研究者流
コーディングの極意

言語処理学会第19回年次大会 (NLP2013)
チュートリアル資料（岡崎担当分）

岡崎　直観
東北大学大学院情報科学研究科
okazaki at ecei.tohoku.ac.jp
http://www.chokkan.org/
@chokkanorg

研究におけるコーディングの極意？

- 今回のチュートリアルをきっかけにサーベイ
  - ソフトウェアエンジニア向けの指南書は存在
  - でも，研究者向けの資料は数少ない
- 自分が修士課程の頃は完全に我流だった
  - 複数文書自動要約のプログラムをすべてC++で実装
  - *NIXを使うスキルはなく，すべてWindows上で実行
  - 今から考えると，無駄だらけの実験作法だった
- ほとんどの大学では実験の講義があるが…
  - 研究のためのコーディング作法は教えてくれない

繰り返される残念な光景

- 論文の締切前日に実験結果の間違いを発見
- コードが複雑すぎて自分でも解読不能
- 途中結果を捨てていて，コードの検証不能
- 実験結果すらストレージに残っていない
- 実験結果を確認する術を知らない
- コードの整理に数日を浪費する
- 正しい確証がないコードを何日も動かしている
- 書いたコードが誰にも使われず，闇の中へ
- …

研究におけるコーディングの位置づけ

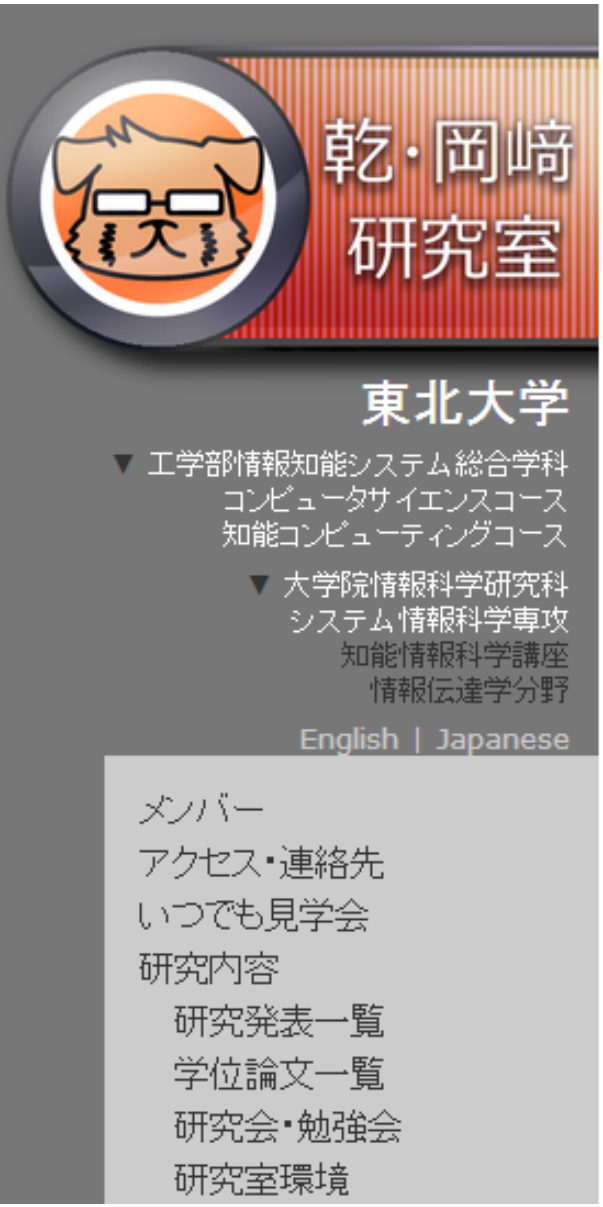
- 研究のアイディアがコンピュータ上で期待通り実現するか確認する
- 研究の成果物は知見（論文）であって，作ったソフトウェアそのものではない [Sutton 2012]
- 初めのうちは論文を出すことに全力を注げ
  - 業績を稼げという意味ではない
  - 「研究する」とはどういうことか，一通り体験してみないと分からない

本発表のねらい

- 研究を加速するコーディングの極意とは
- コーディングの極意の教え方・学び方
- （自分の作ったソフトウェアの公開について）

研究を加速するコーディング

- 研究はスピード感が大切
  - 成功回数 = 試行回数 × 成功率 [賀沢 2012]
  - 「試行回数を増やす」≒「コーディングを加速」
- ソフトウェアエンジニアの仕事とはかなり違う
  - 対象: 仕様が決まっている vs 解かれていない問題
  - 目的: 利益を得る vs アイディア（実験結果）を検証
- 「言語処理100本ノック」を開発
  - コーディングの極意を実践しながら伝授



FrontPage / 言語処理100本ノック

もくじ †

http://www.cl.ecei.tohoku.ac.jp/index.php?言語処理100本ノック

第1セット: テキスト処理の基礎

標準入力からタブ区切り形式のテキスト(address.txt)を読み込み，以下の内容を標準出力に書き出すプログラムを実装せよ．また，ヒントに挙げたツールを用いても，同じ内容が得られることを確認せよ．

- (1) 行数をカウントしたもの．確認にはwcコマンドを用いよ．
- (2) タブ1文字につきスペース1文字に置換したもの．確認にはsedコマンド，trコマンド，もしくはexpandコマンドを用いよ．
- (3) 各行の1列目だけを抜き出したものをcol1.txtに，2列目だけを抜き出したものをcol2.txtとしてファイルに保存せよ．確認にはcutコマンドを用いよ．
- (4) (3)で作ったcol1.txtとcol2.txtを結合し，元のタブ区切りテキストを復元したもの．確認にはpasteコマンドを用いよ．
- (5) 自然数Nをコマンドライン引数にとり，入力のうち先頭のN行だけ．確認にはheadコマンドを用いよ．
- (6) 自然数Nをコマンドライン引数にとり，入力のうち末尾のN行だけ．確認にはtailコマンドを用いよ．
- (7) 1コラム目の文字列の異なり数(種類数)．確認にはcut, sort, uniq, wcコマンドを用いよ．
- (8) 各行を2コラム目の辞書順にソートしたもの(注意: 各行の内容は変更せずに並び替えよ)．確認にはsortコマンドを用いよ(この問題は結果が合わなくてもよい)．
- (9) 各行を2コラム目，1コラム目の優先順位で辞書の逆順ソートしたもの(注意: 各行の内容は変更せずに並び替えよ)．確認にはsortコマンドを用いよ(この問題は結果が合わなくてもよい)．
- (10) 各行の2コラム目の文字列の出現頻度を求め，出現頻度の高い順に並べよ．ただし，(3)で作成したプログラムの出力(col2.txt)を読み込むプログラムとして実装せよ．確認にはcut, uniq, sortコマンドを用いよ．

第2セット: 正規表現・日本語の扱い

標準入力からテキスト(tweets.txt.gz)を読み込み，以下の処理を行うプログラムを実装せよ．

http://www.cl.ecei.tohoku.ac.jp/index.php?言語処理100本ノック

言語処理100本ノックについて

- 佐藤理史先生が作られた問題集がヒント
- ただの問題集ではない
  - コーディングの極意（技術と作法）を自然に身につける
  - 自然言語処理の研究に役立つノウハウを体験させる
  - 実用的で，楽しくなるような題材を採用
  - 自然言語処理に詳しいエンジニアを育成できる
  - 研究室の学生のスキルをアピールするために公開
- NLP内外から意外に大きな反響
  - 145 RTs, 227 favs, 744 Hatena bookmarks

プログラミング基礎勉強会

Home User: §1 §2 §3 §4 §5 §6 §7 §8 §9 §10 All

001 002 003 004 005 006 007 008 009 010

001: Count the number of lines

hiro-sz (.py)

```
# output the number of lines of address.txt

import sys

print len(sys.stdin.readlines())
```

takahiro (.py)

```
import sys
f = open(sys.argv[1])

print len(f.readlines())
```

aki-s (aki-s.py)

```
#-*- coding:utf-8 -*-
import sys
f = open(sys.argv[1], 'r')
gc=0
for line in f:
```

Home User: §1 §2 §3 §4 §5 §6 §7 §8 §9 §10 All

041 042 043 044 045 046 047 048 049 050

```
import sys
import mecab

for S in mecab.readiter(sys.stdin):
    for token in S:
        print '%(surface)s\t%(base)s\t%(pos)s\t%(pos1)s' % token
    print
```

okazaki (mecab.py)

```
def readiter(fi):
    S = []
    for line in fi:
        line = line.strip('\n')
        if line == 'EOS':
            yield S
            S = []
        else:
            fields = line.split('\t')
            values = fields[1].split(',')
            surface = fields[0]
            base = values[6] if values[6] != '*' else surface
            pos = values[0]
            pos1 = values[1]
            S.append(
                {'surface': surface, 'base': base, 'pos': pos, 'pos1': pos1}
                )
```

「tailコマンドを作れ」(問題6) の議論

```
import sys

N = int(sys.argv[1])
val = -1
lines = sys.stdin.readlines()
while val != -N-1 :
    sys.stdout.write(lines[val])
    val += -1
```

valの初期化とループは近い方がよい

入力データが大きいとメモリが不足する

for文を使うべき

リングバッファを使うべき

```
val -= 1
```

絵文字らしき文字列を抽出せよ：自分の答え

```
import sys
import re

pattern = re.compile(
  ur'[^  0-9０-９a-zA-ZＡ-Ｚａ-ｚ一-龠ぁ-んァ-ヴ]+$')
for line in sys.stdin:
  line = line.strip('¥n')
  for m in pattern.finditer(unicode(line)):
      sys.stdout.write('%s¥n' % m.group(0))
```

絵文字らしき文字列を抽出せよ：絵文字を真面目に解釈

```
import sys
import re

p = re.compile(
u"[♥✈☺☻☹♪♀☆✉☠✓♂☚♘☯☾☝♖✻†☄♝✺♀✂✍♕✵★✇♲✕♨❦☁✌
   ♕❁☪☂☛❀☭☞♞✿☮☼♬☑♠☎♫♤☃]+")

for line in sys.stdin:
    a = p.findall(unicode(line,'utf-8'))
    if a:
        for b in a:
            print b
```

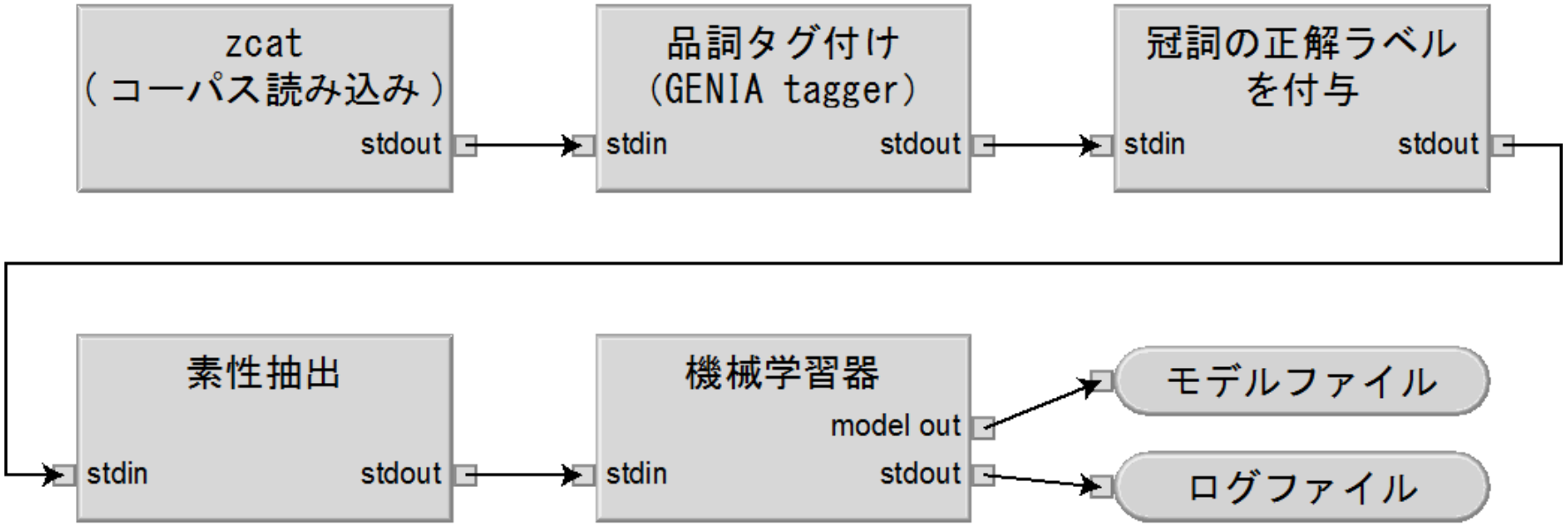
100本ノックの背後にある極意

1. 小さい処理に分解し結合せよ
2. 道具を使え
3. 自分を過信せず検証せよ
4. 常に検証に備えよ
5. 研究成果を可視化せよ
6. 最適化・整理は完成してから
7. 論文を書いたらコードを整理せよ

- 一つ一つは些細なことでも，全体で大きな力に！

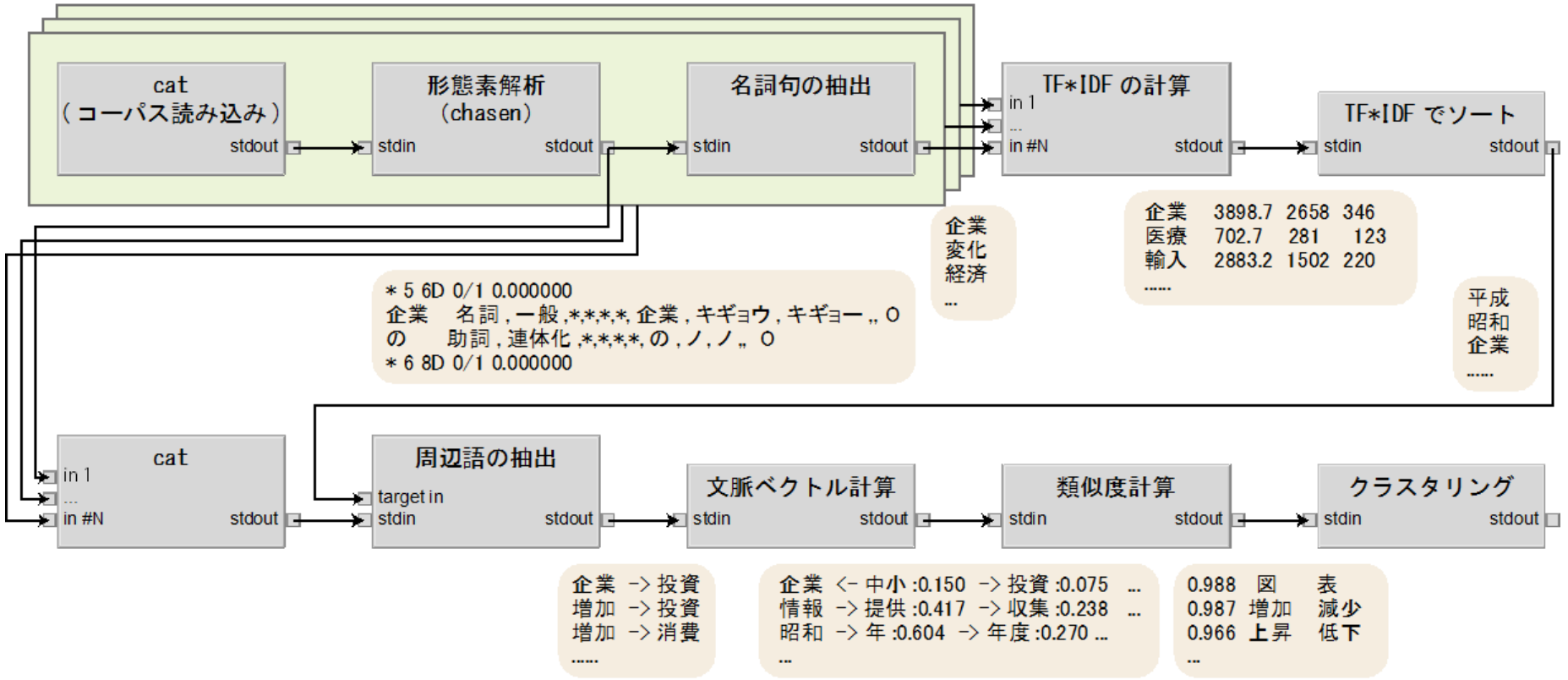
1. 小さい処理に分解し結合せよ

- タスクを小さいプログラムに分解する
- プログラム間が連携するように入出力を規定



冠詞の予測モデルの構築（第８セット）

やや複雑な分解例:
名詞句のクラスタリング（第7セット）

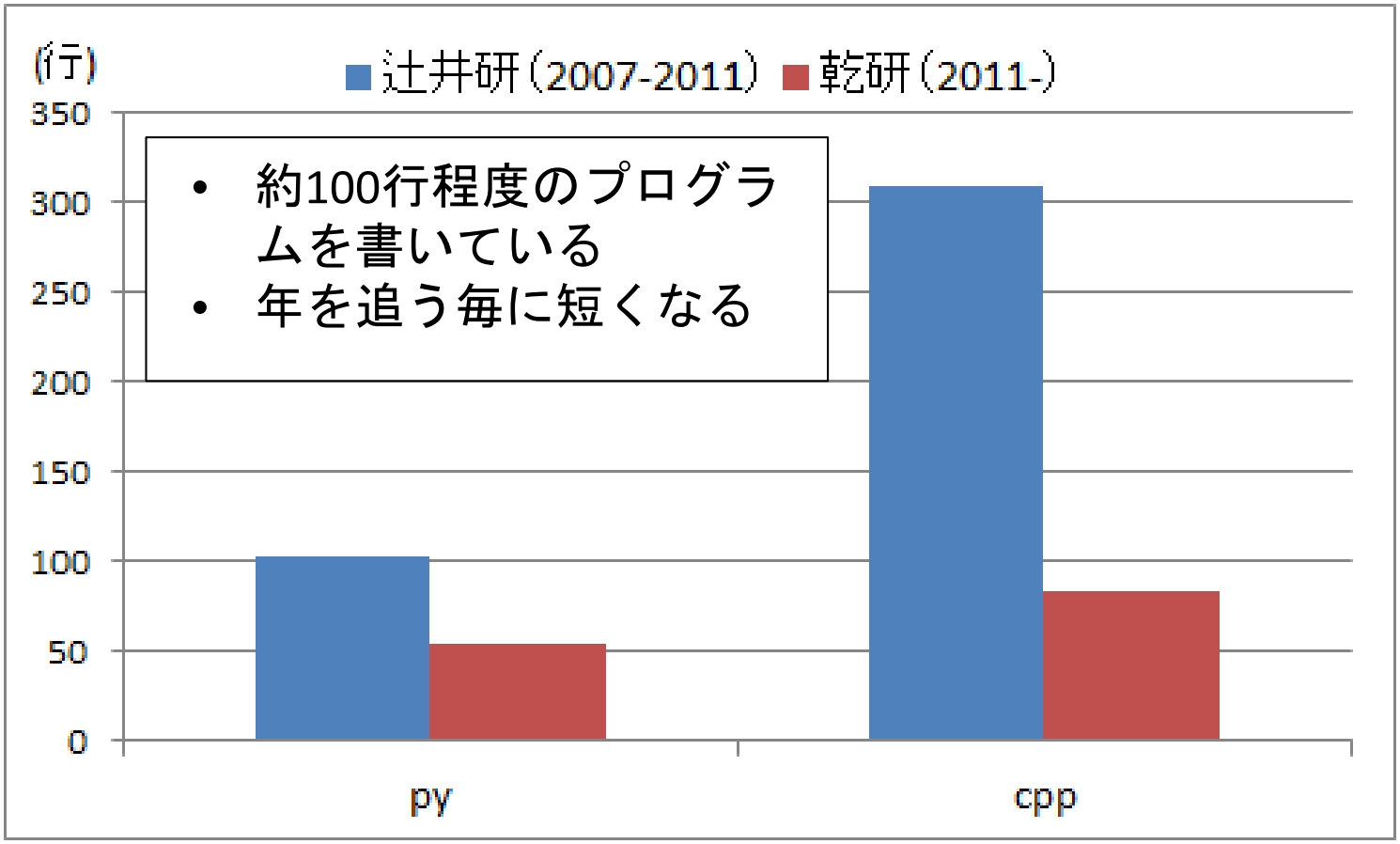


100本ノックでタスクの分解の仕方も学べる

McIlroyのUNIX哲学 [Salus+ 1994]

- ひとつの事を確実に行うプログラムを書け
  - 新しい事をやるときは，古いプログラムに機能を追加するのではなく，新しいプログラムを書け
- 他のプログラムと連携するプログラムを書け
  - 全てのプログラムが別の（未知の）プログラムの入力になることを想定せよ
- 普遍的なインタフェースであるテキスト・ストリームを扱うプログラムを書け

研究用のプログラムの平均行数（岡崎）



2. 道具を使え

source.txt中の名詞の出現頻度を求めよ.

```
cat source.txt | ¥
  mecab | ¥                  # 形態素解析
  grep $'¥t名詞' | ¥         # 名詞のみを通過
  cut -f1 | ¥                # 表層形を取り出す
  sort | uniq -c | ¥         # 頻度を計測
  sort -nr                   # 頻度順にソート
```

１行（80バイト未満）で実現可能

2. 道具を使え

- 道具とは？
  - UNIXコマンド（sort, cut, grep, make, …）
  - ツール（cabocha, gnuplot, SQL, lucene, …）
  - ライブラリ（json, nltk, matplotlib, …）
  - アルゴリズム（二分探索，接尾辞木，…）

- 自分のコードよりも 実績のある道具
- 労力を省いて研究の時間を大切に
- 良い道具を普段から調査・収集せよ

3. 自分を過信せず検証せよ

- “Computers are good at following instructions, but not at reading your mind” (Donald Knuth)

- 実験結果（論文）は信頼性が大切

- 少しの時間でかなりの無駄を省けることも
  - コードを少し書いては動作チェックというプロセスを繰り返すことで，プログラムを大きくしていく
  - 大量のデータで動作させる前に，少量のデータで素早くプログラムを動かし，検証する

コードを少しずつ書く (1/4)

```
def readiter(fi):
    S = []
    for line in fi:
        line = line.strip('¥n')
        if line:
            S.append(dict(zip(('w', 'pos', 'chunk'), line.split(' '))))
        else:
            yield S
            S = []
```

←CoNLL 2000のデータを
文毎に読み込む関数

（チャンキングの素性抽出器を作る）

コードを少しずつ書く (2/4)

```
def readiter(fi):
    S = []
    for line in fi:
        line = line.strip('¥n')
        if line:
            S.append(dict(zip(('w', 'pos', 'chunk'), line.split(' '))))
        else:
            yield S
            S = []

if __name__ == '__main__':
    import sys
    for S in readiter(sys.stdin):
        for token in S:
            sys.stdout.write('%(w)s %(pos)s %(chunk)s¥n' % token)
        sys.stdout.write('¥n')
```

←readiter関数の動作チェック

入力

```
Rockwell NNP B-NP
International NNP I-NP
Corp. NNP I-NP
's POS B-NP
Tulsa NNP I-NP
unit NN I-NP
said VBD B-VP
```

完全に一致

```
Rockwell NNP B-NP
International NNP I-NP
Corp. NNP I-NP
's POS B-NP
Tulsa NNP I-NP
unit NN I-NP
said VBD B-VP
```

出力

コードを少しずつ書く (3/4)

```
def readiter(fi):
    S = []
    for line in fi:
        line = line.strip('¥n')
        if line:
            S.append(dict(zip(('w', 'pos', 'chunk'), line.split(' '))))
        else:
            yield S
            S = []

def observation(v):
    # An initial uppercase letter.
    v['iu'] = v['w'][0].isupper()
```

←新しい素性（先頭が大文字かどうかチェックする）を追加

コードを少しずつ書く (4/4)

```
def readiter(fi):
    S = []
    for line in fi:
        line = line.strip('¥n')
        if line:
            S.append(dict(zip(('w', 'pos', 'chunk'), line.split(' '))))
        else:
            yield S
            S = []

def observation(v):
    # An initial uppercase letter.
    v['iu'] = v['w'][0].isupper()

if __name__ == '__main__':
    import sys
    for S in readiter(sys.stdin):
        for token in S:
            observation(token)
            sys.stdout.write('%(w)s %(pos)s %(iu)s %(chunk)s¥n' % token)
        sys.stdout.write('¥n')
```

```
Rockwell NNP True B-NP
International NNP True I-NP
Corp. NNP True I-NP
's POS False B-NP
Tulsa NNP True I-NP
unit NN False I-NP
said VBD False B-VP
```

出力

←追加したコードの動作チェック

4. 常に検証に備えよ

- 検証が億劫にならないように工夫せよ
  - デバッグ・検証には時間がかかる
- 実験結果の再現性を常に意識
  - ミーティングや論文をまとめる上で重要
- 実験に関する以下の情報を記録・保管・整理
  - 仮定（入力データ，前処理，パラメータ）
  - 手順（プログラムの組み合わせ方）
  - 最終成果物（出力データ，精度）
  - 中間生成物（実験結果の検証に有用）
- ディレクトリやファイル名も工夫 [Eslami 2012]

ディレクトリ構造の例

```
npclust
|
+-------bin
|
+-------data
|
+-------work
|        |
|        +----  ...
|
+-------result
```

研究に関する設定（.bashrcなど）や，実験を動かすスクリプト等を配置

実験用のプログラム配置

実験のデータ（加工前）を配置

実験による中間生成物を配置
実験設定・パラメータ毎にサブディレクトリを作ることも

実験結果や評価結果を配置

検証に役立つツール

- バージョン管理（git, mercurial, subversion）
  - いつでも昔のプログラムや実験設定に戻れる
- ワークフロー管理（Makefile, Ant, スクリプト）
  - 実験をいつでもやり直せるように
- 可視化ツール（次スライド参照）
  - 実験結果の問題点に気づくように
- ソフトウェアテスト（工藤さんのトーク）

5. 研究成果を可視化せよ

- 実験結果を可視化し，検証しやすくする
  - ミーティング時に，何をやったのか一目瞭然
  - 実験結果を他人に説明し，フィードバックをもらう

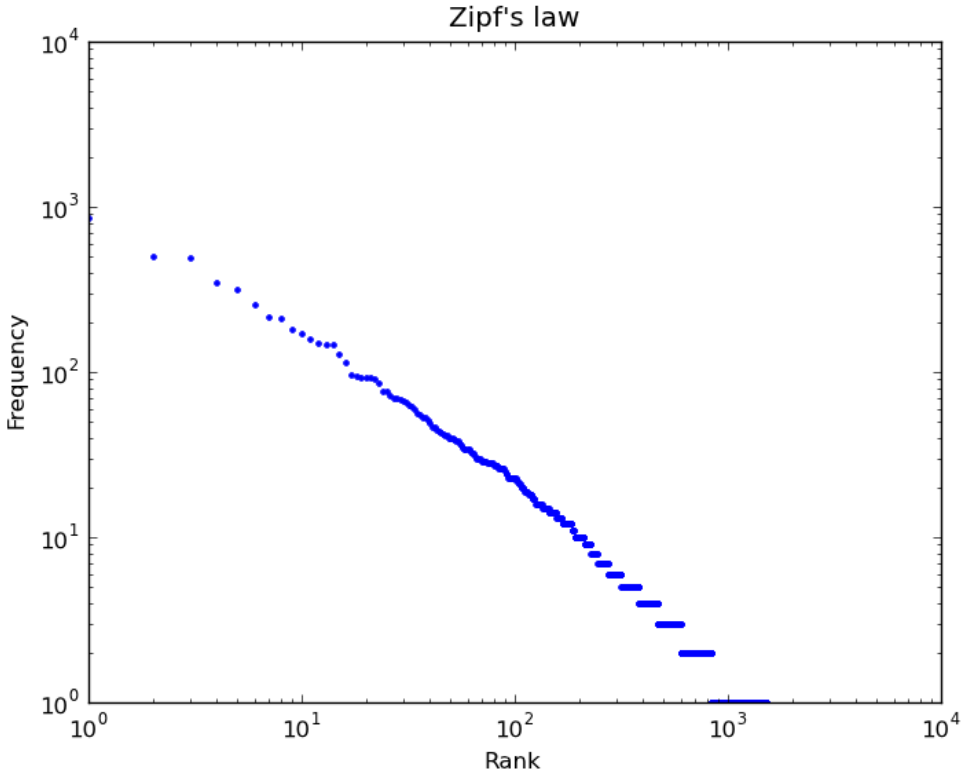
- 【高度】デモを作る
  - 「言葉が通じない」相手にも伝わる
  - そのまま「営業」ができる

可視化ツール

- 表（Microsoft Excel, HTML）
- チャート（gnuplot, matplotlib, Google Chart）
- グラフ（graphviz, Gephi, D3.js）
- テキスト（Microsoft Excel, HTML, brat）
- コード（google-code-prettify, Pygments）
- デモ（HTML, JavaScript, CSS, ...）

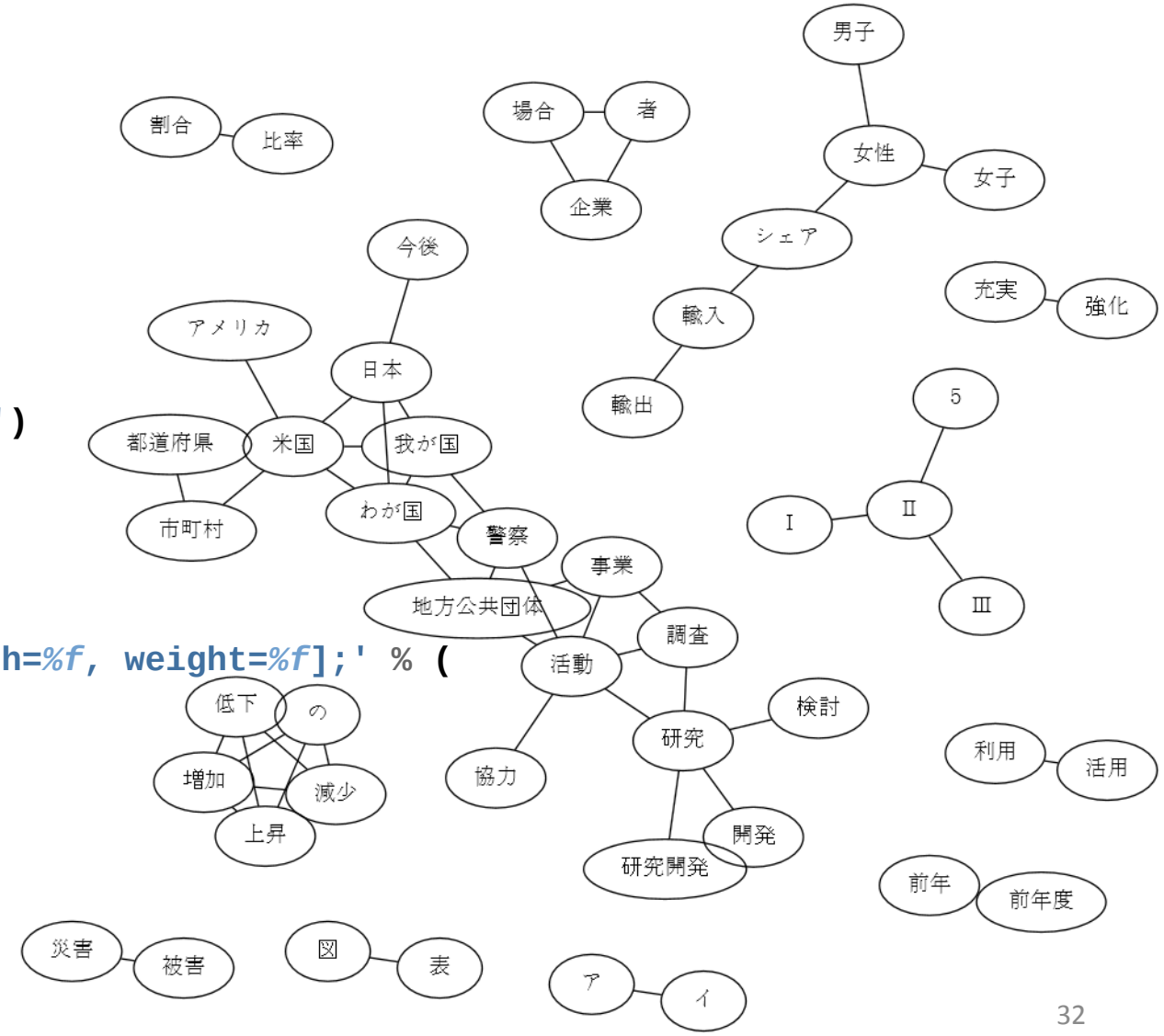
可視化の例（問題50）

```
import sys
import matplotlib; matplotlib.use('Agg')
import matplotlib.pyplot as plt

y = []
for line in sys.stdin:
    line = line.strip('¥n')
    fields = line.split('¥t')
    y.append(float(fields[0]))

y = sorted(y, reverse=True)

plt.loglog(y, '.')
plt.xlabel('Rank')
plt.ylabel('Frequency')
plt.title("Zipf's law")
plt.savefig("zipf.png")
```



可視化の例（問題69: クラスタリング）

```
import sys

print 'graph G {'
print '  size="20,20";'
for line in sys.stdin:
  line = line.strip('¥n')
  fields = line.split('¥t')
  sim = float(fields[0])
  if 0.6 <= sim:
  length = 1 - sim
  weight = sim
  print '  %s -- %s [length=%f, weight=%f];' % (
    fields[1],
    fields[2],
    length,
    weight
    )
print '}'
```



6. 最適化・整理は完成してから

- “Prototype before polishing. Get it working before you optimize it.” [Raymond 2003]
- “Programmer time is expensive; conserve it in preference to machine time.” [Raymond 2003]
- “Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it.” (Brian Kernighan)
- “Make it work. Make it right. Make it fast.” (Kent Beck)

中途半端な最適化は諸悪の根源 (Donald Knuth)

6. なぜ完成を優先させるのか？

とりあえず全体を作る（Make it work）
・特殊な条件だったり，性能が低くても構わない

正しくなるように完成させる（Make it right）
・結果の検証，性能の改善，一般性の確保

研究ではここまで到達できないことも多い

リファクタリング・高速化（Make it fast）
・全体を知らずに大域的な最適化・整理をするのは無理

7. 論文書いたらコードを整理せよ

- これまでは自分のためにコードを書いていた
- 論文が色々な人に読まれると，ソフトウェアやデータが欲しいとの要望が届くようになる
- 他人に読まれる・使われることを想定してコードを書き直し・整理し，後生に伝えよ
  - ソフトウェア資産や開発の軌跡
  - 使い方やインストール方法など，最低限のドキュメント
  - 自分の手から離すことが大切

ソフトウェアを公開することのメリット

- 自分の研究成果を使って貰える
  - 論文の被引用回数が増える
  - 後発の研究が実験しやすい
- （自分が意図せず）自分のことを知って貰える
  - 海外の著名な方の論文・書籍に載っていることを，周りの人から偶然教えてもらう
- 履歴書に書ける（インターン，学振，奨学金返還免除，大学を含めた就職活動）
  - githubアカウントは就活必携アイテム？
- 他の人に使って貰える喜び

まとめ

- "Keep It Simple, Stupid!" (KISSの法則)
  - 単純な問題に分け，正確に解く力（知識・経験・作法）
- プロトタイプを素早く作って，検証・改善せよ
  - 最適化や整理は後回し．早く作らないと忘れちゃう
- コーディングも研究も，とりあえずひと通り体験してから自分なりの（全体）最適化をすればよい
  - 何事においても同じこと
  - 中途半端で考え込んでもダメ！

参考文献

Kenneth W. Church. Unix for Poets.
http://www.stanford.edu/class/cs124/kwc-unix-for-poets.pdf

Mark Dredze, Hanna M. Wallach. 2012. How to be a Successful PhD Student (in Computer Science (in NLP/ML)).
http://people.cs.umass.edu/~wallach/how_to_be_a_successful_phd_student.pdf

S. M. Ali Eslami. 2012. Patterns for Research in Machine Learning.
http://arkitus.com/PRML/

M. D. McIlroy, E. N. Pinson, B. A. Tague. 1978. UNIX Time-Sharing System: Forward. *The Bell System Technical Journal*, Vol. 57, Issue 6, pp. 1899-1904.

Eric S. Raymond. 2003. *The Art of UNIX Programming*. Addison-Wesley.

Peter H. Salus. 1994. *A Quarter-Century of Unix*. Addison-Wesley.

Charles Sutton. 2012. Principles of Research Code.
http://www.theexclusive.org/2012/08/principles-of-research-code.html

賀沢秀人. 2012. 成功の方程式. 自然言語処理, Vol. 19, No. 1, pp. 1–2.