Code Samples by Learning Paradigm

1. Supervised Learning

Models learn from labeled data to predict targets. Below are examples for classification and regression using scikit-learn.

1.1 Classification with Iris Dataset

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Load data
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

# Train classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# Predict & evaluate
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred, target_names=load_iris().target_names))
```

1.2 Regression with California Housing

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error

# Load data
data = fetch_california_housing()
X_train, X_test, y_train, y_test = train_test_split(
    data.data, data.target, test_size=0.2, random_state=42)

# Train regressor
reg = Ridge(alpha=1.0)
reg.fit(X_train, y_train)

# Predict & evaluate
y_pred = reg.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse:.3f}")
```

2. Unsupervised Learning

Models uncover structure in unlabeled data. These examples show clustering and dimensionality reduction.

2.1 K-Means Clustering

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

# Generate synthetic data
X, _ = make_blobs(n_samples=300, centers=4, random_state=42)

# Fit K-Means
kmeans = KMeans(n_clusters=4, random_state=42)
labels = kmeans.fit_predict(X)

# Visualize
plt.scatter(X[:,0], X[:,1], c=labels, cmap='tab10')
plt.scatter(kmeans.cluster_centers_[:,0],
            kmeans.cluster_centers_[:,1],
            s=200, marker='X', c='black')
plt.title("K-Means Clustering")
plt.show()
```

2.2 PCA for Dimensionality Reduction

```
from sklearn.datasets import load_wine
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Load data
X, y = load_wine(return_X_y=True)

# Apply PCA
pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)

# Plot 2D projection
plt.scatter(X_reduced[:,0], X_reduced[:,1], c=y, cmap='tab10')
plt.title("Wine Dataset via PCA")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.show()
```

3. Semi-Supervised Learning

Uses both labeled and unlabeled data. Here we use Label Spreading from scikit-learn.

```
from sklearn import datasets
from sklearn.semi_supervised import LabelSpreading
from sklearn.metrics import accuracy_score
import numpy as np

# Load digits data
digits = datasets.load_digits()
n_total = len(digits.data)
n_labeled = 100

# Prepare labels: only first n_labeled are known, rest = -1
labels = np.full(n_total, -1, dtype=int)
labels[:n_labeled] = digits.target[:n_labeled]

# Train Label Spreading
model = LabelSpreading(kernel='knn', alpha=0.8)
model.fit(digits.data, labels)

# Predict on entire set and evaluate on true labels
y_pred = model.transduction_
print("Accuracy:", accuracy_score(digits.target, y_pred))
```

4. Self-Supervised Learning

Creates proxy tasks from raw data to learn representations. Below is a simple Rotation Prediction (RotNet) in PyTorch.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader

# Define simple CNN
class RotNet(nn.Module):
    def __init__(self, num_classes=4):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(1, 32, 3, stride=2, padding=1), nn.ReLU(),
            nn.Conv2d(32, 64, 3, stride=2, padding=1), nn.ReLU(),
            nn.AdaptiveAvgPool2d(1)
        )
        self.classifier = nn.Linear(64, num_classes)
```

```python
    def forward(self, x):
        x = self.features(x).view(x.size(0), -1)
        return self.classifier(x)

# Data transforms with random rotation
class RotDataset(torch.utils.data.Dataset):
    def __init__(self, dataset):
        self.dataset = dataset
        self.angles = [0, 90, 180, 270]

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
        img, _ = self.dataset[idx]
        angle = self.angles[torch.randint(0, 4, (1,)).item()]
        rot_img = transforms.functional.rotate(img, angle)
        label = self.angles.index(angle)
        return rot_img, label

# Prepare data loader
transform = transforms.Compose([transforms.ToTensor()])
mnist = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
rot_dataset = RotDataset(mnist)
loader = DataLoader(rot_dataset, batch_size=128, shuffle=True)

# Train loop
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = RotNet().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

for epoch in range(3):
    running_loss = 0.
    for imgs, labels in loader:
        imgs, labels = imgs.to(device), labels.to(device)
        optimizer.zero_grad()
        logits = model(imgs)
        loss = criterion(logits, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss: {running_loss/len(loader):.4f}")
```

5. Reinforcement Learning

Agents learn by interacting with an environment to maximize rewards. Below is a tabular Q-Learning example on FrozenLake-v1.

```
import gym
import numpy as np

env = gym.make("FrozenLake-v1", is_slippery=False)
Q = np.zeros((env.observation_space.n, env.action_space.n))
alpha, gamma, epsilon = 0.8, 0.95, 0.1
episodes = 2000

for ep in range(episodes):
    state = env.reset()
    done = False
    while not done:
        # Epsilon-greedy action
        if np.random.rand() < epsilon:
            action = env.action_space.sample()
        else:
            action = np.argmax(Q[state])

        next_state, reward, done, _ = env.step(action)
        # Q-Learning update
        best_next = np.max(Q[next_state])
        Q[state, action] += alpha * (reward + gamma * best_next - Q[state, action])
        state = next_state

# Test learned policy
state = env.reset()
env.render()
done = False
while not done:
    action = np.argmax(Q[state])
    state, _, done, _ = env.step(action)
    env.render()
```

These examples cover hands-on code for each paradigm.

You can adapt data sources, model architectures, and hyperparameters to deeper explore each technique.