B-015

NYSOL: Python における大規模データ前処理支援ツール
NYSOL: Preprocessing Tool for Big Data in Python

中元 政一 † 羽室 行信 †
Masakazu Nakamoto Yukinobu Hamuro

1. まえがき

近年の機械学習/AI のブームを中心としたデータサイエンスの分野において、デファクトスタンダードのプログラミング言語として Python が利用されている。また研究分野としては深層学習などの手法に焦点が当たることが多いが、実際の現場でのデータ解析を実施する中では、データの前処理 (クリーニングや特徴抽出など) に、全体の 80%以上の時間が費やされているとの報告もあり、その効率的かつ有効な支援ツールの存在は不可欠である。本稿では、これまでに我々が開発を進めてきた大規模データ前処理ツールである Nysol(「にそる」と読む) の Python 実装である nysol.mcmd ライブラリについて紹介し、処理効率についても検討する。

2.nysol.mcmd とは

nysol.mcmd とは、我々がこれまでに開発してきた前処理を効率的かつ柔軟に実施するためのコマンドラインツールである M コマンド [1] を Python で利用できるようにしたライブラリである。M コマンドは大規模な CSV ファイルに対して、単一機能を持つコマンドをパイプラインで接続することで柔軟な処理を実現するというものである。今回、M コマンド Python ライブラリとして実装したことで、CSV ファイルだけでなく、Python のリストデータを直接処理することもできうようになっている。さらに、与えられたデータ処理フローについて、個々のメソッドは thread 単位で実行され、データは thread 間パイプを構成することで処理しており、sh 等の OS の Shell を使うことなく Python 内で完結したプログラミングを可能としている。また nysol.mcmd はデータストリーミング型の処理を行うので、入力データの規模が主記憶メモリの量に制約されることはない。コードは git hub で公開されており [2]、`pip` など一般的な方法で容易にインストールできる。

図 1 に例示されるコードは、アイテム集合列挙で用いられる Apriori アルゴリズムの一部を nysol.mcmd により実現したものである。その処理フローが図 2 に示されている。ここで想定している入力データは、表 1 に示される株価の四本値データである。

表 1. 今回の実験で利用した株価四本値データ。左から銘柄コード、日付、始値、高値、安値、終値。

id	date	o	h	l	c
8023-JP	20180101	1315.0	1333.0	1290.0	1315.0
8023-JP	20180102	1315.0	1333.0	1290.0	1315.0
8023-JP	20180103	1315.0	1333.0	1290.0	1315.0

`tra` 変数には、nysol.mcmd のメソッドである `mcut` で始める 6 つのメソッドが順次追加されていき (`<<=`演算子)、我々が「処理フローオブジェクト」と呼ぶオブジェクトが構成される。`mcut` で `id,date,c` の 3 項目を選択し、次の `mjoin` で、そのデータに対して `date` をキーにして Topix の終値を結合している。そして、`mslide` で一日ずらした終値を追加し、超過収益率 `ret` を `mcal` で計算している。最後に、超過収益率が 0.05 以上 0.1 以下の銘柄を選択し (`mselnum`)、日付をトランザクション、銘柄をアイテムとするトランザクションデータが生成される。このように、単一の機能を持ったメソッドを組み合わせることで多様な処理が実現可能となる。

```
1  import nysol.mcmd as nm
2  tra=None
3  tra <<= nm.mcut(f="id,date,c",i=iFile)
4  tra <<= nm.mjoin(k="date",m=topix,f="i")
5  tra <<= nm.mslide(k="id",s="date",f="date:date2,c:c2,i:i2")
6  tra <<= nm.mcal(c="${c2}/${c}-${i2}/${i}",a="ret")
7  tra <<= nm.mselnum(f="ret",c="[0.05,0.1]")
8  tra <<= nm.mcut(f="id,date2:date,ret")
9  freq=None
10 freq <<= nm.mcut(f="id",i=tra)
11 freq <<= nm.mcount(k="id",a="freq")
12 freq <<= nm.mselnum(f="freq",c="[5,]")
13 total=None
14 total <<= nm.mcut(f="date",i=tra)
15 total <<= nm.muniq(k="date")
16 total <<= nm.mcount(a="total")
17 coFreq=None
18 coFreq <<= nm.mcut(f="date,id",i=tra)
19 coFreq <<= nm.mcommon(k="id",m=freq)
20 coFreq <<= nm.mcombi(k="date",n=2,f="id",a="id1,id2")
21 coFreq <<= nm.mcut(f="id1,id2")
22 coFreq <<= nm.mfsort(f="id1,id2")
23 coFreq <<= nm.mcount(k="id1,id2",a="coFreq")
24 coFreq <<= nm.mjoin(k="id1",m=freq,K="id",f="freq:freq1")
25 coFreq <<= nm.mjoin(k="id2",m=freq,K="id",f="freq:freq2")
26 coFreq <<= nm.mproduct(m=total,f="total")
27 coFreq <<= nm.mcal(c="(${coFreq}*${total})/(${freq1}*${freq2
       }",a="lift")
28 coFreq <<= nm.msel(c="${lift}>2 && ${coFreq}>20")
29 r=itemCoFreq.run()
```

図 1. NYSOL.MCMD を用いた PYTHON スクリプト。株価データから、日をトランザクションとして超過収益率が高くなる時に共起する 2 銘柄を抽出している。APRIORI で用いられるアルゴリズムの一部を実装している。

その後、`tra` オブジェクトの出力データを入力にして、3 つのオブジェクト `freq,total,coFreq` が生成され、それらのオブジェクトは図 2 の矢印 a〜f で示されるように接続される。処理の接続を明示的に指定しなければ、登録順に前の処理の出力が、後の処理の入力として接続される。`i=,m=`パラメータを指定すれば、処理の接続を明示的に構成できる。例えば図 2 の接続 f は、`mjoin(m=freq)` によって `mjoin` に `freq` オブジェクトを参照データとして接続し、同時に前行の `mjoin` の出力データが暗黙に入力データとして接続される。

その後、`freq` オブジェクトでは、出現頻度が 5 以上のアイテム (銘柄) を抽出し、`total` オブジェクトでは、総トランザクション数 (日数) を計算している。そして `coFreq` オブジェクトでは、2 アイテム集合の頻度をカウントし (20-23 行)、それぞれのアイテム単独での頻度を `freq` オブジェクトの出力から結合し (24,25 行, 接続 e,f)、次に `total` で計算された総トランザクション数を結合 (26 行, 接続 g) している。そして最後に、リフト値を計算し (27 行)、最小ポートと最小リフト値で

†関西学院大学経営戦略研究科

アイテム集合を選択している (28 行)。

以上のようにして登録された処理フローの実行には、`run()` メソッドを用いる (29 行目)。各メソッドは thread 上で動作し、接続が指定されたメソッド間にパイプラインが敷設される。全ての thread は MISD(Multiple Instruction Single Data) 型の並列処理で実行される。

実行結果は、最終出力として明示的にファイル名が指定されていなければ、リストが返される。この例では、変数 `r` に、列挙された 2 アイテム集合の表が 2 重リストとして出力される。

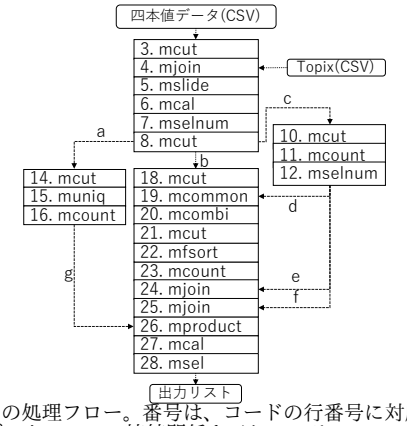

図 2. 図 1 の処理フロー。番号は、コードの行番号に対応している。矢印は、データフローの接続関係を示している。

3. ベンチマークテスト

nysol.mcmd の処理速度を評価するために、pandas をベンチマークとした処理速度実験を実施した[‡]。評価に使ったコードは、図 3 に示される通りである。データは先に紹介した株価データで、サイズとして small(29 日,11 万行,5.1MB)、mid(289 日,1.1 百万行,50.2MB)、large(8633 日,27 百万行,1072.4MB) の 3 種類用意した。計算の方法として、日別の四本値の平均値の計算 (表 2 の task=avg、コードの `pd1`,`nm1` 関数)、終値の 5 日間移動平均 (task=win、`pd2`,`nm2`)、そして一行ずつ読み込みながら Python ネイティブコードで四本値の合計する iteration 処理 (task=for,`pd3`,`nm3`)、の大きく 3 タイプの実験を実施した。平均値の計算においては、入力である株価データを日別に 8633 ファイルに分割しておき、それぞれのファイルで平均値を並列計算させる処理を参考までに実施した (`nm1a`)。また、iteration 処理については、pandas の提供するインデックス参照 `iloc` が低速であったために、`values` を用いた処理 `pd3a` の結果も掲載している。これらの実験結果を表 2 に示す。

表 2. ベンチマークテストの結果 (経過秒数)[i]。

task	program	def	small	mid	large
avg	pandas	pd1	0.130	1.28	29.18
	nysol.mcmd	nm1	0.036	0.33	**7.39**
	nysol.mcmd+multi	nm1a	-	-	5.38
win	pandas	pd2	16.91	19.22	74.88
	nysol.mcmd	nm2	**0.27**	**2.54**	**63.94**
	nysol.mcmd+file	nm2a[ii]	0.19	1.63	41.87
for	pandas	pd3	18.72	174.54	-
	pandas(values)	pd3a[iii]	0.35	3.10	73.42
	nysol.mcmd	nm3	**0.27**	**2.73**	**60.43**

[i] 実験で利用した PC: MacPro(2013),CPU: 2.7GHz 12-Core Intel Xeon E5, メモリ:64GB, ストーレッジ:USB3 HDD

[ii] 図 3 の `nm2` 関数の `writelist` を削除し、`mavg` でファイル出力したもの。

[iii] 図 3 の `pd3` 関数のインデックス参照 `iloc` を `values` に変えたもの (30,31 行目)。

[‡]https://github.com/nysol/bench にて本稿で使用したベンチマークテストのコードを公開している。

概ね nysol.mcmd は pandas より高速であるが、`pd2`(large) と `pd3a` は比較的高速である。`pd3a` のコードでは、データに null 値が含まれるために、その判定処理をしているが、それがなければ nysol.mcmd と同等性能となる。また task=win において、python のリストではなくファイルに直接出力する `nm2a` は非常に高速であり、Python のデータ型への変換に時間を要していることが推測される。

4. おわりに

本稿で紹介した機能は nysol.mcmd の一部であり、その他にも、80 以上にも及ぶ多様なメソッド、ネイティブの Python 関数を nysol.mcmd のメソッドとして登録する機能、OS コマンドをメソッドとして登録する機能、行 iterator のキーブレーク処理、など多様な機能を有しており、うまく使えば、前処理を柔軟かつ効率的に実施することが可能となるであろう。

```
1  import pandas as pd
2  import nysol.mcmd as nm
3  from glob import glob
4
5  t={'id':'str','date':'int','o':'float','h':'float','l':'
       float','c':'float'}
6  def pd1():
7    df = pd.read_csv(iFile)
8    dfg=df.groupby("date")
9    r = df_date.mean(numeric_only = True)
10 def nm1():
11   r = nm.mhashsum(k="date",f="o,h,l,c",i=iFile).run()
12 def nm1a():
13   fs=[]
14   for iFile in glob("sep/*"):
15     fs.append(nm.mhashsum(f="o,h,l,c",i=iFile))
16   r=nm.runs(fs)
17 def pd2():
18   df=pd.read_csv(iFile,dtype=t,usecols=['id','date','c'])
19   df_id=df.groupby("id", sort=False).apply(lambda x: x.
         sort_values(["date"])).reset_index(drop=True)
20   r=df_id.groupby("id", sort=False).rolling(on="date",window
         =3, min_periods=3).mean()
21 def nm2():
22   f=None
23   f <<= nm.mcut(f="id,date,c",i=iFile)
24   f <<= nm.mwindow(k="id",wk="date:win",t=3)
25   f <<= nm.mavg(k="id,win",f="c")
26   f <<= nm.writelist(dtype="win:int,date:int,c:float")
27   r=f.run()
28 def pd3():
29   df = pd.read_csv(iFile,dtype=t)
30   dfo = df.o.iloc; dfh = df.h.iloc
31   dfl = df.l.iloc; dfc = df.c.iloc
32   r=[0.0,0.0,0.0,0.0]
33   for idx in range(df.shape[0]):
34     r[0]+=dfo[idx] if not math.isnan(dfo[idx]) else 0.0
35     r[1]+=dfh[idx] if not math.isnan(dfh[idx]) else 0.0
36     r[2]+=dfl[idx] if not math.isnan(dfl[idx]) else 0.0
37     r[3]+=dfc[idx] if not math.isnan(dfc[idx]) else 0.0
38 def nm3():
39   rsl=[0.0,0.0,0.0,0.0]
40   for line in nm.mnullto(f="*",v=0,i=iFile).convtype(t2):
41     r[0]+= line[2];r[1]+= line[3]
42     r[2]+= line[4];r[3]+= line[5]
```

図 3. ベンチマークテストに利用した Python スクリプト

謝辞

本研究は、JST CREST(グラント番号: JPMJCR1401) の研究助成を受けている。

参考文献

[1] 中原孝信, 中元政一, 羽室行信, ビッグデータ解析ツール NYSOL 一性能評価, 並列処理, ビジネス応用ケース一, オペレーションズ・リサーチ, vol.61, No1, pp.11-18, 2016.

[2] https://github.com/nysol/mcmd