Artificial Intelligence

AI-04: linear_reg_adv.py の解説

brute_force_aic()

model: 回帰用のモデル。今回は線形重回帰なので、smf.ols が代入される。
exog: 説明変数のリスト
endog: 目的変数 (今の場合は1つだけ) のリスト
**kwards: model.fit() に与える引数

```
def brute_force_aic(model, exog, endog, **kwargs):
    min_score = None
    min_selected = None
    formula_head = ' + '.join(endog) + ' ~ '
    exog_ar = np.array(exog)
    n = len(exog_ar)
    …
```

min_score: それまでに見つけた最良の(=AICが最小の) 式のAICの値。

min_selected: それまでに見つけた最良の(=AICが最小の) 式の説明変数のリスト。

式は、「目的変数 ~ 説明変数1 + 説明変数2 + …」という文字列で与える。formula_head はその先頭の「目的変数 ~ 」部分。
なお、目的変数が複数の場合は、formula_headは「目的変数1 + 目的変数2 + … ~ 」となる。

exog_ar: 説明変数のリストをndarrayに変換。
n: 説明変数の数

n個の説明変数のあらゆる組み合わせを調べる。
このため、i を 1から 2**n -1 まで動かし、i を2進数のビット化したものを nb とする。

nb:　0 0 … 0 0 1 ← 0 番目の説明変数のみ使用
　　0 0 … 0 1 0 ← 1 番目の説明変数のみ使用
　　0 0 … 0 1 1 ← 0, 1番目の説明変数を使用
　　…
　　1 1 … 1 1 1 ← すべての説明変数を使用

```
…
for i in range(1, 2**n):
    ib = str(bin(i))[2:]; m = len(ib)
    nb = np.array(list('0'*(n-m)+ib))
    selected = exog_ar[ nb=='1' ]
    formula_tail = ' + '.join(selected)
    formula = formula_head + formula_tail
    aic = model(formula=formula, **kwargs).fit().aic
    print('AIC: {:.3f}, formula: {}'.format(aic, formula))
    …
```

nb の1に対応する説明変数のみを取り出したリストを selected とする

式「目的変数 ~ 説明変数1 + 説明変数2 + …」の右辺部分が formula_tail

今の説明変数の組み合わせを用いた式 formula で線形重回帰を行ったときの AIC を aic に取り出し、式とともに表示。

```
        …
        if min_score == None or min_score > aic:
                min_score = aic
                min_selected = selected.copy()

    formula = formula_head + ' + '.join(min_selected)
    print('The best formula: {}'.format(formula))
    print('Minimum AIC: {:.3f}'.format(min_score))
    return model(formula, **kwargs).fit()
```

最初の式またはこれまで出てきた中で最小の aic の式で使った説明変数たち (selected) は aic値とともに保存。

forループが終了したら、最小 aic の式を formula として aic値とともに表示し、そのときの線形重回帰の結果を return する。

実行例

abalone_modified.csv の 'len', 'd', 'h', 'w_all' 列を説明変数、
'ring' を目的変数とした例

```
results = brute_force_aic(smf.ols, exog, endog, data=df_scaled)
```

```
AIC: 10406.185, formula: ring ~ w_all
AIC: 10293.403, formula: ring ~ h
AIC: 10164.339, formula: ring ~ h + w_all
AIC: 10175.294, formula: ring ~ d
AIC: 10173.998, formula: ring ~ d + w_all
AIC: 10048.947, formula: ring ~ d + h
AIC: 10050.341, formula: ring ~ d + h + w_all
AIC: 10299.876, formula: ring ~ len
AIC: 10274.645, formula: ring ~ len + w_all
AIC: 10117.269, formula: ring ~ len + h
AIC: 10115.510, formula: ring ~ len + h + w_all
AIC: 10151.690, formula: ring ~ len + d
AIC: 10145.454, formula: ring ~ len + d + w_all
AIC: 10017.609, formula: ring ~ len + d + h
AIC: 10019.510, formula: ring ~ len + d + h + w_all
The best formula: ring ~ len + d + h
Minimum AIC: 10017.609
```

全組み合わせを一気に試して、
もっとも小さな AIC の式を出力

最適なのは ring ~ len + d + h (AIC: 10017.609)

step_aic()

step_aic_forward() は、step_aic() で引数direction が 'forward' の場合に対応。

model: 回帰用のモデル。今回は線形重回帰なので、smf.ols が代入される。
exog: 説明変数のリスト
endog: 目的変数 (今の場合は1つだけ) のリスト
direction: 変数増加法なら 'forward', 変数減少法なら 'backward'
**kwards: model.fit() に与える引数

```
def step_aic(model, exog, endog, direction='forward', **kwargs):
    exog = np.r_[[exog]].flatten()
    endog = np.r_[[endog]].flatten()
    remaining = set(exog)
    selected = []  # Selected exogenous variables
    …
```

引数の exog, endog がどんな形状かわからないので、
ともかく 1-d ndarray に変換

変数増加法の場合:
・まだ使っていない説明変数の集合が remaining (初期値は全説明変数)。
・選択した説明変数のリストが selected (初期値は空)。

変数減少法の場合:
・まだ残っている説明変数の集合が remaining (初期値は全説明変数)。
・選択した説明変数のリストが selected (初期値は空)。

```
    formula_head = ' + '.join(endog) + ' ~ '
    if direction == 'forward':
        formula = formula_head + '1'
    elif direction == 'backward':
        formula = formula_head + ' + '.join(remaining)
        selected = remaining.copy()
    aic = model(formula=formula, **kwargs).fit().aic
    print('AIC: {:.3f}, formula: {}'.format(aic, formula))

    current_score = aic

    ...
```

まず、
step-forward: 定数項だけの式を formula に作る (「目的変数 ~ 1」)。
step-backward: 全説明変数を使った式を formula に作り、selected を全説明変数にする

このときの AIC を計算し、変数 aic に格納、式とともに表示。

現在の最良の (=AICが最小の) 式のスコア (current_score) を aic にしておく。

```
...
while True:
    score_with_candidates = []
    for candidate in remaining:
        # Calculate AIC for adding an exog one by one
        if direction == 'forward':
            formula_tail = ' + '.join(selected + [candidate])
        elif direction == 'backward':
            picked = remaining.copy()
            picked.remove(candidate)
            formula_tail = ' + '.join(picked)
        formula = formula_head + formula_tail
        aic = model(formula=formula, **kwargs).fit().aic
        print('AIC: {:.3f}, formula: {}'.format(aic, formula))

        score_with_candidates.append((aic, candidate))
    ...
```

score_with_candidates は、(aic, 説明変数1つ) をメンバーとするリスト。

remaining から1つずつ説明変数を candidateに取り出す。

変数増加法の場合:
確定した selected に candidate を加えて formula_tail を作成。

変数減少法の場合:
残っている remaining から candidate を削除して formula_tail を作成

AICを計算して変数 aic に格納、対応する式とともに表示。
変数増加法では (aic, そのときに加えた説明変数1つ)
変数減少法では (aic, そのときに削除した説明変数1つ) を score_with_candidates に append。

```
…
# Select best_candidate with minimum AIC
 score_with_candidates.sort()
 best_score, best_candidate = score_with_candidates[0]

 # select best_candidate if AIC is improved
 improved = False
 if best_score < current_score:
     remaining.remove(best_candidate)
     if direction == 'forward':
         selected.append(best_candidate)
     else:
         selected = remaining.copy()
     current_score = best_score
     improved = True

 if not remaining or not improved: break
 …
```

score_with_candidates をソート。
aic の小さい順に並ぶ。

一番小さかった aic を best_score、そのときに加えた
(変数増加法の場合) または削除した (変数減少法の場合)
変数1つを best_candidate に、それぞれ格納。

best_score が current_score (現在の最小値) を下回っていたら、remaining からそのときの説明変数
(best_candidate) を削除した上で、
変数増加法では、best_candidate を selected に追加。
(式に追加することが決定)
変数減少法では、remaining を selected にコピー。
(式から除くことが決定)
そして、improved の値を True にする。

残っている説明変数がない または 変数の追加/削除 をしても AICの最小値が更新されなかったら、while True を break で抜ける。

while True が終了したら …

```
…
formula = formula_head + ' + '.join(selected)
print('Direction:', direction)
print('The best formula: {}'.format(formula))
aic = model(formula=formula, **kwargs).fit().aic
print('Minimum AIC: {:.3f}'.format(aic))
return model(formula, **kwargs).fit()
```

最小 aic の式を formula として aic値とともに表示し、
そのときの線形重回帰の結果を return する。

実行例 (forward)

abalone_modified.csv の 'len', 'd', 'h', 'w_all' 列を説明変数、
'ring' を目的変数とした例

```
results = step_aic(smf.ols, exog, endog, data=df_scaled, direction='forward')
```

```
AIC: 11847.299, formula: ring ~ 1
AIC: 10299.876, formula: ring ~ len
AIC: 10293.403, formula: ring ~ h
AIC: 10406.185, formula: ring ~ w_all
AIC: 10175.294, formula: ring ~ d
AIC: 10151.690, formula: ring ~ d + len
AIC: 10048.947, formula: ring ~ d + h
AIC: 10173.998, formula: ring ~ d + w_all
AIC: 10017.609, formula: ring ~ d + h + len
AIC: 10050.341, formula: ring ~ d + h + w_all
AIC: 10019.510, formula: ring ~ d + h + len + w_all
Direction: forward
The best formula: ring ~ d + h + len
Minimum AIC: 10017.609
```

まず定数項のみ

1つだけ試してみる → d の場合にこれまでの best なので確定

d に1つ加えてみる → hの場合にこれまでの best なので確定

d + h に1つ加えてみる → len の場合にこれまでの best なので確定

d + h + len に 残った w_all を加えてみる
→ これまでの best を更新できなかった。またこれで説明変数が尽きたため終了

最適なのは ring ~ d + h + len (AIC: 10017.609)

実行例 (backward)

abalone_modified.csv の 'len', 'd', 'h', 'w_all' 列を説明変数、
'ring' を目的変数とした例

```
results = step_aic(smf.ols, exog, endog, data=df_scaled, direction='backward')
```

AIC: 10019.510, formula: ring ~ h + len + w_all + d　まず全説明変数 (h + len + w_all + d)
AIC: 10145.454, formula: ring ~ len + w_all + d
AIC: 10050.341, formula: ring ~ h + w_all + d
AIC: 10017.609, formula: ring ~ h + len + d
AIC: 10115.510, formula: ring ~ h + len + w_all

1つだけ削除してみる
→ w_all の場合に (つまり h + len + d) これまでの best なので確定

AIC: 10151.690, formula: ring ~ d + len
AIC: 10048.947, formula: ring ~ d + h
AIC: 10117.269, formula: ring ~ len + h

h + len + d からさらに1つ削除してみる
→これまでの best を更新できなかったので終了

Direction: backward
The best formula: ring ~ d + len + h
Minimum AIC: 10017.609

最適なのは ring ~ d + len + h (AIC: 10017.609)