第8期（2017年9月～2018年8月）
クラウド・テクノロジー活用部会
活動成果報告

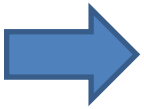
2018年10月1日

先端IT活用推進コンソーシアム
クラウド・テクノロジー活用部会
リーダー　荒本道隆

2017年度の主な活動内容 - 1

- 開催実績：月例部会11 回、参加者77 名（毎回5～12人）
- 情報交換（イベントへの参加、ニュース、その他）
- 参加者が講師となり、内部勉強会を開催（敬称略）

強化学習（戸田）
遺伝的アルゴリズム（松井）



オープンラボに発展
2018年2月24日

量子コンピュータと暗号化アルゴリズムについて（宮地）
仮想通貨の話（宮地）
車のIoT の話（宮地）
デジタル社会に合わせた本人確認方法の必要性について（宮地）
量子アニーリングとは（宮地）
セキュリティ概要「攻撃者の動機編」（荒本）

- 気象庁XML公開APIの運用 http://api.aitc.jp/
  - 運用期間を2021年8月まで延長

2017年度の主な活動内容 - 2

- 持ち回り講師で、書籍を輪読。実際に動かしてもみる

『仕事ではじめる機械学習』

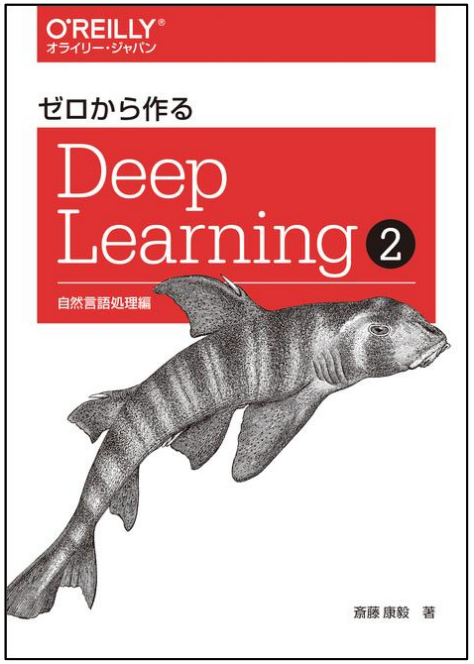
第１章：機械学習プロジェクトのはじめ方（荒本）
第２章：機械学習で何ができる？（上村）
第３章：学習結果を評価しよう（簑島）
第４章：システムに機械学習を組み込む（宮地）
第５章：学習のためのリソースを収集しよう（宮地）
第６章：効果検証（大越）
第７章：映画の推薦システムをつくる（中嶋）
第８章：Kickstarter の分析、機械学習を使わないという選択肢（北川）

気象庁XMLの取得APIのアクセス数

	REST API		SPARQL		WebSocket	
	リクエスト数	ユニークIP	リクエスト数	ユニークIP	リクエスト数	ユニークIP
2017/01	441,740	166	58	45	5,463	136
2017/02	403,027	118	83	39	1,000	126
2017/03	469,496	159	114	36	1,437	148
2017/04	428,595	289	64	39	2,173	117
2017/05	452,716	441	100	40	923	155
2017/06	461,664	124	120	53	1,100	179
2017/07	485,524	139	205	54	1,743	164
2017/08	473,130	147	433	70	98,483	187
2017/09	487,245	582	503	62	79,967	191
2017/10	414,585	779	48	27	42,216	166
2017/11	302,232	558	67	30	1,552	148
2017/12	253,900	431	53	27	546	112
2018/01	349,202	265	32	28	758	162
2018/02	341,489	348	225	32	534	134
2018/03	388,140	294	77	22	1,580	148
2018/04	357,686	328	18	17	418	128
2018/05	368,571	460	39	26	369	108
2018/06	343,283	477	49	36	521	116
2018/07	257,046	351	55	35	463	103
2018/08	247,556	379	51	37	1,921	109
2018/09	213,315	257	27	22	388	101

2018年度の活動予定

- 月１回（13:30～17:30ごろ）のミーティング
- 情報交換（イベント、ニュース、その他）
- クラウドを使った新しいサービス
- クラウド上で動作させると面白い技術
- それらを実際に動かしてみる
- 輪読候補
  - 『ゼロから作る Deep Learning ②』



オープンデータを料理する
～ 気象データを題材に ～

2018年10月1日

先端IT活用推進コンソーシアム
クラウド・テクノロジー活用部会
リーダー　荒本道隆

試用するデータの概要－１

- 気象庁がクローズドβで提供
  - 「試行提供気象データー覧」参照→
  - 案内範囲は「WXBCの会員」
  - AITCはWXBCの会員なので入手可
- データサイズ概算
  - 観測全体：52GByte
  - 予報全体：1,300GByte
  - 全てgzip圧縮済みのサイズ
- 今回、対象とするデータ
  - 地域気象観測（アメダス）

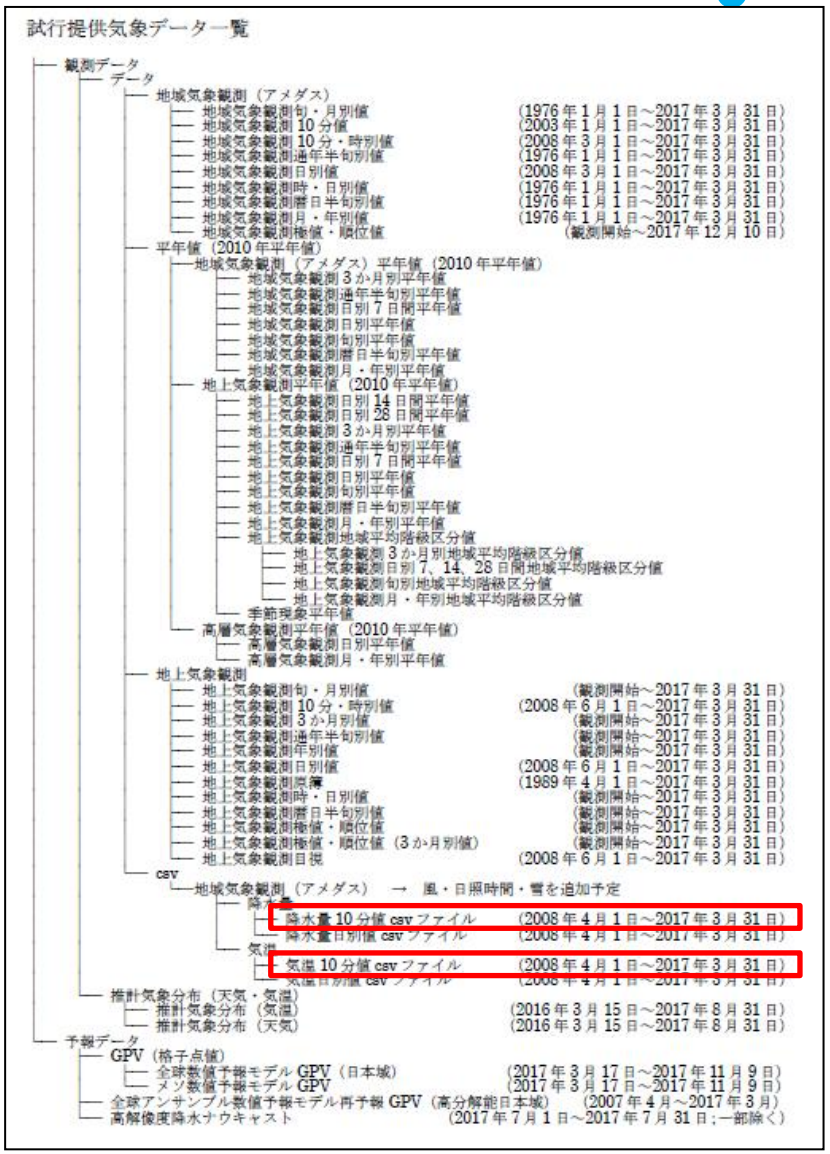
降水量と気温のCSVファイル、
10分毎、時間ごと、
2008年4月1日～2017年~~3月31日~~11月30日

/kansoku/amedas_csv/pre/10m/time

3.0GByte

/kansoku/amedas_csv/tem/10m/time

4.8GByte



試用するデータの概要：補足

- 発表資料が完成した翌朝に届いたメール

2．気象庁過去データ試用提供のデータの追加のご案内

8月10日（金）より運用を開始した、気象庁過去データ試用提供のデータの追加についてお知らせいたします。

気象庁過去データ試用提供とは、WXBC会員のみなさまにまずはお試しで気象データを利用していただくために、気象庁の過去の観測・予測データの一部を暫定的に無償で提供するものです。

今回追加するデータは、地域気象観測（アメダス）データをより扱いやすくした**CSV形式のアメダスデータ**です。昨年度から提供していたのは**降水量**と**気温**の2要素でしたが、今回は**風・日照・雪**の3要素を追加します。**既存の2要素についてもファイルを追加し、2018年7月までのデータを提供いたします。**昨年度、気象庁過去データ試用提供を利用されていた方もこれを機に今年度も利用いただければと思います。

利用にあたっては事前に以下のページ応募フォームから登録をお願いします。登録後数日以内に、事務局から利用のために必要な情報をお知らせいたします。

気象庁過去データ試用提供（平成30年度）
https://www.wxbc.jp/mypage/jmadata/

発表者によって、使用している期間が違う場合があります。

試用するデータの概要－2

- 降水量CSVフォーマット
- 気になった点
  - 品質情報って、何？
  - 「-9999」「-9999.0」の存在

（2）降水量（単位は mm）

（a）10 分値ファイル（※ 地点別ファイル、時間別ファイル共通）

各項目の間にはカンマが入り、レコード最終には改行（LF）が付く

項番	項目名	意味	表記	備考
1	amd_no	アメダス地点番号	整数	
2	sfc_no	国際地点番号	整数	該当ない場合「0」
3	yr	年	整数（西暦 4 桁）	
4	mn	月	整数（ゼロ埋め 2 桁）	
5	dy	日	整数（ゼロ埋め 2 桁）	
6	hr	時	整数（ゼロ埋め 2 桁）	
7	mi	分	整数（ゼロ埋め 2 桁）	
前 10 分間降水量				
8	pre10m_rm	品質情報	整数	「3．品質情報」参照、対象範囲は項番 9～11
9	pre10m	前 10 分間降水量	小数（小数点以下 1 桁）	品質情報が 0～2 の場合「-9999.0」
10	pre10m_ad	同附帯情報	整数	「4．附帯情報」参照
11	pre10m_sf	同有効桁数	整数	「5．有効桁数」参照
前 1 時間降水量				
12	pre1h_rm	品質情報	整数	「3．品質情報」参照、対象範囲は項番 13～15
13	pre1h	前 1 時間降水量	小数（小数点以下 1 桁）	品質情報が 0～2 の場合「-9999.0」
14	pre1h_ad	同附帯情報	整数	「4．附帯情報」参照
15	pre1h_sf	同有効桁数	整数	「5．有効桁数」参照
前 10 分間最大 10 分間降水量				
16	pre10m_mx_rm	品質情報	整数	「3．品質情報」参照、対象範囲は項番 17～22
17	pre10m_mx	前 10 分間最大 10 分間降水量	小数（小数点以下 1 桁）	品質情報が 0～2 の場合「-9999.0」
18	pre10m_mx_ad	同附帯情報	整数	「4．附帯情報」参照
19	pre10m_mx_sf	同有効桁数	整数	「5．有効桁数」参照
20	pre10m_mx_dtd	同起日	整数（ゼロ埋め 2 桁）	品質情報が 0～2 の場合「-9999」
21	pre10m_mx_dth	同起時（時）	整数（ゼロ埋め 2 桁）	品質情報が 0～2 の場合「-9999」
22	pre10m_mx_dtm	同起時（分）	整数（ゼロ埋め 2 桁）	品質情報が 0～2 の場合「-9999」
前 10 分間最大 1 時間降水量				
23	pre1h_mx_rm	品質情報	整数	「3．品質情報」参照、対象範囲は項番 24～29
24	pre1h_mx	前 10 分間最大 1 時間降水量	小数（小数点以下 1 桁）	品質情報が 0～2 の場合「-9999.0」
25	pre1h_mx_ad	同附帯情報	整数	「4．附帯情報」参照
26	pre1h_mx_sf	同有効桁数	整数	「5．有効桁数」参照
27	pre1h_mx_dtd	同起日	整数（ゼロ埋め 2 桁）	品質情報が 0～2 の場合「-9999」
28	pre1h_mx_dth	同起時（時）	整数（ゼロ埋め 2 桁）	品質情報が 0～2 の場合「-9999」
29	pre1h_mx_dtm	同起時（分）	整数（ゼロ埋め 2 桁）	品質情報が 0～2 の場合「-9999」

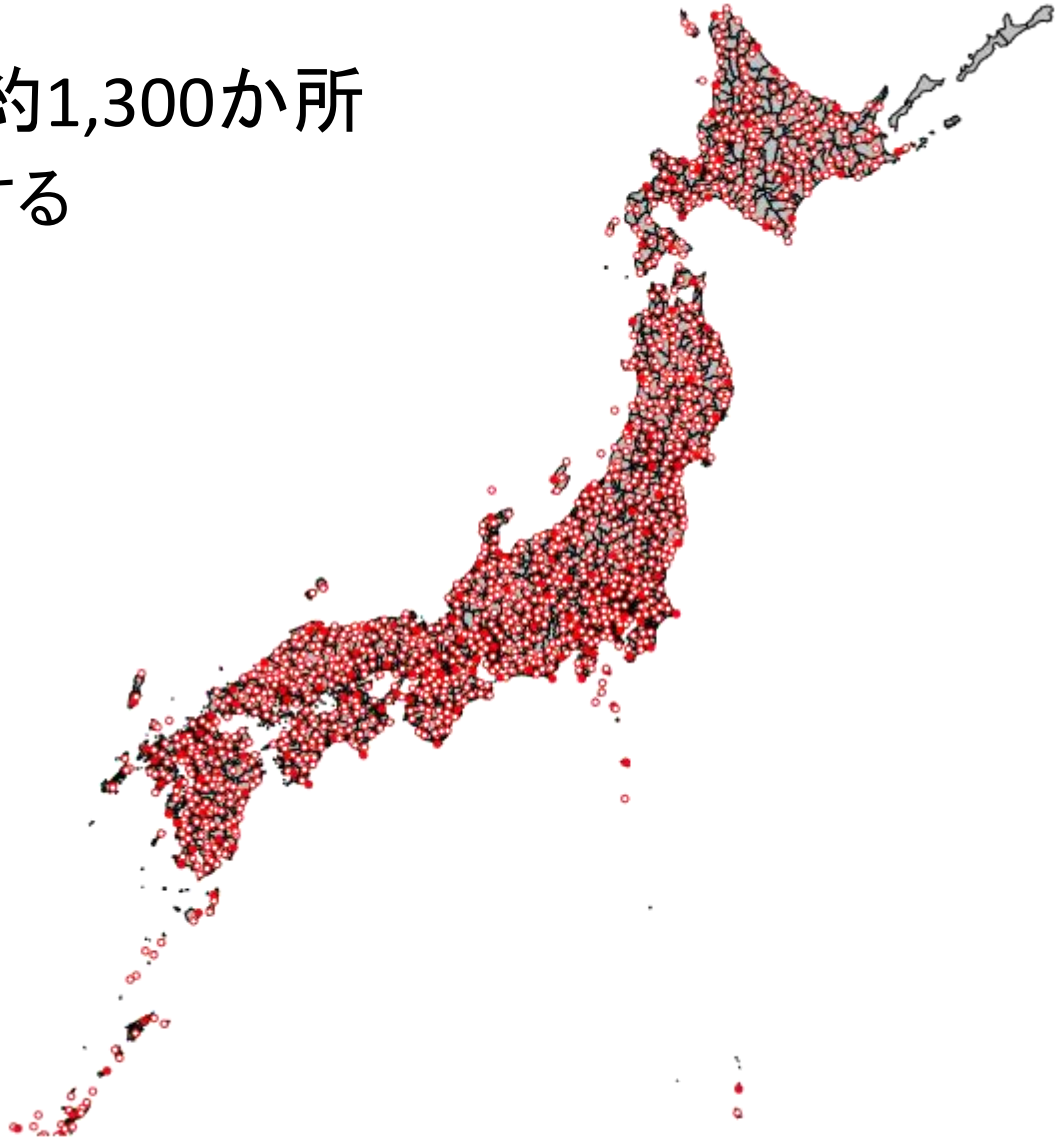
＜品質情報＞

値	意味	説明
0	統計しない	観測（統計）対象外の要素
1	資料なし（欠測）、未報告	欠測（統計値が得られない）、未来時刻のデータ
2	利用不適値	利用に適さない
3	疑問値	値が非常に疑わしい
4	資料不足値	統計を行うための元データに一定以上の欠落がある
5	準正常値	値がやや疑わしい（統計を行うための、元データに若干の欠落がある）
6		使用しない
7		使用しない
8	正常値	品質に問題がない（統計を行うための、元データに欠落がない）
9		使用しない

『format_amedas_csv_20180810.pdf』より

試用するデータの概要－３

- アメダス地点番号：約1,300か所
  - 増えたり、減ったりする



http://aramoto.sakura.ne.jp/amedas/

試用するサーバの概要

- クラウド上の某社仮想サーバ
  - CPU：8core、RAM：16GByte、HDD：1.6TByte
  - HDD容量重視で選択
  - かなりCPUパワーが高く、メモリも潤沢に使える

- 注意事項
  - 本試用はベンチマークが目的ではありません
  - 「この大きさのデータを扱う時のサイズ感」を確認するためです
  - そのため、厳密な環境下での時間測定ではありません
  - 再現性が無い場合も多々あります
  - あくまで参考値として見てください

降水量と気温を解凍

- １年ごとにtar&gzipされている
  - 10分ごとと、エリアごとの集計がある
- 解凍後

```
$ tar  xfz  pre_10m_2008.tar.gz
$ tar  xfz  pre_10m_2009.tar.gz
   :
```

	降水量	気温
容量	3.3G -> 58GByte	5.2G -> 32GByte
ファイル数	543,456	543,456
レコード数	711,310,414	507,886,014
アメダス地点数(のべ)	1,451	961
期間	2008年4月1日～ 2018年7月31日	2008年4月1日～ 2018年7月31日

```
$ tree  -d
.
|-- 2008
|    |-- 04
|    |-- 05
|    |-- 06
|    |-- 07
|    |-- 08
|    |-- 09
|    |-- 10
|    |-- 11
|    `-- 12
|-- 2009
|    |-- 01
|    |-- 02
```

- 降水量の方が、圧縮率が高い
- ファイル数は同じだが、レコード数が違う
  - アメダス地点数が違う

降水量:内容

- 先頭ファイルの先頭付近を見てみる

```
$ head  2008/04/amd_pre_10m_20080401_0000.csv
amd_no,sfc_no,yr,mn,dy,hr,mi,pre10m_rm,pre10m,pre10m_ad,pre10m_sf,pre1h_rm,pre1h,pre1h_ad,pre1h_sf,pre10m_mx_r
m,pre10m_mx,pre10m_mx_ad,pre10m_mx_sf,pre10m_mx_dtd,pre10m_mx_dth,pre10m_mx_dtm,pre1h_mx_rm,pre1h_mx,pr
e1h_mx_ad,pre1h_mx_sf,pre1h_mx_dtd,pre1h_mx_dth,pre1h_mx_dtm
11001,0,2008,04,01,00,00,8,0.0,1,0,8,0.0,0,0,0,-9999.0,0,1,-9999,-9999,-9999,0,-9999.0,0,1,-9999,-9999,-9999
11012,0,2008,04,01,00,00,1,-9999.0,1,0,1,-9999.0,0,0,0,-9999.0,0,1,-9999,-9999,-9999,0,-9999.0,0,1,-9999,-9999,-9999
11016,47401,2008,04,01,00,00,0,-9999.0,0,0,8,0.0,2,0,0,-9999.0,0,1,-9999,-9999,-9999,0,-9999.0,0,1,-9999,-9999,-9999
11046,0,2008,04,01,00,00,8,0.0,1,0,8,0.0,0,0,0,-9999.0,0,1,-9999,-9999,-9999,0,-9999.0,0,1,-9999,-9999,-9999
11061,0,2008,04,01,00,00,1,-9999.0,1,0,1,-9999.0,0,0,0,-9999.0,0,1,-9999,-9999,-9999,0,-9999.0,0,1,-9999,-9999,-9999
11076,0,2008,04,01,00,00,8,0.0,1,0,8,0.0,0,0,0,-9999.0,0,1,-9999,-9999,-9999,0,-9999.0,0,1,-9999,-9999,-9999
11091,0,2008,04,01,00,00,1,-9999.0,1,0,1,-9999.0,0,0,0,-9999.0,0,1,-9999,-9999,-9999,0,-9999.0,0,1,-9999,-9999,-9999
11121,0,2008,04,01,00,00,8,0.0,1,0,8,0.0,0,0,0,-9999.0,0,1,-9999,-9999,-9999,0,-9999.0,0,1,-9999,-9999,-9999
11151,0,2008,04,01,00,00,8,0.0,1,0,8,0.0,0,0,0,-9999.0,0,1,-9999,-9999,-9999,0,-9999.0,0,1,-9999,-9999,-9999
```

amd_no	sfc_no	yr	mn	dy	hr	mi	pre10m_rm	pre10m	pre10m_ad	pre10m_sf	pre1h_rm	pre1h	pre1h_ad	pre1h_sf
11001	0	2008	4	1	0	0	8	0.0	1	0	8	0.0	0	0
11012	0	2008	4	1	0	0	1	-9999.0	1	0	1	-9999.0	0	0
11016	47401	2008	4	1	0	0	0	-9999.0	0	0	8	0.0	2	0
11046	0	2008	4	1	0	0	8	0.0	1	0	8	0.0	0	0
11061	0	2008	4	1	0	0	1	-9999.0	1	0	1	-9999.0	0	0
11076	0	2008	4	1	0	0	8	0.0	1	0	8	0.0	0	0
11091	0	2008	4	1	0	0	1	-9999.0	1	0	1	-9999.0	0	0
11121	0	2008	4	1	0	0	8	0.0	1	0	8	0.0	0	0
11151	0	2008	4	1	0	0	8		1	0	8	0.0	0	0

「品質情報」が「8:正常値」以外が意外と多い

気温：内容

- 先頭ファイルの先頭付近を見てみる

```
$ head  2008/04/amd_tem_10m_20080401_0000.csv
amd_no,sfc_no,yr,mn,dy,hr,mi,tem_rm,tem,mxtem_rm,mxtem,mxtem_dtd,mxtem_dth,mxtem_dtm,mntem_rm,mntem,mntem_
dtd,mntem_dth,mntem_dtm
11001,0,2008,04,01,00,00,8,-3.1,0,-9999.0,-9999.0,-9999,-9999,0,-9999.0,-9999.0,-9999,-9999
11012,0,2008,04,01,00,00,8,-2.1,0,-9999.0,-9999.0,-9999,-9999,0,-9999.0,-9999.0,-9999,-9999
11016,47401,2008,04,01,00,00,8,-2.5,0,-9999.0,-9999,-9999,-9999,0,-9999.0,-9999,-9999,-9999
11046,0,2008,04,01,00,00,8,-2.3,0,-9999.0,-9999.0,-9999,-9999,0,-9999.0,-9999.0,-9999,-9999
11061,0,2008,04,01,00,00,8,-2.9,0,-9999.0,-9999.0,-9999,-9999,0,-9999.0,-9999.0,-9999,-9999
11076,0,2008,04,01,00,00,8,-2.6,0,-9999.0,-9999.0,-9999,-9999,0,-9999.0,-9999.0,-9999,-9999
11091,0,2008,04,01,00,00,8,-2.7,0,-9999.0,-9999.0,-9999,-9999,0,-9999.0,-9999.0,-9999,-9999
11121,0,2008,04,01,00,00,8,-3.2,0,-9999.0,-9999.0,-9999,-9999,0,-9999.0,-9999.0,-9999,-9999
11151,0,2008,04,01,00,00,8,-2.6,0,-9999.0,-9999.0,-9999,-9999,0,-9999.0,-9999.0,-9999,-9999
```

amd_no	sfc_no	yr	mn	dy	hr	mi	tem_rm	tem	mxtem_rm	mxtem	mxtem_dtd	mxtem_dth	mxtem_dtm
11001	0	2008	4	1	0	0	8	−3.1	0	−9999.0	−9999.0	−9999	−9999
11012	0	2008	4	1	0	0	8	−2.1	0	−9999.0	−9999.0	−9999	−9999
11016	47401	2008	4	1	0	0	8	−2.5	0	−9999.0	−9999	−9999	−9999
11046	0	2008	4	1	0	0	8	−2.3	0	−9999.0	−9999.0	−9999	−9999
11061	0	2008	4	1	0	0	8	−2.9	0	−9999.0	−9999.0	−9999	−9999
11076	0	2008	4	1	0	0	8	−2.6	0	−9999.0	−9999.0	−9999	−9999
11091	0	2008	4	1	0	0	8	−2.7	0	−9999.0	−9999.0	−9999	−9999
11121	0	2008	4	1	0	0	8	−3.2	0	−9999.0	−9999.0	−9999	−9999
11151	0	2008	4	1	0	0	8	−2.6	0	−9999.0	−9999.0	−9999	−9999

API提供データフォーマット表には
『品質情報が0～2の場合「-9999」』
でも、「-9999」と「-9999.0」が混在している

Linuxのコマンドでテキスト処理し、データの概要を把握する

最新版（2018年9月）のデータを使用

2018年10月1日

先端IT活用推進コンソーシアム
クラウド・テクノロジー活用部会
リーダー　荒本道隆

2017年の降水量をカウント

- 2017年の「前10分間降水量」値の件数を調べる

```
$ cd  2017
$ find  .  -name  ¥*.csv  |  sort  |  awk ' {print "cut -d, -f9 " $0}'  >  $HOME/aaa.sh
$ time  bash  $HOME/aaa.sh  |  sort  -n  |  uniq  -c
```

  - 所要時間：**3分50秒**（1年分）
- 動かしてみると
  - 値は0.5きざみ
  - 「-9999.0」が結構ある
  - 「pre10m」（ヘッダ）が何度も出てくる
  - 「214748400.0」って何だ？
  - とりあえず1年分で約4分は遅い
    - 11年分だと単純計算で約40分かかる

```
    309,655   -9999.0
 65,438,539   0.0
     52,560   pre10m
  2,035,345   0.5
    390,934   1.0
    136,837   1.5
     63,891   2.0
     36,093   2.5
     23,404   3.0
          ～    ～
          1   31.0
          1   32.0
         15   214748400.0
```

2017年の降水量ファイルを結合

- 1つのファイルにまとめると、処理が楽になるはず

```
$ cd  2017
$ time  cat  */*.csv  >  merge2017.csv
```

  - 所要時間：**2分1秒**（1年分）
- 「前10分間降水量」の分布

```
$ time  cut  -d,  -f9  merge2017.csv  |  sort  -n  |  uniq  -c
```

  - 所要時間：**2分37秒**（1年分）
    - ファイルを1つにしただけで、**230秒** → **157秒**になった
  - 結果はまったく同じ
- もうちょっとだけ頑張って高速化
  - 「sort」の負荷が高い → 同じ数字が大量に出てくる → 随時集計

```
$ time  awk 'BEGIN{FS=","}{t[$9]=t[$9]+1}END{for(i in t){print t[i]"¥t"i;}}' < merge2017.csv | ¥
sort  -n  -k 2
```

  - 所要時間：**61秒**（1年分）

2017年、月の降水量値の合計－1

- 単純に降水量を合計すると

```
$ time  awk  'BEGIN{FS=","}{t[$4]=t[$4]+$9}END{for(i in t){print t[i]"¥t"i;}}' < merge2017.csv |  ¥
sort  -n  -k 2
```

  - 所要時間：**82秒**（1年分）
  - 「-9999.0」「214748400.0」が邪魔している

-4.79497e+08	01
-5.05135e+08	02
-42892983	03
958373497	04
-180652869	05
-7.22066e+07	06
-129262253	07
1008556028	08
-1.14489e+08	09
-152758694	10
139590297	11
-302348747	12

2017年、月の降水量値の合計－2

- 品質情報が「8:正常値」だけを合計

```
$ time  awk 'BEGIN{FS=","}{if ($8=="8"){t[$4]=t[$4]+$9}}END{for(i in t){print t[i]"¥t"i;}}' < merge2017.csv | ¥
sort  -n  -k 2
```

  - 所要時間：**98秒**（1年分）
  - 2017年は、10月に雨が多かった

```
114,924    01
114,810    02
102,844    03
179,794    04
118,822    05
215,958    06
264,591    07
237,293    08
269,380    09
425,852    10
134,454    11
110,890    12
```

11年分の品質情報の分布

- 「品質情報」の各件数

```
$ time  awk 'BEGIN{FS=","}{t[$8]=t[$8]+1}END{for(i in t){print t[i]"¥t"i;}}' < merge.csv | ¥
sort  -n  -k 2
```

  - 所要時間：**11分16秒**（11年分）
  - 1.59%が「8：正常値」以外（ヘッダ除く）

1,885,963	0：統計しない、観測（統計）対象外
543,456	pre10m_rm：ヘッダ
8,939,437	1：資料なし（欠測）、未報告
409,967	3：疑問値
29,989	4：資料不足値
76,073	5：準正常値
699,968,985	8：正常値

11年分のアメダス地点番号の件数

- 「アメダス地点番号」の件数と各レコード数

```
$ time  awk  'BEGIN{FS=","}{t[$1]=t[$1]+1}END{for(i  in t){print t[i]"¥t"i;}}' < merge.csv |  ¥
sort  -n  -r  > merge.log
$ wc  -l  merge.log
$ head  merge.log
$ tail  merge.log
```

- 所要時間：**11分7秒**（11年分）

```
1,451
```

```
556,848   64036
556,704   86141
556,704   33776
556,704   33751
556,704   33336
556,704   20606
555,984   40191
552,385   71231
552,385   61286
552,384   88317
```

```
 35,137   44206
 35,137   31331
 30,817   49051
 26,353   15366
 22,032   34051
 17,568   33901
  8,784   11013
      1   71221
      1   71041
      1   49106
```

特定の地点だけ月集計－１

- アメダス地点番号「44132」だけを集計

```
$ time  awk 'BEGIN{FS=","}{if ($1=="44132" && $8=="8"){t[$4]=t[$4]+$9}}END{for(i in t){print t[i]"¥t"i;}}' < merge2017.csv | ¥
sort  -n  -k 2
```

  - 所要時間：**67秒**（1年分）
  - 5.7GByteのフルスキャンは効率が悪い

26	01
15.5	02
85.5	03
122	04
49	05
106.5	06
81	07
141.5	08
209.5	09
531.5	10
47	11
15	12

- 地点ごとに整理されたファイルもある

  /kansoku/amedas_csv/pre/10m/area

  amd_pre_10m_201701_**44132**.csv

  - 各アメダス地点の１月分が1ファイル
  - １年分の集計＝12ファイルの操作

特定の地点だけ月集計－２

- アメダス地点番号が「44132」だけ集計、高速化

```
$ cd  area
$ time  for  i  in  `find  .  -name  amd_pre_10m_2017??_44132.csv | sort `; do
    awk 'BEGIN{FS=","}{if ($8=="8"){t[$4]=t[$4]+$9}}END{for(i in t){print t[i]"¥t"i;}}' < $i
done
```

- 所要時間：**0.3秒**（1年分）
- ファイル名で、要素を絞りこめる
  - 年、月、アメダス地点番号
- 結果はまったく同じ

26	01
15.5	02
85.5	03
122	04
49	05
106.5	06
81	07
141.5	08
209.5	09
531.5	10
47	11
15	12

特定の地点だけ月集計－3

- アメダス地点番号が「44132」だけ集計、高速化、11年分

```
$ cd  area
$ time  for  i  in  `find  .  -name  amd_pre_10m_20????_44132.csv | sort `; do
    awk 'BEGIN{FS=","}{if ($8=="8"){t[$3"/"$4]=t[$3"/"$4]+$9}}END{for(i in t){print t[i]"¥t"i;}}' < $i
done
```

  - 所要時間：**1.7秒**（11年分）
  - 2017年の結果はまったく同じ
  - 年単位で算出するには、もう1工夫必要
    - アメダス地点番号ごとのファイルを作る
    - 1ファイルなら、シェルの書き方がブレなくなる
- この性能なら、11年分でも問題無し
  - 100年分なら？

    → 別方式で処理

    → 年単位で並列処理

```
 48.5    2008/06
 48      2008/07
387.5    2008/08
158.5    2008/09
204.5    2008/10
 74      2008/11
 70.5    2008/12
142      2009/01
 46.5    2009/02
 98.5    2009/03
162.5    2009/04
242      2009/05
～       ～
```

所感

- 利便性＆高速化を狙うのなら
  - 様々な観点で、ファイルにまとめる
    - time と area で同じフォーマットなのがありがたい
    - awk コマンドだけでたいていの事ができる
  - 不要なカラムを削除する
    - ファイルサイズが小さくなる -> 高速化
- 簡単な集計処理なら、十分にできそうだが
  - カラム番号を暗記すれば、あまり困らない
  - ディレクトリ構造やファイル名についても記憶する必要アリ
  - UNIXのコマンドも覚えておく必要アリ（後々、便利）
- 今回は、ヘッダの処理を省略
  - カラム番号を間違えていないかの確認に使用

参考：CSVに直接クエリできるツール

- 『q』を使えばCSVファイルに直接SQLを実行できる

http://harelba.github.io/q/ja/index.html

```
$ time  ./q  "select count(*),pre10m from merge2017.csv group by pre10m"  --delimiter=,  -H
```

- 所要時間：**66分**（1年分）

大きなデータとクラウド、データベースを少し
～ Amazon S3 Select と PostgreSQL ～

2018年10月1日

先端IT活用推進コンソーシアム
クラウド・テクノロジー活用部会
サブリーダー上村準也

大きなデータをクラウドで

- Amazon S3 Select
  - 大きなCSVを圧縮したまま Amazon S3 へ置く
  - その内容を SQL の SELECT 文で分析できます
  - 公式「すぐに活用できるクエリ」の最初の1つ
  - REST 型 API
    - 各種 SDK と AWS CLI から使える
  - ここでは簡単に AWS CLI から試します

Amazon S3 Select

- 材料
  - アメダスの気温10分値 CSV ファイルを10年分
  - AWS Command Line Interface (CLI)
    - 公式「AWS Command Line Interface のインストール」
  - tar, head, grep などコマンドラインツールを少々

Amazon S3 Select

- 下拵え
  - 10年分のデータを1つの大きなCSVファイルにしておく
  - ヘッダ行は使いますので、最初の1行は残して、他は取り除いておきます
  - 詳しくはシェルスクリプト「amedas_csv.sh」を見てください

https://gist.github.com/uemuraj?sort=updated

```
$ time amedas_csv.sh kansoku/amedas_csv/tem/10m/time/tem_10m_{2008..2018}.tar.gz | gzip > /var/tmp/tem_10m.csv.gz

real    34m27.401s
user    39m40.226s
sys     2m38.902s
```

時間別のファイル 2008 年から10年分

  - これを S3 バケットへ転送して置いてください

Amazon S3 Select

- 道具
  - AWS CLI で s3api サブコマンドの select-object-content という機能を使うのですが…
  - 繰り返し易いようシェルスクリプト「s3select.sh」を使っています

```
$ time s3select.sh s3://amedas.sierra1337.info/tem_10m.csv.gz “
> SELECT COUNT(*) FROM s3object”
506799101

real    5m24.375s
user    0m0.870s
sys     0m0.149s
```

5億レコード…

これはキーワードのようです

Amazon S3 Select

- ヘッダ行から列名を認識してくれます
- サポートするデータ型がありCAST()して扱います
  - CAST()しないと文字列扱いです
- 集計関数が何種類か使えます
  - でもGROUP BYが…

```
$ time s3select.sh s3://amedas.sierra1337.info/tem_10m.csv.gz "
> SELECT MIN(CAST(amd_no AS integer)), MAX(CAST(amd_no AS integer))
FROM s3object"
11001,94121

real    28m19.881s
user    0m1.505s
sys     0m0.165s
```

列名でカラム指定可能

全件処理だと約30分間

Amazon S3 Select

- WHERE 句や LIMIT 句で行を絞り込むと速い
  - 普通のテキスト処理と同じ
- NULL 値の扱いが SQL のルールのままで便利

```
$ time s3select.sh s3://amedas.sierra1337.info/tem_10m.csv.gz “
> SELECT MAX(NULLIF(CAST(tem as real), -9999.0)), MIN(NULLIF(CAST(tem
as real), -9999.0)) FROM s3object
> WHERE amd_no = '11001'"
29.7,-14.7

real    4m59.943s
user    0m0.852s
sys     0m0.206s
```

値が -9999.0 の場合に NULL を返すが適当に計算される

絞り込めば約5分間

Amazon S3 Select

- ただし、アメダスの CSV の内容だと年月日時分が列で分かれているので SQL としては見た目が良くない

- SELECT 文で CSV から新しい CSV を作ります
  - 無効値は最初から NULL にしておきます
  - 年,月,日,時,分をまとめてタイムスタンプに変換します
    - CASE 式も使えるので、前10分間の時刻の処理も何とかできました
    - 前10分で日をまたぐと、月もまたぐし、年もまたぐ…
    - ただし日はあるのでカレンダーを考慮しなくても計算できます
    - 以下は「API 提供データフォーマット表」から一部抜粋

最低気温				
15	mntem_rm	品質情報	整数	「３．品質情報」参照、対象範囲は項番 16～19
16	mntem	前 10 分間最低気温	小数（小数点以下 1 桁）	品質情報が 0～2 の場合「-9999.0」
17	mntem_dtd	同起日	整数（ゼロ埋め 2 桁）	品質情報が 0～2 の場合「-9999」
18	mntem_dth	同起時（時）	整数（ゼロ埋め 2 桁）	品質情報が 0～2 の場合「-9999」
19	mntem_dtm	同起時（分）	整数（ゼロ埋め 2 桁）	品質情報が 0～2 の場合「-9999」

Amazon S3 Select

```
SELECT amd_no, sfc_no,
 yr || '-' || mn || '-' || dy || 'T' || hr || ':' || mi || ':00Z',
 tem_rm, NULLIF(CAST(tem as real), -9999.0),
 mxtem_rm, NULLIF(CAST(mxtem as real), -9999.0),
 CASE
  WHEN dy <> '01' OR hr <> '00' OR mi <> '00' THEN
   yr || '-' || CAST(CAST(mn AS integer) AS string) || '-' || CAST(NULLIF(CAST(CAST(mxtem_dtd AS float) AS integer), -9999) AS string) || 'T' || mxtem_dth || ':' || mxtem_dtm || ':00Z'
  WHEN mn <> '01' THEN
   yr || '-' || CAST((CAST(mn AS integer) - 1) AS string) || '-' || CAST(NULLIF(CAST(CAST(mxtem_dtd AS float) AS integer), -9999) AS string) || 'T23:' || mxtem_dtm || ':00Z'
  ELSE
   CAST((CAST(yr As integer) - 1) AS string) || '-12-' || CAST(NULLIF(CAST(CAST(mxtem_dtd AS float) AS integer), -9999) AS string) || 'T23:' || mxtem_dtm || ':00Z'
 END,
 mntem_rm, NULLIF(CAST(mntem as real), -9999.0),
 CASE
  WHEN dy <> '01' OR hr <> '00' OR mi <> '00' THEN
   yr || '-' || CAST(CAST(mn AS integer) AS string) || '-' || CAST(NULLIF(CAST(CAST(mntem_dtd AS float) AS integer), -9999) AS string) || 'T' || mntem_dth || ':' || mntem_dtm || ':00Z'
  WHEN mn <> '01' THEN
   yr || '-' || CAST((CAST(mn AS integer) - 1) AS string) || '-' || CAST(NULLIF(CAST(CAST(mntem_dtd AS float) AS integer), -9999) AS string) || 'T23:' || mntem_dtm || ':00Z'
  ELSE
   CAST((CAST(yr As integer) - 1) AS string) || '-12-' || CAST(NULLIF(CAST(CAST(mntem_dtd AS float) AS integer), -9999) AS string) || 'T23:' || mntem_dtm || ':00Z'
 END
FROM s3object
```

SQL で文字列連結

前10分時刻を3パターンで処理

NULL を連結しても結果が単純に NULL になるだけ

日が -9999.0 であることがあったので2重にキャスト

Amazon S3 Select

- 新しい CSV ファイルができました
  - これをもう一度 S3 に転送して S3 Select することができます
  - あるいはデータベースのテーブルデータとして利用することもできます

```
$ time s3select.sh s3://amedas.sierra1337.info/tem_10m.csv.gz "
> SELECT
（中略ー前スライドのSQL文）
> FROM s3object" /var/tmp/tem_table.csv

real    66m19.440s
user    16m35.167s
sys     8m55.473s

$ cat /var/tmp/tem_table.csv | hea
11001,0,2008-06-18T12:00:00Z,8,17.0,0,,,0,,
11012,0,2008-06-18T12:00:00Z,8,15.2,0,,,0,,
11016,47401,2008-06-18T12:00:00Z,8,15.4,0,,,0,,
11046,0,2008-06-18T12:00:00Z,8,13.4,0,,,0,,
11061,0,2008-06-18T12:00:00Z,8,16.1,0,,,0,,
11076,0,2008-06-18T12:00:00Z,8,18.2,0,,,0,,
11091,0,2008-06-18T12:00:00Z,8,16.8,0,,,0,,
11121,0,2008-06-18T12:00:00Z,8,15.3,0,,,0,,
11151,0,2008-06-18T12:00:00Z,8,14.1,0,,,0,,
11176,0,2008-06-18T12:00:00Z,8,15.2,8,15.2,2008-6-18T12:00:00Z,8,15.1,2008-6-18T11:59:00Z
```

文字列連結で作成した時刻

NULL 値はカンマだけになります

Amazon S3 Select

- ヘッダ付けてからもう一度 S3 に転送、SELECT を試します

```
$ time echo "amd_no,sfc_no,tem_dt,tem_rm,tem,mxtem_rm,mxtem,mxtem_dt,mntem_rm,mntem,mntem_dt"
| cat - /var/tmp/tem_table.csv | gzip > /var/tmp/tem_table.csv.gz

real    32m30.852s
user    28m33.622s
sys     1m48.514s

$ zcat tem_table.csv.gz | head
amd_no,sfc_no,tem_dt,tem_rm,tem,mxtem_rm,mxtem,mxtem_dt,mntem_rm,mntem,mntem_dt
11001,0,2008-06-18T12:00:00Z,8,17.0,0,,,0,,
11012,0,2008-06-18T12:00:00Z,8,15.2,0,,,0,,
11016,47401,2008-06-18T12:00:00Z,8,15.4,0,,,0,,
11046,0,2008-06-18T12:00:00Z,8,13.4,0,,,0,,
11061,0,2008-06-18T12:00:00Z,8,16.1,0,,,0,,
11076,0,2008-06-18T12:00:00Z,8,18.2,0,,,0,,
11091,0,2008-06-18T12:00:00Z,8,16.8,0,,,0,,
11121,0,2008-06-18T12:00:00Z,8,15.3,0,,,0,,
11151,0,2008-06-18T12:00:00Z,8,14.1,0,,,0,,
```

オリジナルの新しいヘッダ行を最初に

3つの時刻がそれぞれ1つの列に

Amazon S3 Select

- タイムスタンプがちょっと良い感じで使えます

この条件だと意味はありませんが試してみました

```
$ time s3select.sh s3://amedas.sierra1337.info/tem_table.csv.gz "
> SELECT * FROM s3object WHERE CAST(tem_dt AS timestamp) BETWEEN
> CAST('2008-06-18T12:00:00Z' AS timestamp) AND CAST('2008-06-18T12:00:00Z' AS timestamp)
> LIMIT 10"
11001,0,2008-06-18T12:00:00Z,8,17.0,0,,,0,,
11012,0,2008-06-18T12:00:00Z,8,15.2,0,,,0,,
11016,47401,2008-06-18T12:00:00Z,8,15.4,0,,,0,,
11046,0,2008-06-18T12:00:00Z,8,13.4,0,,,0,,
11061,0,2008-06-18T12:00:00Z,8,16.1,0,,,0,,
11076,0,2008-06-18T12:00:00Z,8,18.2,0,,,0,,
11091,0,2008-06-18T12:00:00Z,8,16.8,0,,,0,,
11121,0,2008-06-18T12:00:00Z,8,15.3,0,,,0,,
11151,0,2008-06-18T12:00:00Z,8,14.1,0,,,0,,
11176,0,2008-06-18T12:00:00Z,8,15.2,8,15.2,2008-6-18T12:00:00Z,8,15.1,2008-6-18T11:59:00Z
```

ソートがされていないので
head コマンドと同じ結果が見られます

データベースを少し

- PostgreSQL
  - 説明は省略します
  - 今回はバージョン 10.4 を使用しています

- 新しい CSV をそのままテーブルデータとしてロードします
  - 非標準の COPY 文を利用します
  - CSV から1行ずつ対応する INSERT 文を作ることもできますが…
  - 5億レコードもあるのですから、試す前から止めておきます

PostgreSQL

- 表の定義は簡単に済ませておきます
  - 新しい CSV に合わせただけで、特に何もしていません

```
$ psql -U amedas
psql (10.4)
Type "help" for help.

amedas=> ¥d tem
                        Table "public.tem"
  Column   |            Type             | Collation | Nullable | Default
-----------+-----------------------------+-----------+----------+---------
 amd_no    | integer                     |           |          |
 sfc_no    | integer                     |           |          |
 tem_dt    | timestamp without time zone |           |          |
 tem_rm    | character(1)                |           |          |
 tem       | real                        |           |          |
 mxtem_rm  | character(1)                |           |          |
 mxtem     | real                        |           |          |
 mxtem_dt  | timestamp without time zone |           |          |
 mntem_rm  | character(1)                |           |          |
 mntem     | real                        |           |          |
 mntem_dt  | timestamp without time zone |           |          |
```

PostgreSQL

- データをロードして件数を見ます
  - COPY 文には2種あり、サーバープロセスから直接 CSV を読む、管理者によるロードの方が速いそうです

```
$ psql -U postgres
psql (10.4)
Type “help” for help.

postgres=# ¥c amedas
You are now connected to database “amedas” as user “postgres”.

amedas=# ¥timing
Timing is on.

amedas=# COPY tem FROM ‘/var/tmp/tem_table.csv’ WITH (FORMAT csv);
COPY 506799101
Time: 2125981.681 ms (35:25.982)
```

5億レコード…

PostgreSQL

- ここから先は…

あなたが自分の手で
確かめてください！

```
$ psql -U amedas

amedas=> ¥timing
Timing is on.

amedas=> SELECT COUNT(*) FROM tem;
   count
-----------
 506799101
(1 row)

Time: 262723.327 ms (04:22.723)

amedas=> SELECT MAX(tem), MIN(tem) FROM tem WHERE amd_no = 11001;
 max  |  min
------+-------
 29.7 | -14.7
(1 row)

Time: 202871.501 ms (03:22.872)
```

最後に:大事なお金の話

▾ Data Transfer		$21.39
▾ **Asia Pacific (Tokyo)**		**$21.39**
Bandwidth		$21.39
$0.000 per GB - data transfer in per month	22.819 GB	$0.00
$0.000 per GB - first 1 GB of data transferred out per month	1 GB	$0.00
$0.114 per GB - first 10 TB / month data transfer out beyond the global free tier	187.600 GB	$21.39
▾ Simple Storage Service		$0.51
▾ **Asia Pacific (Tokyo)**		**$0.51**
Amazon Simple Storage Service APN1-Requests-Tier1		$0.01
$0.0047 per 1,000 PUT, COPY, POST, or LIST requests	2,983 Requests	$0.01
Amazon Simple Storage Service APN1-Requests-Tier2		$0.00
$0.0037 per 10,000 GET and all other requests	159 Requests	$0.00
Amazon Simple Storage Service APN1-Select-Returned-Bytes		$0.15
$0.0008 per GB - for bytes returned by S3 Select in Standard	188.305 GB	$0.15
Amazon Simple Storage Service APN1-Select-Scanned-Bytes		$0.16
$0.00225 per GB - for bytes scanned by S3 Select in Standard	71.135 GB	$0.16
Amazon Simple Storage Service APN1-TimedStorage-ByteHrs		$0.19
$0.025 per GB - first 50 TB / month of storage used	7.502 GB-Mo	$0.19

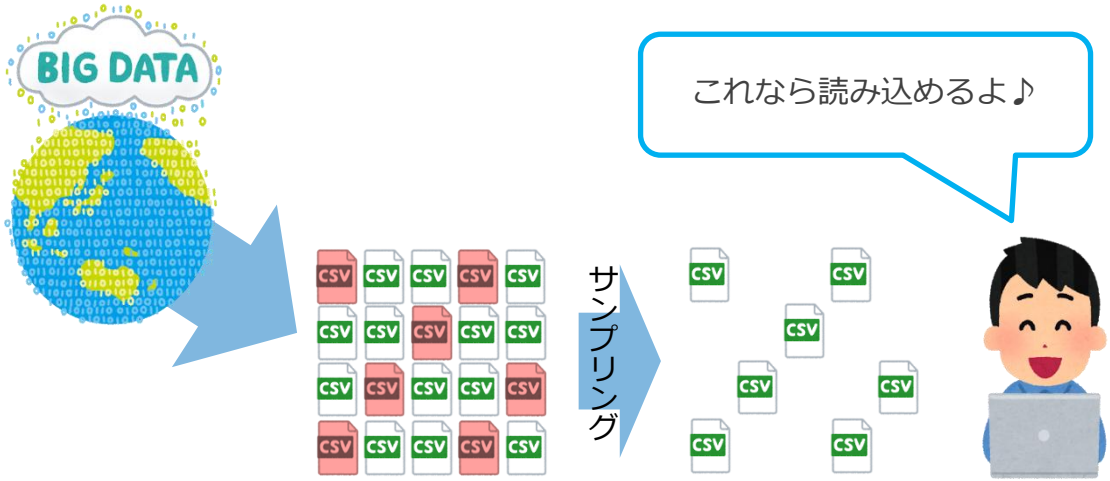
オープンデータで機械学習
データ管理形式の検討
～SQLiteに入れて観測点のクラスタリングを試してみる～

2018年10月1日

先端IT活用推進コンソーシアム
クラウド・テクノロジー活用部会
中嶋　俊治

オープンデータで機械学習

- 問題点：データ容量の問題
  - 機械学習はオンメモリで行うものが多いため，全データを使っての学習は難しい
  - 適切にサンプリングや集計を行ったデータを用意する必要がある

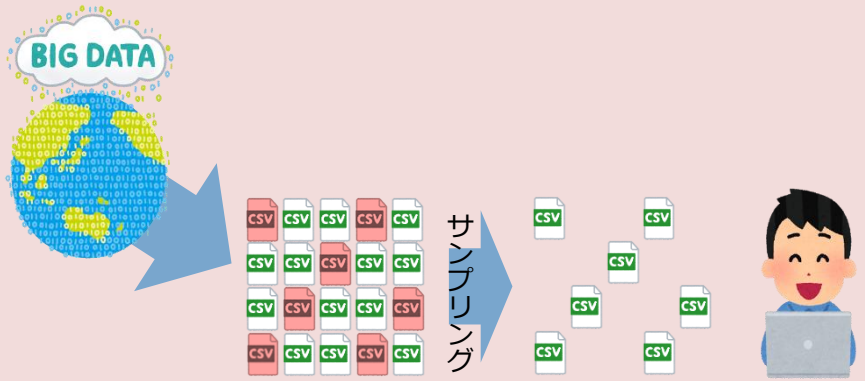


オープンデータで機械学習

- ファイルで管理　VS　DBで管理

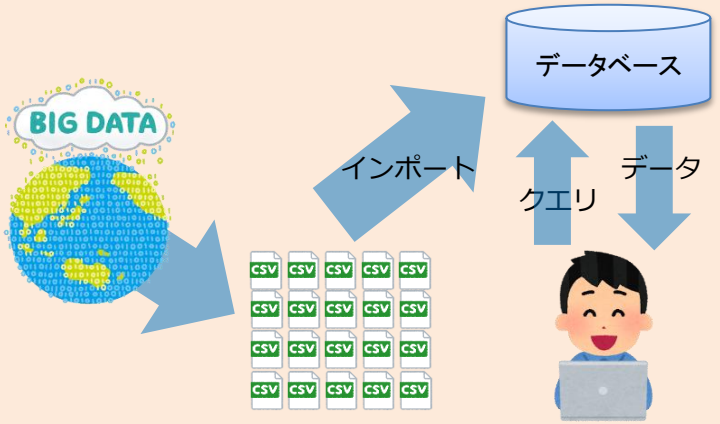
✓ ファイルで管理

- 高速に読み書きが可能
- 各ファイルの一部のみを取り出すような処理は面倒
- サンプリングの仕方等が属人化する恐れがある



✓ DBで管理

- SQLで書けばサンプリングの条件や集計の仕方が明確
- DBによってはクエリ実行が遅い
- モデルとデータの紐付けが困難になりやすい

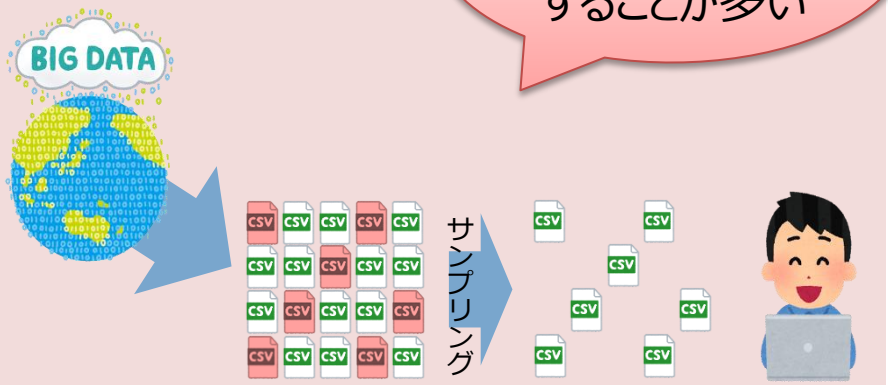


オープンデータで機械学習

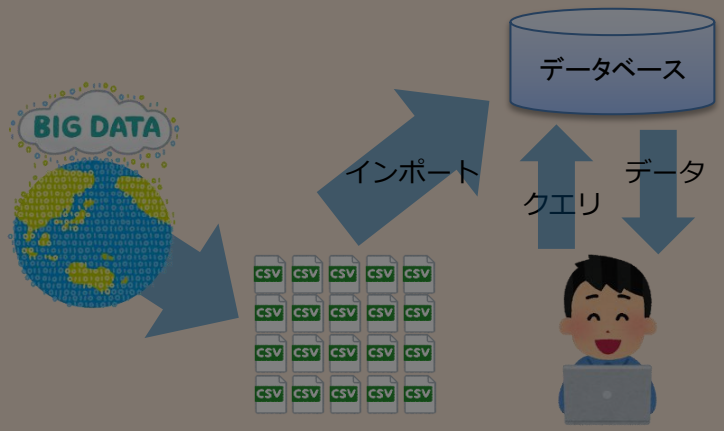
- ファイルで管理　VS　DBで管理

一般的には
ファイルで管理
することが多い

✓ ファイルで管理
- 高速に読み書きが可能
- 各ファイルの一部のみを取り出すような処理は面倒
- サンプリングの仕方等が属人化する恐れがある



✓ DBで管理
- SQLで書けばサンプリングの条件や集計の仕方が明確
- DBによってはクエリ実行が遅い
- モデルとデータの紐付けが困難になりやすい



オープンデータで機械学習

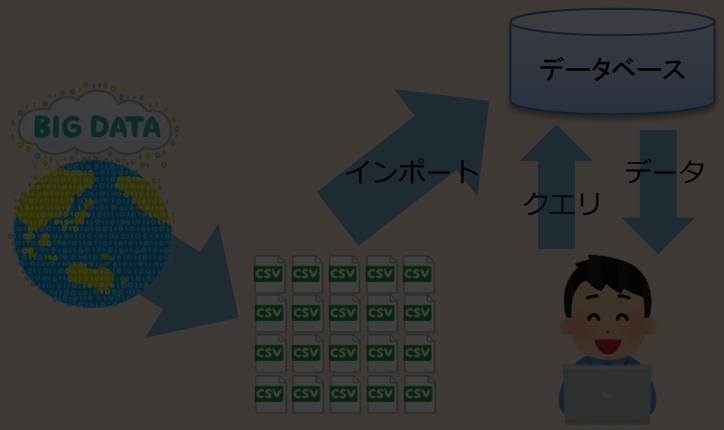
- ファイルで管理　VS　DBで管理

一般的には
ファイルで管理
することが多い

✓ ファイルで管理
- 高速に読み書きが可能
- 各ファイルの一部のみを取り出すような処理は面倒
- サンプリングの仕方毎が属人化する恐れがある

**実際のところ機械学習において
DBでのデータ管理が
有効なのかを検討**

✓ DBで管理
- SQLで書けばサンプリングの条件や集計の仕方が明確
- DBによってはクエリ実行が遅い
- モデルとデータの紐付けが困難になりやすい



機械学習におけるDBデータ管理

- 機械学習に親和性が高いDB ： SQLite3
  - ➢ 特徴
    - アプリケーションに組み込んで利用される軽量DB
    - 複数人での同時書き込みやパスワードでのデータ保護などができないが，その分高速に動作する
  - ➢ 機械学習との親和点
    - 1データ1ファイルで管理できるためデータとモデルの紐付けが容易
    - Pythonの標準ライブラリでSQLite3を扱うモジュールが用意されており，言語との親和性も高い

実験内容

- 実験環境 – 部会の仮想サーバ
  - OS：CentOS 6.9
  - CPU： Intel Xeon E312xx (Sandy Bridge) 8 core
  - RAM：16GByte
- 使用データ
  - アメダスにより記録された10分ごと気温データ10年分（2008年4月～2017年12月）の日本全国1302地点分
- 実験内容
  - 各地点の月別平均・最高・最低気温を用いて各地点をクラスタリング
  - DBからの実行とファイルからの実行で速度を比較

実験の流れ

1. CSVをまとめる
   - 10分毎に別ファイルになっているものを一つにまとめる
2. SQLite3にCSVをインポート
   - インポートにかかる時間を計測する
3. SQLite3を用いてクラスタリング
   - データの読み出し・クラスタリングにかかる時間を計測
4. CSVからクラスタリング
   - 実行時間を計測し、SQLite3の実行時間と比較

実験結果

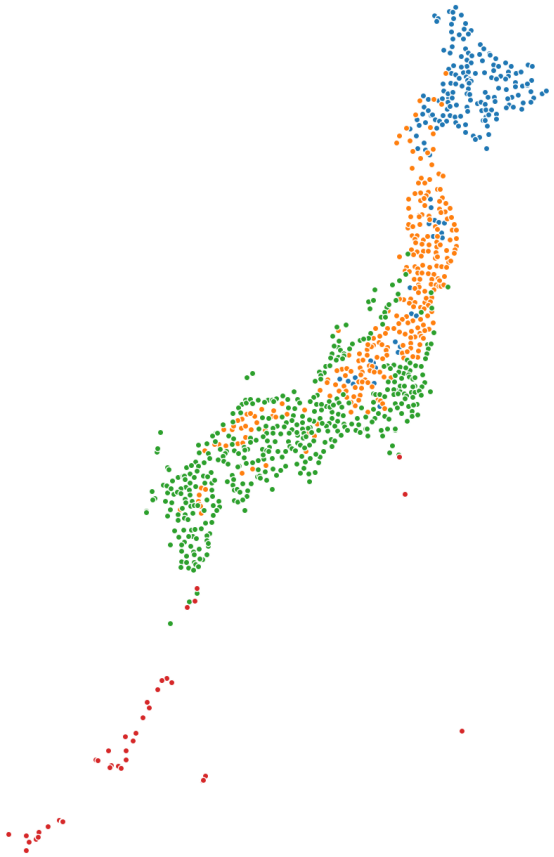
1. CSVをまとめる
   - 所要時間：1736秒（28分56秒）
   - 29GByteのCSVが生成
2. SQLite3にCSVをインポート
   - 所要時間：3222秒（53分40秒）
   - インポート後のファイル容量は32GByte
   - 474,222,480レコード

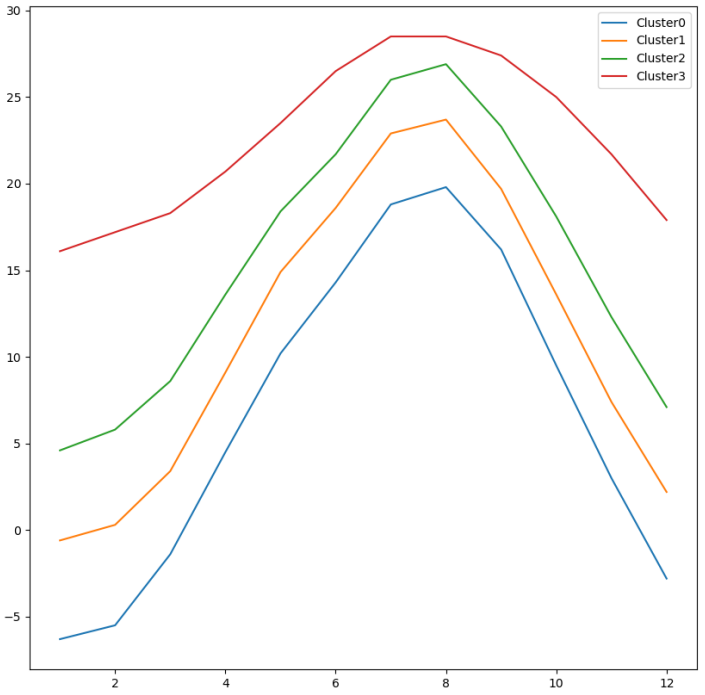
手順3：SQLite3からクラスタリング

- Pythonを用いてSQLiteから地点・月ごとに集計
  - > select amd_no, mn, avg(tem), max(tem), min(tem) from tem where tem != -9999.0 group by amd_no, mn;
  - 地点・月ごとに平均気温・最高気温・最低気温を取得
  - 異常値には-9999.0が入っているのでそれは除く
  - 所要時間：1108秒（18分28秒）← 5回実行した平均
- sklearnのKMeans関数でクラスタリング
  - 特徴量は，各月の平均・最高・最低気温で36次元のデータ
  - K=4でクラスタリング
    - > KMeans(n_clusters=4, random_state=1).fit_predict(data)
    - 所要時間：4秒　あらかじめ集計したデータを用いているので高速

手順3：SQLite3からクラスタリング

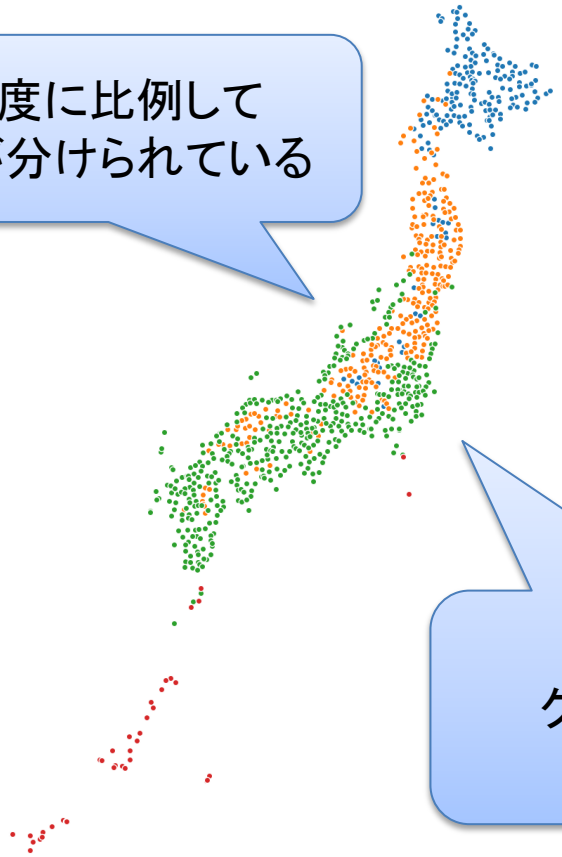
- 各クラスタの平均気温平均とアメダスの座標をプロット

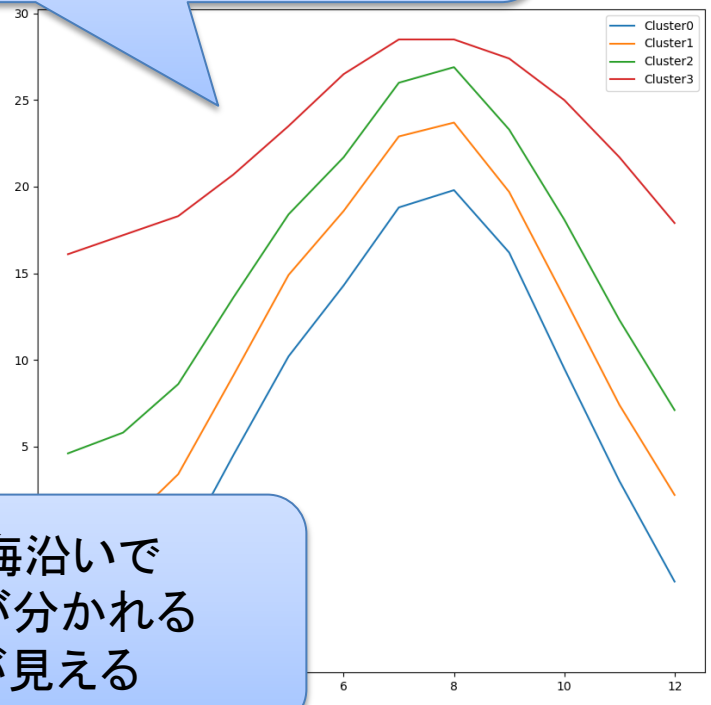




手順3：SQLite3からクラスタリング

- 各クラスタの平均気温平均とアメダスの座標をプロット





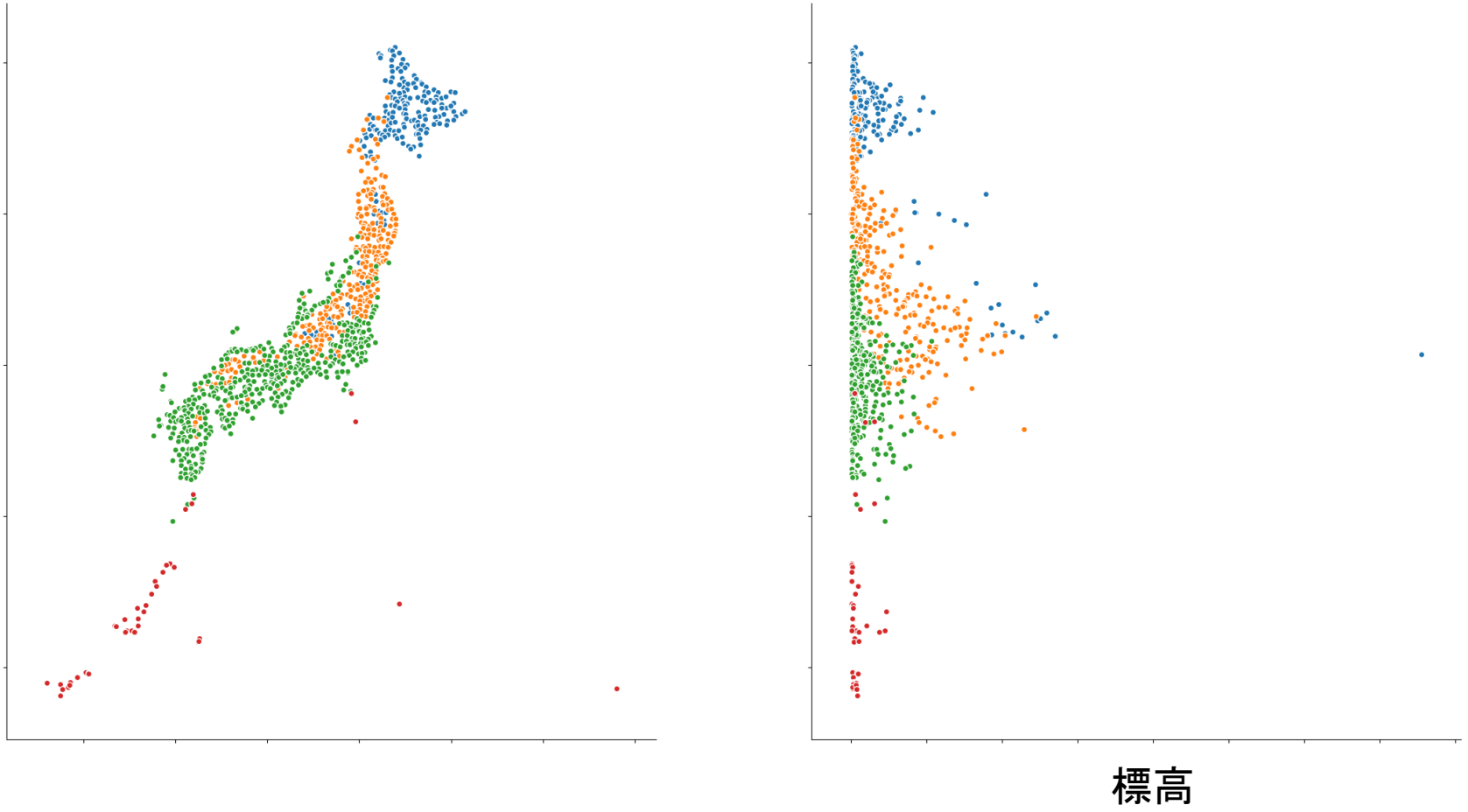
ほぼ緯度に比例して
クラスタが分けられている

平均気温の差で
クラスタが分けられている
ことが分かる

内陸と海沿いで
クラスタが分かれる
傾向が見える

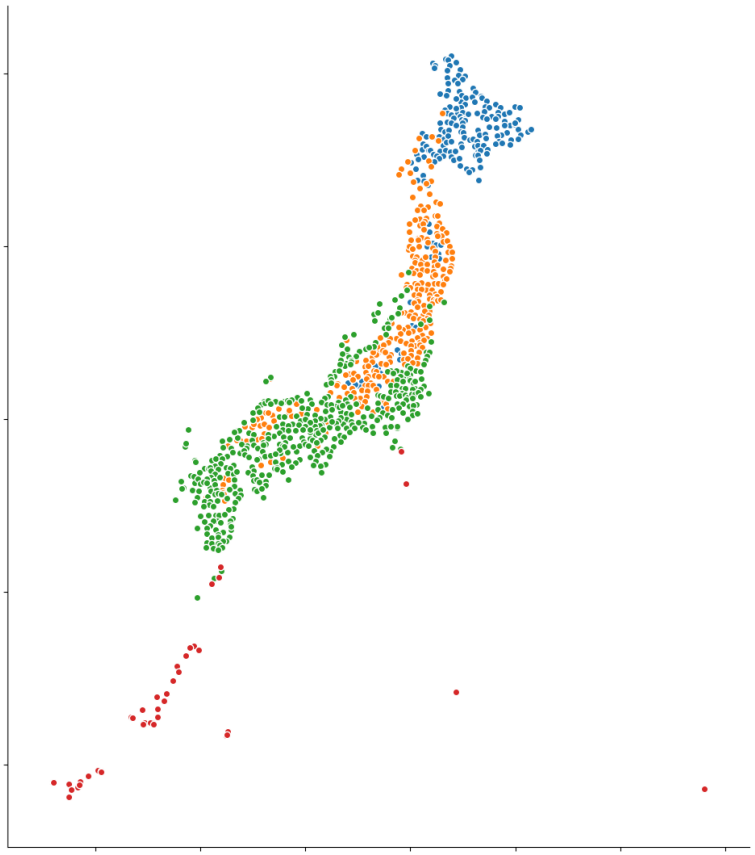
手順3：SQLite3からクラスタリング

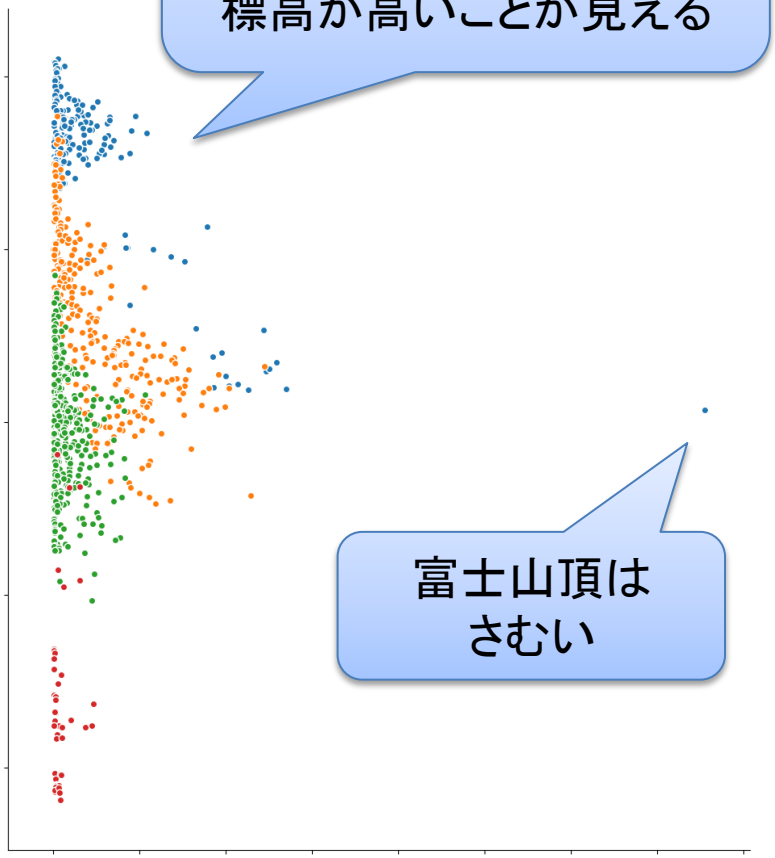
- アメダスの座標と標高をプロット



手順3：SQLite3からクラスタリング

- アメダスの座標と標高をプロット





手順4：CSVからクラスタリング

- CSVからデータを読み込んでからPythonで集計
  - 29GByteの4億行超のCSVを一度には読み込めないので，pandasパッケージで1000万行ずつ読み込む
  - （地点，月）でgroupbyして平均・最高・最低気温を計算
  - CSVが全て読み込めた時点でマージ
  - 所要時間：851秒（14分11秒）← 5回実行した平均

- K-means法で地点をクラスタリング
  - 乱数のシードを固定することで、SQLiteのときと全く同じ結果が得られた
  - 実行時間も同じ

比較結果

- SQLite vs CSV
  - SQLite：18分28秒
  - CSV ： 14分11秒

CSVのほうが30%ほど高速

- 集計の煩雑さはSQLiteほうが圧倒的にすっきり

SQLiteから読み込むPythonコード

```
with closing(sqlite3.connect(dbname)) as conn:
    c = conn.cursor()
    count_table = '''select amd_no, mn, avg(tem), max(tem),
min(tem) from tem where tem != -9999.0 group by amd_no, mn;'''
    sqlResult = c.execute(count_table)
```

CSVから読み込むPythonコード

```
reader = pd.read_csv("preAll.csv", chunksize=10000000)
firstFlag = True
for r in reader:
    r = r[['amd_no', 'mn', 'tem']]
    r = r[r.tem != -9999.0]
    grouped = r.groupby(['amd_no','mn'])
    agg = pd.concat([grouped.size(), grouped.sum(), grouped.max(), grouped.min()], axis = 1)
    agg.columns = ['size', 'sum', 'max', 'min']
    if firstFlag:
        aggData = agg
        firstFlag = False
    else:
        aggData = pd.concat([aggData, agg])
grouped = aggData.groupby(['amd_no','mn'])
aggData = pd.concat([grouped['size', 'sum'].sum(), grouped['max'].max(), grouped['min'].min()],
axis = 1)
aggData['mean'] = aggData['sum'] / aggData['size']
aggData = aggData[['mean', 'max', 'min']]
```

比較結果

- SQLite vs CSV
  - SQLite：18分28秒
  - CSV ： 14分11秒

CSVのほうが30%ほど高速

- 集計の煩雑さはSQLiteほうが圧倒的にすっきり

SQLiteから
読み込む
Pythonコード

```
with closing(sqlite3.connect(dbname)) as conn:
    c = conn.cursor()
    count_table = '''select amd_no, mn, avg(tem), max(tem),
min(tem) from tem where tem != -9999.0 group by amd_no, mn;'''
    sqlResult = c.execute(count_table)
```

CSVから読み込む
Pythonコード

```
reader = pd.read_csv("preAll.csv", chunksize=10000000)
firstFlag = True
for r in reader:
    r = r[['amd_no', 'mn', 'tem']]
    r = r[r.tem != -9999.0]
    grouped = r.groupby(['amd_no','mn'])
    agg = pd.concat([grouped.size(), grouped.sum(), grouped.max(), grouped.min()], axis = 1)
    agg.columns = ['size', 'sum', 'max', 'min']
    if firstFlag:
        aggData = agg
        firstFlag = False
    else:
        aggData = pd.concat([aggData, agg])
```

```
grouped['min'].min()],
```

管理の容易さを考えるならば,
SQLiteの採用はアリ

まとめ

- SQLiteへのインポート速度
  - 29GByte，19列，4億7400万レコードのデータで53分40秒
  - バッチで読み込むなら十分な早さ

- CSVとSQLiteの読み込み速度の比較
  - CSVのほうが30%ほど高速
  - 大きな差ではないため，データ管理の利便性を重視してSQLiteを選択することは検討の価値アリ
  - とにかく試行回数を稼ぐために少しでも早くするか，管理の容易さや移植性を重視するかに依存

AITC
先端IT活用推進コンソーシアム

クラウド活用部会への参加、
お待ちしてます
10/22（月）13:30～ 天王洲アイル
キヤノンITソリューションズ株式会社にて

ハルミン
AITC非公式イメージキャラクター

http://aitc.jp

https://www.facebook.com/aitc.jp