情報工学実験4：データマイニング班

(week 1) 開発スタイルと実験で使う環境

1. UNIX哲学
2. 実験で使う環境
3. （アジャイル）

実験ページ: http://ie.u-ryukyu.ac.jp/~tnal/2019/info3/dm/

情報工学実験4：データマイニング班

(week 1) 開発スタイルと実験で使う環境

1. UNIX哲学
2. 実験で使う環境
3. （アジャイル）

「UNIXという考え方」より

Mike Gancarz著, 芳尾桂監訳, オーム社, 2001

-> Python !!

- 定理1: 小さいものは美しい
- 定理2: 一つのプログラムには一つのことをうまくやらせる
- 定理3: できるだけ早く試作する
- 定理4: 効率より移植性を優先する
- 定理5: 数値データはASCIIフラットファイルに保存する
- 定理6: ソフトウェアを梃子として使う
- 定理7: シェルスクリプトによって梃子の効果と移植性を高める

- 定理8: 過度の対話的インタフェースを避ける
- 定理9: すべてのプログラムをフィルタとして設計する
- +α
  - 好みに応じて環境を調整できるようにする
  - 並行して考える
  - 部分の和は全体よりも大きい
  - 90%の解を目指す
  - 階層的に考える

Unix哲学 (1) KISSの原則

- Keep It Simple, Stupid!
  - 小さく作り、組み合わせる。
    - 組み合わしやすいようにクラス/モジュール/関数のI/Oを設計。
    - 各部品は一つのことを確実に行うように完成度を高める。
  - 各部品をテストする。
    - 100%を目指す必要はない。まずは関数毎に単体テストを1件は試そう。それ以上は必要に応じてぐらいの気持ちで。
    - 検証／再現性を意識すること。

Q: 例えばどういうI/Oだと組み合わせやすい？

Q: 完成度とは？どうやれば完成度を高められる？

Q: 単体テストとは？他にどんなテストがある？

Q: 乱数を使う場合にはどう再現したら良い？

Unix哲学 (2) プロトタイプ+道具

- プロトタイプを素早く作って、検証・改善。
  - 類似: Demo or Die, テスト駆動開発。
  - 見える形でアウトプットし、修正し続けることで完成度を高める。
  - プロトタイプ実装・検証・改善をスムーズに進めるために道具を使う。
    - バージョン管理。
    - ワークフロー(再現性)管理。
    - 可視化。
    - ディレクトリ構造。
- 必要に応じて最適化。

Q: 例えばどういう可視化の方法が考えられるか？　向き・不向き等の特性はあるだろうか？

Q: どんなワークフローがあるか？それをどう管理したら良いか？

Q: オープンソースとして公開されてるプロジェクトのディレクトリ構造はどうなっているだろうか？

Q: ここでいう最適化とは？どうやれば最適化しやすくなるだろう？

（補足スライド）研究段階コーディング

研究段階

- e.g., ゴールやアプローチが曖昧な状況。
- 理想：変化を見据えた実装。
- 現実：どのような変化にも柔軟に対応できるシステム構築は困難。
- 代替手段：プロトタイプ（動くデモ）を素早く作って、検証・改善を繰り返し、ゴールやアプローチを明確にすることを優先する。この時点ではそれ以外の部分には目をつぶる。
  - 見た目、実行速度、メモリ消費量、、

研究段階でも踏まえたい項目

- 構造化
  - KISS原則
    - 正しい分割になっていなくて良い。テストしやすさの観点で分割しよう。
    - 分解し、結合してみることで、後からベターな分解方法を再検討することができる。
  - 道具を使う（既にモジュールがあればそれを利用する）
    - Scikit-learn, Numpy, Pandas,,,
- テスト
- 可視化（デモ）
  - 処理結果が想定どおりであることを確認できること、もしくは結果を俯瞰しやすくすること。

（補足スライド）最終段階コーディング

最終段階

- ゴールやアプローチが明確に決まっている状況。
  - リファクタリング。
    - テストを通すだけのコードから、整理されたコードへ。
  - 実行速度が不十分ならば、最適化しよう。
- モニタリングにより要因を特定。
  - リソース使用率、実行速度、DB、特定SQLクエリ、File I/O、、、

モニタリングして最適化

- 必要に応じて、、、
  - （一部/全部を）別言語で書き直す。
  - リソースを見直す。
    - CPU/Memory/Storage等をリプレースするだけで実行速度等の要求仕様を満たせるなら、それで済ますのも手。
  - 並列and/or分散処理。
  - ロジックの見直し。

（補足スライド）可視化

- 数値化／テキスト化／グラフ化
- 表
  - 表計算ソフト, HTML
- 線グラフ、円グラフ, 棒グラフ, 帯グラフ,散布図, 箱ひげ図
- 2次元、3次元軸上のグラフ
  - gnuplot, matplotlib, google charts
- データ構造やグラフ理論の「ノードとエッジのあるグラフ」
  - Graphviz (dot), D3.js, draw.io
- ヒストグラム
  - matplotlib, Google charts
- レーダーチャート
  - Google charts
- デモ
  - HTML, JavaScript, CSS
- コード／インタラクティブ
  - IPython Notebook -> Jupyter

Q: 意思決定とは？例えばどういう意思決定があるか？

どう可視化することで意思決定しやすくなるかを考え、作図できるようになろう！

情報工学実験4：データマイニング班

(week 1) 開発スタイルと実験で使う環境

1. UNIX哲学
2. **実験で使う環境**
3. （アジャイル）

実験で使う環境(推奨/目安)

- プログラミング言語
  - Python 3.6以上
- 補足:
  - Mac OS X 10.13 標準ではPython 2.x系列がインストール済み。
  - Python 2.x系と3.x系は互換性が無い。
  - Python 2系は開発終了(->少し延長)。既に2系への対応をしていないライブラリも出始めている。
  - Pyhon 3.x系は、全ての文字列がデフォルトでUnicodeになり、文字列処理がしやすくなった。
- 参考: Python 2 と Python 3 のどちらを使って開発すべき？
  - http://nyagao.hateblo.jp/entry/2014/03/25/210415

- 開発環境
  - PyCharm or Editors(Emacs, Vim,,,)
  - VS Code? (Visual Studio Code)
- バージョン管理
  - 推奨: Git + GitHub
- Python仮想環境
  - 推奨: pyenv + anaconda
- 機械学習パッケージ
  - Scikit-learn
    - Python (NumPy, SciPy, matplotlib)
    - 上記が動作するなら環境構築方法は自由。
- （自然言語処理）＊オマケ
  - Mecab, NLTK
  - 単体テスト
    - pytest, **doctest**

情報工学実験4：データマイニング班

(week 1) 開発スタイルと実験で使う環境

1. UNIX哲学
2. 実験で使う環境
3. （アジャイル）

アジャイルソフトウェア開発宣言

http://agilemanifesto.org/iso/ja/

- 前提
  - 事務的な作業と異なり、新しいモノを生み出す作業の場合、その「新しいモノ」は仕様を厳密に確定することは困難。
  - ***必ず***何かしらの仕様変更が入る。
- アジャイルソフトウェア開発宣言
  - 左記のことがらに価値があることを認めながらも、私たちは右記のことがらにより価値をおく。
    - プロセスやツールよりも個人と対話を、
    - 包括的なドキュメントよりも動くソフトウェアを、
    - 契約交渉よりも顧客との協調を、
    - 計画に従うことよりも変化への対応を、
  - 「動くソフトウェア、変化への対応」のために
    - UNIX哲学（KISS＋プロトタイプ＋道具）
    - YAGNIの原則（実際に必要になるまでは機能を追加しない）