Practical Implementation of Machine Learning Methods in Python

This course introduces practical machine learning techniques using Python. It provides step-by-step guidance for implementing key ML methods with commonly used libraries like **scikit-learn**, **TensorFlow**, and **pandas**.

1. Setting Up the Environment

Before starting, ensure you have the necessary libraries installed. Use the following commands:

```
pip install numpy pandas matplotlib seaborn scikit-learn tensorflow
```

2. Data Preparation

Most ML tasks begin with preprocessing the data. Let's load and inspect a dataset using **pandas**.

Example: Load and Explore the Dataset

```
import pandas as pd

# Load dataset
data = pd.read_csv("sample_dataset.csv")

# Explore data
print(data.head())
print(data.info())
print(data.describe())
```

Data Cleaning and Preprocessing

- Handle missing values:

  ```
  data.fillna(data.mean(), inplace=True)
  ```

- Encode categorical variables:

  ```
  from sklearn.preprocessing import LabelEncoder
  le = LabelEncoder()
  data['category'] = le.fit_transform(data['category'])
  ```

- Split data into features (X) and labels (y):

```
from sklearn.model_selection import train_test_split
X = data.drop('target', axis=1)
y = data['target']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

3. Implementing Machine Learning Models

3.1. Decision Trees

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Train a Decision Tree
clf = DecisionTreeClassifier(max_depth=3, random_state=42)
clf.fit(X_train, y_train)

# Make predictions
y_pred = clf.predict(X_test)

# Evaluate
print("Decision Tree Accuracy:", accuracy_score(y_test, y_pred))
```

Visualization of the Decision Tree

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
plot_tree(clf, feature_names=X.columns, class_names=True, filled=True)
plt.show()
```

3.2. Support Vector Machines (SVM)

```
from sklearn.svm import SVC

# Train an SVM
svm = SVC(kernel='rbf', C=1.0, gamma='scale')
svm.fit(X_train, y_train)

# Make predictions and evaluate
y_pred = svm.predict(X_test)
print("SVM Accuracy:", accuracy_score(y_test, y_pred))
```

3.3. Random Forests

```
from sklearn.ensemble import RandomForestClassifier

# Train a Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Evaluate
y_pred = rf.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred))
```

Feature Importance

```
importances = rf.feature_importances_
for feature, importance in zip(X.columns, importances):
    print(f"{feature}: {importance:.2f}")
```

3.4. Artificial Neural Networks (ANNs) with TensorFlow

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# Build the model
model = Sequential([
    Dense(64, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(32, activation='relu'),
    Dense(1, activation='sigmoid')  # Use 'softmax' for multi-class
classification
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_split=0.2)

# Evaluate on test data
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print("Neural Network Test Accuracy:", test_accuracy)
```

Plot Training History

```
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.show()
```

4. Hyperparameter Tuning

To improve model performance, tune hyperparameters using **GridSearchCV**.

```
from sklearn.model_selection import GridSearchCV

# Example: Grid search for SVM
param_grid = {'C': [0.1, 1, 10], 'gamma': ['scale', 'auto'], 'kernel':
['rbf', 'linear']}
grid = GridSearchCV(SVC(), param_grid, cv=5)
grid.fit(X_train, y_train)

print("Best Parameters:", grid.best_params_)
print("Best Cross-Validated Accuracy:", grid.best_score_)
```

5. Model Evaluation and Metrics

Evaluate model performance using classification metrics.

Confusion Matrix and Classification Report

```
from sklearn.metrics import confusion_matrix, classification_report

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)

# Classification report
print("Classification Report:\n", classification_report(y_test, y_pred))
```

ROC Curve and AUC

```
from sklearn.metrics import roc_curve, auc

# Compute probabilities for the positive class
y_proba = rf.predict_proba(X_test)[:, 1]

# ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

# Plot
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {roc_auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
```

6. Practical Tips

1. **Data Scaling**: Use StandardScaler for algorithms like SVM or KNN.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

2. **Imbalanced Data**: Use **SMOTE** or class weights.

```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

3. **Save and Load Models**: Persist models using joblib or TensorFlow.

```
import joblib
joblib.dump(rf, 'random_forest_model.pkl')
loaded_rf = joblib.load('random_forest_model.pkl')
```

7. Hands-On Practice

Explore these datasets for practice:

- **Iris Dataset** (Classification): Available in `sklearn.datasets`.
- **Boston Housing Dataset** (Regression): Use `from sklearn.datasets import load_boston`.
- **MNIST Dataset** (Deep Learning): Available via `tensorflow.keras.datasets`.