1　機械学習で学ぶ Python の基礎

Python の入門書は多くありますが, 数値計算やリストの扱いで飽きてしまう人が多く, 読み切れる人は少ないでしょう. そこで本稿では機械学習を題材にして, Python の基礎を学ぶことを目的とします. 具体的には, 機械学習の基本的な概念やアルゴリズムを学びながら, Python の文法やデータ構造を理解していきます. これにより, 最速で Python の基礎を習得し, 機械学習の実装に取り組むことができるようになります.

1.1　早速実践機械学習

機械学習とは、コンピュータがデータから学習し、予測や分類を行う技術です。

1.2　Module, Package, Library

Python では、機械学習のための多くのライブラリが用意されています。代表的なものには、NumPy、Pandas、Scikit-learn、TensorFlow、PyTorch などがあります。import numpy as np という表現は、NumPy というライブラリを np という名前でインポートすることを意味します。これにより、NumPy の機能を np という短い名前で使用できるようになります。すなわち, import module 名 as あだ名 という形で import するのですね. それではここで, module を実際に作ってみましょう. 例えば

module の基本的な構成がわかると from module 名 import 関数名 as あだ名という形も用意に理解できるようになります. これはつまり, from filename import function name as あだ名 が可能ということを意味します.

1.3　機械学習の種類

機械学習には、教師あり学習、教師なし学習、強化学習などの種類があります。

1.4　機械学習の応用

機械学習は、画像認識、自然言語処理、音声認識など、様々な分野で応用されています。

2　機械学習のアルゴリズム

機械学習のアルゴリズムには、回帰分析、決定木、ニューラルネットワークなどがあります。

2.1　回帰分析

回帰分析は、数値データの予測に使用される手法です

2.2　決定木

決定木は、データを分類するためのツリー構造のモデルです。

2.3　ニューラルネットワーク, CNN

ニューラルネットワークは、人間の脳の構造を模したモデルで、深層学習に使用されます。

付録 A　Git によるバージョン管理とプロジェクト構成

A.1　Git の基本概念

Git は分散型バージョン管理システムで、ソースコードの変更履歴を追跡・管理するためのツールです。機械学習プロジェクトでは、コードの変更履歴を管理し、実験結果を再現可能にするために重要な役割を果たします。

A.2　プロジェクト構造とモジュール化

大規模な機械学習プロジェクトでは、以下のような構造でコードを整理することが推奨されます：

```
MyProject/
├── src/                    # ソースコード
│   ├── models/             # モデル定義
│   ├── data/               # データ処理
│   ├── train/              # 学習スクリプト
│   ├── analysis/           # 分析ツール
│   └── visualization/      # 可視化
├── exefile/                # 実行ファイル
├── models/                 # 保存されたモデル
├── results/                # 実験結果
└── data/                   # データセット
```

A.3　Python のインポートシステム

異なるディレクトリのモジュールをインポートする際は、パスの設定が重要です：

```python
import sys
import os

# プロジェクトルートをパスに追加
sys.path.append(os.path.join(os.path.dirname(__file__), '..', 'src'))

# モジュールのインポート
from train.mn_cnn_train import train_model
from analysis.mn_cnn_lossf import analyze_loss_landscape
from visualization.mn_cnn_plots import plot_training_results
```

A.4　Git の基本操作

A.4.1　全ファイルの追加とコミット

プロジェクトの全ファイルを Git に追加する基本的な手順：

```
# 現在の状態を確認
git status

# 全ファイルを追加
git add .
# または
git add -A

# コミット
git commit -m "MNIST CNN プロジェクト初期実装"

# リモートリポジトリにプッシュ
git push
```

A.4.2　.gitignore ファイルの活用

GitHub にアップロードしたくないファイル（大きなデータセットやモデルファイル）は、.gitignore ファイルで除外します：

```
# Python関連
__pycache__/
*.pyc
*.pyo
.Python

# 機械学習関連
models/*.pt
models/*.pth
data/
results/
*.pkl

# IDE設定
.vscode/
.idea/

# OS関連
.DS_Store
Thumbs.db
```

A.5　改行コードの問題

異なる OS 間でプロジェクトを共有する際、改行コードの違いによる警告が表示されることがあります：

```
warning: in the working copy of 'file.py',
LF will be replaced by CRLF the next time Git touches it
```

この警告は以下の方法で対処できます：

```
# 改行コード自動変換を無効化
git config core.autocrlf false

# またはグローバル設定
git config --global core.autocrlf false
```

A.6　ローカル Git と GitHub の使い分け

- **ローカル Git**: 全ての変更履歴、大きなファイル、個人的な実験結果を管理
- **GitHub**: ソースコードのみを公開、データセットやモデルファイルは除外

この使い分けにより、コードの共有と個人データの保護を両立できます。

A.7　環境管理の将来展望

A.7.1　Docker による環境コンテナ化

長期的な開発では、Docker による環境のコンテナ化が重要になります：

```
# Dockerfile例
FROM python:3.11-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["python", "exefile/mn_cnn_main.py"]
```

A.7.2　段階的な環境管理戦略

1. **学習期**: Git + .gitignore による基本管理
2. **中級期**: Docker 導入による環境統一
3. **上級期**: Kubernetes + CI/CD による本格運用

この段階的アプローチにより、学習コストを最小化しながら本格的な機械学習システムへの発展が可能になります。