel(f'PC2 ({pca.explained_variance_ratio_[1]:.2%} variance)')
plt.title('PCA of Iris Dataset')
plt.legend()
plt.grid(True)
plt.show()
```

4 Model Evaluation

Proper evaluation is crucial for assessing model performance and preventing overfitting.

4.1 Cross-Validation

K-Fold Cross-Validation: Split data into K folds, train on $K-1$ folds, validate on remaining fold. Repeat K times.

$$\text{CV Score} = \frac{1}{K}\sum_{i=1}^{K}\text{Score}_i \tag{17}$$

```python
from sklearn.model_selection import cross_val_score, KFold

# K-fold cross-validation
kfold = KFold(n_splits=5, shuffle=True, random_state=42)
model = LogisticRegression(random_state=42, max_iter=1000)

scores = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')

print(f"Cross-validation scores: {scores}")
print(f"Mean accuracy: {scores.mean():.4f} (+/- {scores.std():.4f})")
```

4.2 Evaluation Metrics

4.2.1 Classification Metrics

Accuracy:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{18}$$

Precision:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{19}$$

Recall (Sensitivity):

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{20}$$

F1-Score:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{21}$$

4.2.2 Regression Metrics

Mean Absolute Error (MAE):

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{22}$$

Root Mean Squared Error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{23}$$

R^2 Score (Coefficient of Determination):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{24}$$

```python
from sklearn.metrics import mean_absolute_error, mean_squared_error

# For regression
y_true = np.array([3, -0.5, 2, 7])
y_pred = np.array([2.5, 0.0, 2, 8])

mae = mean_absolute_error(y_true, y_pred)
mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_true, y_pred)

print(f"MAE: {mae:.4f}")
print(f"MSE: {mse:.4f}")
print(f"RMSE: {rmse:.4f}")
print(f"R  : {r2:.4f}")
```

4.3 Bias-Variance Tradeoff

The expected prediction error can be decomposed as:

$$\mathbb{E}[(y - \hat{f}(\mathbf{x}))^2] = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error} \tag{25}$$

where:

$$\text{Bias}^2 = (\mathbb{E}[\hat{f}(\mathbf{x})] - f(\mathbf{x}))^2 \tag{26}$$

$$\text{Variance} = \mathbb{E}[(\hat{f}(\mathbf{x}) - \mathbb{E}[\hat{f}(\mathbf{x})])^2] \tag{27}$$

Key Insights:

- High bias: underfitting (model too simple)
- High variance: overfitting (model too complex)
- Goal: balance bias and variance

5 Regularization

Regularization prevents overfitting by adding penalty terms to the cost function.

5.1 Ridge Regression (L2 Regularization)

$$J(\mathbf{w}) = \frac{1}{2n}\sum_{i=1}^{n}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 + \lambda\sum_{j=1}^{d} w_j^2 \tag{28}$$

5.2 Lasso Regression (L1 Regularization)

$$J(\mathbf{w}) = \frac{1}{2n}\sum_{i=1}^{n}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 + \lambda\sum_{j=1}^{d} |w_j| \tag{29}$$

```python
from sklearn.linear_model import Ridge, Lasso

# Ridge Regression
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
```

```
y_pred_ridge = ridge.predict(X_test)

# Lasso Regression
lasso = Lasso(alpha=1.0)
lasso.fit(X_train, y_train)
y_pred_lasso = lasso.predict(X_test)

print(f"Ridge MSE: {mean_squared_error(y_test, y_pred_ridge):.4f}")
print(f"Lasso MSE: {mean_squared_error(y_test, y_pred_lasso):.4f}")
```

6 Conclusion

This document covered fundamental machine learning concepts including supervised learning (linear regression, logistic regression, decision trees), unsupervised learning (K-means, PCA), and model evaluation techniques. Understanding these foundations is essential for developing more advanced ML systems.

6.1 Further Topics

- Neural Networks and Deep Learning
- Ensemble Methods (Random Forests, Gradient Boosting)
- Support Vector Machines
- Natural Language Processing
- Computer Vision