力学系による将来のパラメータ推定

前多啓一

2019 年 3 月 18 日

1 基本的な方針

注意 1.0.1 (方針). 以下の方針で予測を行う．与えられたデータは，n 個の観測ポイントにおける関数 $x : \mathbb{R} \to \mathbb{R}^n \quad t \mapsto (x_1, \cdots, x_n)$ の等間隔 τ の時間 $t_1, \cdots, t_m$ でのデータである．推定するのは，k 番目の特定の変数 x_k の将来での動きである．

- $(1, 2, \cdots, n)$ のなかから，L 個の数が入っているタプルを s 個選ぶ．
- l 番目のタプルから，次の値を最小化する $\psi_l : \mathbb{R}^L \to \mathbb{R}$ を推定する．(ガウス過程回帰)

$$\sum_{i=1}^{m-1} |x_k(t_{i+1}) - \psi_l(x_{l_1}(t_i), x_{l_2}(t_i), \cdots, x_{l_L}(t_i))|$$

- 各 ψ_l より 1 ステップの推定 $\tilde{x}_k^l(t+\tau) = \psi_k^l(x_{l_1}(t), \cdots, x_{l_L}(t))$ を計算する．
- 集めてできた推定の集合から，カーネル密度推定を行うことで，確率密度関数 $p(x)$ を推定する．
- 確率密度関数の歪度 γ を計算し，γ が 0.5 以下であれば採用し，$\tilde{x}_k(t+\tau) = \int xp(x)dx$ を推定として確定する．そうでなければ，以下のように推定値を修正する．交差検証によりインサンプルエラー δ_l を計算し，それに従って r 個のベストなサンプルを選び出す．

$$\tilde{x}_k(t+\tau) = \sum_{i=1}^{r} \omega_i \tilde{x}_k^{l_i}(t+\tau)$$

ここで，$\omega_i = \dfrac{\exp(-\delta_i/\delta_1)}{\sum_j \exp(-\delta_j/\delta_1)}$ である．

定義 1.0.2 (カーネル密度推定). $x_1, \cdots, x_n$ を確率密度関数 f をもつ独立同分布からの標本とする．カーネル関数 K，バンド幅 h のカーネル密度推定量とは，

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

基本的に，$K(x) = \dfrac{1}{\sqrt{2\pi}} e^{-x^2/2}$ を使う．また，最適なバンド幅として，以下の値がある．

$$h^* = \frac{c_1^{-2/5} c_2^{1/5} c_3^{-1/5}}{n^{1/5}},$$

where $c_1 = \displaystyle\int x^2 K(x)dx,\ c_2 = \int K(x)^2 dx,\ c_3 = \int (f''(x))^2 dx.$

これについてはカーネル密度推定が scipy に標準搭載されているのでそちらを援用.

2 ガウス過程回帰について

Bishop を参照 [2] しながら，ガウス過程回帰について復習する.

定義 2.0.1 (カーネル関数 (正定値カーネル)). Ω を集合とし，$k : \Omega \times \Omega \to \mathbb{R}$ を写像とする．k が Ω の正定値カーネルであるとは，次の 2 つを満たすことを言う.

(1) k は対称．すなわち，$k(x, y) = k(y, x)$ である.
(2) k の n 次元グラム行列 $(k(x_i, x_j))_{i,j}$ が半正定値である.

- 推定の仮定
  関数 $\phi : \mathbb{R}^n \to \mathbb{R}^n$ に対し，$y = \boldsymbol{w}^T \phi(\boldsymbol{x})$ とし，パラメータ $\boldsymbol{w}$ がガウス分布に従うと仮定する.
  すなわち，任意の $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n$ に対し，$\boldsymbol{y} = \Phi \boldsymbol{w}$ はガウス分布に従う．このことから，$\boldsymbol{y}$ は無限次元のガウス分布に従う，などとも言われる．ただし，$\Phi = (\phi(\boldsymbol{x}_i))_{i=1,\cdots,n}$ は計画行列である.
  $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} : (x, x') \mapsto k(x, x') = \phi(x)^T \phi(x')$ はカーネル関数である.
- 与えられるデータ (サンプル)
  $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \in \mathbb{R}^n$ および $t_1, \cdots, t_n \in \mathbb{R}$ ただし，$t_n = y_n + \varepsilon_n$ であるとする．ε_n はノイズで，ガウス分布に従うとする.
- 推定するもの
  新しい入力 x_{n+1} が与えられたときの出力 t_{n+1} の確率分布を推定する．すなわち，

$$p(t_{n+1} | \boldsymbol{x}_{n+1}, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_n, t_1, \cdots, t_n) = N(t_{n+1} | m, \sigma^2)$$

  における m と σ^2 を推定する.

定理 2.0.2. 以下のようにカーネル関数のグラム行列を定義する.

$$K = (k(\boldsymbol{x}_i, \boldsymbol{x}_j))_{i,j}$$

さらに，以下のように置く.

$$\boldsymbol{t} = \begin{pmatrix} t_1 \\ \vdots \\ t_n \end{pmatrix}, \quad \boldsymbol{k} = \begin{pmatrix} k(\boldsymbol{x}_1, \boldsymbol{x}) \\ \vdots \\ k(\boldsymbol{x}_n, \boldsymbol{x}) \end{pmatrix}$$

最適な推定は，以下の通り.

$$m = \boldsymbol{k}^T (K + \sigma_n^2 I)^{-1} \boldsymbol{t}$$
$$\sigma^2 = k(\boldsymbol{x}, \boldsymbol{x}) - \boldsymbol{k}^T (K + \sigma_n^2 I) \boldsymbol{k}$$

3 コード

以上を踏まえ，以下のようにコードを組んだ (参考 [1]). 比較のため，線形回帰よび LASSO 回帰による計算も行う.

ソースコード 1　GPRandLinearandLasso.py

```
# -*- coding: utf-8 -*-
"""
Created␣on␣Thu␣March␣7␣10:04:54␣2019

@author:␣maeta
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')
from scipy.stats import gaussian_kde
from sklearn.gaussian_process import kernels as sk_kern
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn import linear_model

#予測期間
start = 20160107
end = 20161230

similar = ['1801␣JT␣Equity',
'1802␣JT␣Equity',
'1803␣JT␣Equity',
'1812␣JT␣Equity',
'1820␣JT␣Equity',
'1821␣JT␣Equity',
'1824␣JT␣Equity',
'1833␣JT␣Equity',
'1860␣JT␣Equity',
'1893␣JT␣Equity']

# 推定のために 1日リターンに
df_open = pd.read_csv('Open.csv',usecols =similar)
df_close = pd.read_csv('Close.csv',usecols =similar)
df = (df_close - df_open) /df_open

date = pd.read_csv('Date.csv')
po_start = int(date[date['Date']==start]['No'])
po_end = int(date[date['Date']==end]['No'])
index = date['Index'][po_start:po_end+1]

##GPR による方法
# df の similar に入っている株を，p 個ペアで q 回ずつ，n の深さで予想する.
class GPRestimation:
    def __init__(self, df, similar, n, p, q):
        self.df = df
        self.similar = similar
        self.n = n

        self.p = p
        self.q = q

    # リストを,ある値との距離順に並べる関数
    def pointsort(self,arr,val):
        return [y[1] for y in sorted([(abs(x-val),x) for x in arr])]

```

```
57      # random に n 個の配列を作る
58      def pickup(self,arr,n):
59          arr2 = arr[:]
60          result = []
61          for i in range(n):
62              x = arr2[int(len(arr2) * np.random.rand())]
63              result.append(x)
64              arr2.remove(x)
65          return result
66
67      # a を含まないランダムな配列を n 個作る
68      def pickup2(self,arr,a,n):
69          arr2 = arr[:]
70          arr2.remove(a)
71          return self.pickup(arr2,n-1)
72
73      # GPR
74      def GPR_fit(self,x_train,y_train,x_test):
75          kernel = sk_kern.RBF(1.0, (1e-3, 1e3)) + sk_kern.ConstantKernel(1.0, (1e-3, 1e3)) +
                sk_kern.WhiteKernel()
76          clf = GaussianProcessRegressor(
77              kernel=kernel,
78              alpha=1e-10,
79              optimizer="fmin_l_bfgs_b",
80              n_restarts_optimizer=20,
81              normalize_y=True)
82          clf.fit(x_train,y_train)
83          pred_mean, pred_std = clf.predict(x_test, return_std=True)
84          return pred_mean,pred_std
85
86      # 期待値を算出する関数
87      def expectation(self,x,y):
88          y = y / sum(y)
89          return sum(x*y)
90
91      # kernel density estimation をしたのち，期待値を算出する
92      def kde_process(self,data_list):
93          kde_model = gaussian_kde(data_list)
94          y = kde_model(data_list)
95          skew = pd.Series(y).skew()
96          if abs(skew) < 0.1:
97              return self.expectation(data_list,y)
98          else:
99              data_list2=self.pointsort(data_list,np.average(data_list))[0:int(len(data_list
                    )/3)]
100             return self.expectation(data_list2,kde_model(data_list2))
101
102     def onetimeestimation(self,i,terget):
103         # terget の情報（1 つだけずらして取得）
104         y_train = self.df[terget].values[i-self.n:i]
105         result = np.array([])
106         result_sd = np.array([])
107         j = 0
108         while j < self.q:
109             try:
110                 x = self.df[self.pickup2(self.similar, terget, self.p)].values
111                 x_train,x_test = x[i-self.n:i],[x[i]]
```

```
112                 pred_y ,pred_y_sd = self.GPR_fit(x_train,y_train,x_test)
113                 result = np.append(result,pred_y)
114                 result_sd = np.append(result_sd,pred_y_sd)
115                 j += 1
116             except:
117                 j += 1
118         mean = self.kde_process(result)
119         err = (self.df[terget].values[i+1]-mean)**2
120         sd = np.average(result_sd)
121         return mean,sd,err
122 
123     # 同業種リストsimilar のすべての 1 ステップを推定する関数
124     def onetimeallestimation(self,i):
125         result = []
126         result_sd = []
127         err = 0
128         for terget in self.similar:
129             result.append(self.onetimeestimation(i,terget)[0])
130             result_sd.append(self.onetimeestimation(i,terget)[1])
131             err += self.onetimeestimation(i,terget)[2]
132         return result,result_sd,err
133 
134 ## 線形回帰による方法
135 class Linearestimation:
136     def __init__(self, df, similar, n):
137         self.df=df
138         self.similar = similar
139         self.n = n
140 
141     def linearregression(self,train_x,train_y,test_x):
142         clf = linear_model.LinearRegression()
143         clf.fit(train_x,train_y)
144         return np.dot(clf.coef_,test_x)
145 
146     def linearestimation(self,terget,n,i):
147         train = self.similar[:]
148         train.remove(terget)
149         train_x = self.df[train].values[i-n:i]
150         train_y = self.df[terget].values[i-n:i]
151         test_x = self.df[train].values[i:i+1][0]
152         result = self.linearregression(train_x,train_y,test_x)
153         err = (result - self.df[terget].values[i+1])**2
154         return result, err
155 
156     def linearonetimeallestimation(self,i):
157         result = np.array([])
158         err = 0
159         for terget in similar:
160             result = np.append(result,self.linearestimation(terget,self.n,i)[0])
161             err += self.linearestimation(terget,self.n,i)[1]
162         return result,err
163 
164 ## LASSO による方法
165 class Lassoestimation:
166     def __init__(self, df, similar, n):
167         self.df = df
168         self.similar = similar
```

```
169         self.n = n
170 
171     def lassoregression(self,train_x,train_y,test_x):
172         lasso = linear_model.Lasso(alpha=0.00001)
173         lasso.fit(train_x,train_y)
174         return np.dot(lasso.coef_,test_x)
175 
176     def lassoestimation(self,terget,n,i):
177         train = self.similar[:]
178         train.remove(terget)
179         train_x = self.df[train].values[i-n:i]
180         train_y = self.df[terget].values[i-n:i]
181         test_x = self.df[train].values[i:i+1][0]
182         result = self.lassoregression(train_x,train_y,test_x)
183         err = (result - self.df[terget].values[i+1])**2
184         return result, err
185 
186     def lassoonetimeallestimation(self,i):
187         result = np.array([])
188         err = 0
189         for terget in similar:
190             result = np.append(result,self.lassoestimation(terget,self.n,i)[0])
191             err += self.lassoestimation(terget,self.n,i)[1]
192         print(err)
193         return result,err
194 
195 # GPRcheck
196 PortRet = []
197 GPR = GPRestimation(df,similar,10,3,5)
198 err = 0
199 for i in range(po_start,po_end+1):
200     tmpdf = df[i:i+1]
201     result = GPR.onetimeallestimation(i)[0]
202     result_sd = GPR.onetimeallestimation(i)[1]
203     err += GPR.onetimeallestimation(i)[2]
204     long = similar[np.argmax(result)]
205     short = similar[np.argmin(result)]
206     portreturn = tmpdf[long].values[-1] - tmpdf[short].values[-1]
207     PortRet.append(portreturn)
208     print(str(i)+"...finished")
209     print(err)
210 PortRet = pd.DataFrame(PortRet)
211 PortRet.index = index
212 PortRet.columns = ['Ret-GPR']
213 PortRet.to_csv('Result.csv')
214 
215 # linearcheck
216 PortRet = []
217 Line = Linearestimation(df,similar,10)
218 err = 0
219 for i in range(po_start,po_end+1):
220     tmpdf = df[i-11:i]
221     result = Line.linearonetimeallestimation(i)[0]
222     err += Line.linearonetimeallestimation(i)[1]
223     long = similar[np.argmax(result)]
224     short = similar[np.argmin(result)]
225     portreturn = tmpdf[long].values[-1] - tmpdf[short].values[-1]
```

```
    PortRet.append(portreturn)
    print(str(i)+"...finished")
    print(err)
print(err)
PortRet = pd.DataFrame(PortRet)
PortRet.index = index
PortRet.columns = ['Ret-Linear']
PortRet.to_csv('Result-Linear.csv')

# Lassocheck
PortRet = []
Lasso = Lassoestimation(df,similar,10)
err = 0
for i in range(po_start,po_end+1):
    tmpdf = df[i-11:i]
    result = Lasso.lassoonetimeallestimation(i)[0]
    err += Lasso.lassoonetimeallestimation(i)[1]
    long = similar[np.argmax(result)]
    short = similar[np.argmin(result)]
    portreturn = tmpdf[long].values[-1] - tmpdf[short].values[-1]
    PortRet.append(portreturn)
    print(str(i)+"...finished")
    print(err)
PortRet = pd.DataFrame(PortRet)
PortRet.index = index
PortRet.columns = ['Ret-Lasso']
PortRet.to_csv('Result-Lasso.csv')
```

4 結果

以下の同業種の株で推定を行った.

group1：業種グループ:Engineering　業種サブグループ:Building&Construc-Misc
group2：業種グループ:BasicMaterials　業種サブグループ:Chemicals
group3：業種グループ:Consumer,Noncyclical　業種サブグループ:food
group4：業種グループ:Machinery diversified　業種サブグループ:Machinery-generalindust

	Linear	Lasso	GPR
group1	0.75427	0.75644	0.6943
group2	3.2203	2.6910	2.0573
group3	1.7092	0.5125	0.3745
group4	0.6737	0.5925	0.51308

表 1　二乗誤差の比較

参考文献

[1] Qiita PRML 第 6 章 ガウス過程による回帰 Python 実装 https://qiita.com/ctgk/items/

4c4607edf15072cddc46

[2] Christopher M. Bishop “Pattern Recognition and Machine Learning” 2013