Python を用いた 部分最小二乗（PLS）回帰

森　田　成　昭

1　はじめに

本稿ではプログラミング言語の Python[1)2)] を用いて部分最小二乗（partial least squares, PLS）回帰[3)]の計算をする方法を紹介する．種々の機器分析において，着目物質の量に比例する信号を一つ選定し，単回帰により定量分析を行ったことがある人は多いだろう．このとき，夾雑（きょうざつ）物の影響などによって定量分析がうまくいかないことがあるが，そのときは単回帰ではなく，機器分析データを多変量データと捉えて重回帰[4)]を試すとよい．

重回帰の基礎となる計算は線形重回帰（multiple linear regression, MLR）であるが，機器分析データの場合，波長や溶出時間といった説明変数の数が，回帰にあてはめるサンプルの数よりも多くなりやすく，多重共線性や過学習（オーバーフィッティング）の影響を受けてうまくいかないことがある．これを回避する方法として，主成分分析（principal component analysis, PCA）のような次元削減を行い，説明変数の数をサンプルの数よりも少なくすることが有効である．PLS 回帰の計算にも次元削減が含まれており，分光分析において比較的ロバストに定量分析ができることから，汎用的に使われてきた実績がある．

多変量データを使って回帰の計算をする方法には他にも support vector machine（SVM）など，いろいろとあるが，Python の機械学習ライブラリである scikit-learn[5)] の使い方がわかってくると書き換えは容易である．また，定性分析に使われる線形判別分析（linear discriminant analysis, LDA）のようなクラス分類の学習器も scikit-learn に含まれており，Python を使いこなすことで典型的なケモメトリックス[6)~9)]の計算ができるようになる．以前に機器分析データを用いて Python で PCA の計算を行う方法を紹介したが[10)]，本稿では次のステップとして PLS 回帰の計算方法を解説する．

以降で紹介する Python のサンプルコードは，統合環境である Anaconda[11)] をインストールしたコンピュータの Jupyter Notebook[12)] で動作することを確認した．ハッシュ（#）から始まるコメント行を除くと 50 行しかないサンプルコードなので，手を動かしながら PLS 回帰の計算を修得してほしい．

2　データセット

以降では公開されている軽油の近赤外スペクトルを用いて，ディーゼルエンジンでの着火性の指標であるセタン価を回帰する．用いるデータセットは https://eigenvector.com/resources/data-sets/ からダウンロードできる．データセットは Matlab フォーマットと csv フォーマットから選べるが，今回は csv フォーマットのデータを使うことにする．スペクトルデータは diesel_spec.csv に，物性値データは diesel_prop.csv に書き込まれているので，図 1 のようにして読み込み，pandas.DataFrame オブジェクトである data に変換しておく．

ここで 2～4 行は必要なライブラリの読み込みを行っている．5～6 行では pandas.read_csv 関数を用いてスペクトルデータ diesel_spec.csv と物性値データ diesel_prop.csv を pandas.DataFrame オブジェクトとして読み込み，それぞれの変数名を spec と prop とした．7～9 行では spec と prop をまとめて，新たに data という変数名の pandas.DataFrame オブジェクトをつくり，data.

```
001 # データの読み込み
002 import numpy
003 import pandas
004 from matplotlib import pyplot
005 spec = pandas.read_csv("diesel_spec.csv", header=9, index_col=1)
006 prop = pandas.read_csv("diesel_prop.csv", header=8, index_col=1)
007 data = spec.iloc[:, 1:-1]
008 data.index = prop.iloc[:, 2].values
009 data.columns = data.columns.astype(int)
010 data = data[data.index.notna()]
011 display(data)
```

	750	752	754	756	758	760	762	764	766	768	...
55.1	-0.028073	-0.025056	-0.020949	-0.016544	-0.011938	-0.007299	-0.003553	-0.002041	-0.001611	-0.001701	...
46.5	-0.024340	-0.021221	-0.016691	-0.011428	-0.006280	-0.000757	0.002302	0.002651	0.002176	0.001553	...
53.6	-0.021778	-0.018382	-0.014348	-0.010099	-0.005716	-0.001257	0.002011	0.003679	0.004023	0.003941	...
45.0	-0.022484	-0.019083	-0.014823	-0.010395	-0.006021	-0.001663	0.001402	0.002746	0.002786	0.002084	...
45.8	-0.005264	-0.002123	0.001831	0.006170	0.010394	0.014847	0.018028	0.019488	0.019644	0.019701	...
...	...	...	...	...	...	...	...	...	...	...	...
51.2	-0.027873	-0.024856	-0.020544	-0.016217	-0.010540	-0.005448	-0.002336	-0.001839	-0.002849	-0.003495	...
50.9	-0.027034	-0.024048	-0.019643	-0.015469	-0.009928	-0.004994	-0.001675	-0.000624	-0.001729	-0.002087	...
51.5	-0.026734	-0.023591	-0.019396	-0.014691	-0.009706	-0.004795	-0.001207	-0.000346	-0.001298	-0.001923	...
50.6	-0.027399	-0.024747	-0.020104	-0.016381	-0.010736	-0.005670	-0.002499	-0.001968	-0.002908	-0.003550	...
50.1	-0.027894	-0.024818	-0.020525	-0.016340	-0.011272	-0.006216	-0.002493	-0.001425	-0.002270	-0.002796	...

381 rows × 401 columns

図 1　データの読み込み

Partial Least Squares（PLS）Regression in Python.

index が目的変数であるセタン価，data.columns が説明変数である波長，data.values がスペクトルデータの吸光度となるようにした．10 行では，セタン価に欠損値があったので，欠損値となっているセタン価と，対応するスペクトルデータを削除した．11 行は data の確認であり，目的変数（セタン価）が 381 個，説明変数（波長）が 401 個である横長の行列が得られている．以降で自身のデータセットを用いて解析を行う場合は，同様に，data.index が目的変数，data.columns が説明変数，data.values が多変量データとなるように pnadas.DatFrame オブジェクトである data を準備すればよい．

3　データの前処理

図 2 は，図 1 で読み込んだ 381 本の近赤外スペクトルをプロットした結果である．横軸は波長であり，960−1120 nm の領域と 1200−1400 nm の領域にそれぞれ，特徴的な近赤外吸収の信号がみられる．ここでは 960−1120 nm の領域だけを選んで PLS 回帰を行ってみることにする．

図 3 は，横軸を 960−1120 nm の範囲で指定して，pandas.DataFrame オブジェクトである data を上書きし，再度プロットした結果である．横軸の範囲指定は 16 行のように，data.columns が 960 以上 1120 以下である列を抽出することで行った．これをみると，この領域にいくつかの近赤外吸収ピークがあり，それらの強度がサンプルによって変化しているのがわかる．しかし，ベースライン強度の変動も大きく，解析に影響することが予想される．

そこで図 4 のように，フィルターの窓内でデータを多項式に近似する Savitzky-Golay フィルターを用いて二次微分スペクトルを計算した．Python で Savitzky-Golay フィルターの計算をするには scipy.signal.savgol_filter 関数を用いればよい．20 行のように savgol_filter 関数をライブラリから読み込んでおき，21 行のようにいくつかの引数を指定する．引数の指定方法は，scipy.signal.savgol_filter をインターネット検索すると SciPy のオンラインマニュアルが見つかるし，あるいは ChatGPT のようなチャットボットを用いて対話的に教えてもらってもよい．チャットボットを活用するなら「Python で Savitzky-Golay 微分をしたい」くらいから始めてみよう．

今回は 21 行にあるように，savgol_filter 関数に四つの引数を指定した．順に説明すると，一つ目は入力データ，二つ目はフィルターの窓の大きさ，三つ目は近似する多項式の次数，四つ目は微分の次数である．すなわち，入力データとして pandas.DataFrame オブジェクトである data を指定し，フィルターの窓の大きさを 9，近似する多項式を二次関数，微分の次数を二次微分と指定した．

フィルターの窓の大きさが 9 ということは，ここでは横軸の波長間隔が 2 nm なので，ある波長点での二次微分強度を計算するのに，左側 4 点×2 nm と右側 4 点×2 nm で合計 16 nm の窓を用いて元データを多項式に近似し，その多項式を微分したということである．離散データの平滑化と同様に，窓の大きさが大きいと小さな信号を消してしまい，逆に小さいとノイズの影響を受け

```
012 # データのプロット
013 data.T.plot(legend=None)
014 pyplot.show()
```

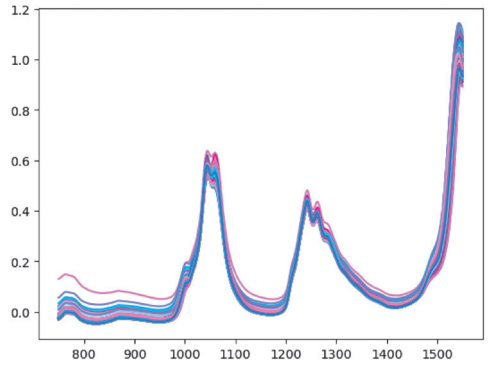

図 2　データのプロット

```
015 # 横軸の範囲指定
016 data = data.iloc[:, (960 <= data.columns) & (data.columns <= 1120)]
017 data.T.plot(legend=None)
018 pyplot.show()
```

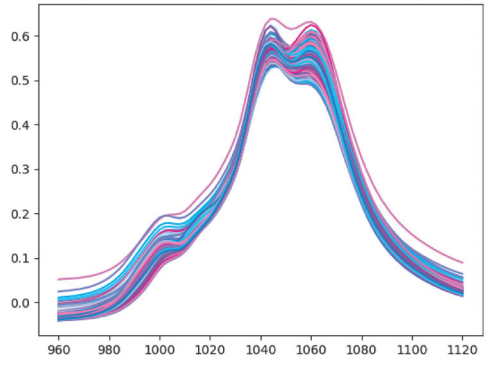

図 3　横軸の範囲指定

```
019 # 二次微分
020 from scipy.signal import savgol_filter
021 data = pandas.DataFrame(savgol_filter(data, 9, 2, 2),
    index=data.index, columns=data.columns)
022 data.T.plot(legend=None)
023 pyplot.show()
```

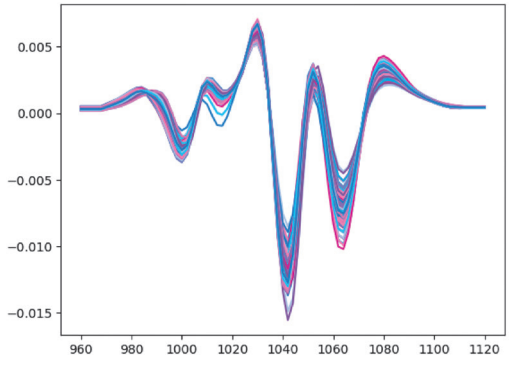

図 4　二次微分

てしまうので，窓の大きさは慎重にチューニングする必要がある．

二次微分スペクトルの計算結果（図 4）をみると，ベースライン強度の変動が抑えられており，また，下向きの四つのピークは，負の強度がサンプルによって異なっているのがわかる．以降ではこの二次微分スペクトルを用いて回帰の計算を行うために，21 行で data を二次微分スペクトルで上書きした．

4　回帰モデルのチューニング

今回は 381 本のスペクトルデータを用いて回帰を行うが，そのすべてを回帰モデルの構築に使ってしまうと，新たなスペクトルデータが得られたときにロバストな回帰が行えるかを検証できなくなってしまう．そこで一般に，全データセットを，モデル構築用のトレーニングセットと，モデル検証用のテストセットの二つに分けておき，トレーニングセットで回帰モデリング（キャリブレーション）を，テストセットで得られた回帰モデルの検証（バリデーション）を行う．

Python の機械学習ライブラリである scikit-learn には，データセットをトレーニングセットとテストセットに分けるための sklearn.model_selection.train_test_split 関数が準備されている．ここでは 25 行で train_test_split 関数をライブラリから読み込んでおき，26 行で train_size=0.6 と指定することで，データセットである data を 60 % と 40 % の割合で分割し，トレーニングセットを train，テストセットを test という変数名でメモリに書き込んだ．最後の引数を random_state=12 としたが，これによりデータセットをランダムに分割するときの乱数の初期値を指定している．このときの data，train，test にそれぞれ割り振られたデータのセタン価を 27 行でバイオリンプロットした．図 5 をみると，テストセットのセタン価の分布がトレーニングセットのセタン価の分布より狭くなっているのがわかる．もしテストセットの目的変数の分布がトレーニングセットのそれよりも広くなっているときは，モデル構築時に学習していない範囲を予測しなければならなくなるので，予測誤差が大きくなることが予想される．データセットの分割は，ここで示したように，テストセットの目的変数の分布がトレーニングセットの分布よりも狭くなるように random_state の値を適切に選んで指定しておくとよい．

Python で PLS 回帰の計算をするには sklearn.cross_decomposition.PLSRegression クラスを用いればよい．ここでは 30 行で PLSRegression クラスをライブラリから読み込んでいる．PLS 回帰におけるハイパーパラメータは次元削減の成分数だけである．例えば成分数を 5 に固定して Y を X で PLS 回帰するには PLSRegression(5).fit(X, Y) とすればよい．今回は X に二次微分スペクトルを，Y にセタン価を指定する．

次にハイパーパラメータのチューニングについて説明する．今回はハイパーパラメータが一つだけなので 1 次元のグリッドサーチを行う．グリッドサーチを行うには sklearn.model_selection.GridSearchCV クラスを用いればよい．ここでは 31 行で GridSearchCV クラスをライブラリから読み込んでいる．32 行で，1 から 20 まで 1 ずつ増加する 1 次元配列 p1 を準備し，32 行で PLS 回帰の成分数 n_components の値が p1 である辞書型の変数 parm を準備した．これを使って 1 次元グリッドサーチを実行するには 34 行のように書けばよい．ここで cv は *k*-fold クロスバリデーションの分割数であり，今回は cv=5 と指定することで 5-fold クロスバリデーションを行った．グリッドサーチの結果は変数 search に格納し，35 行でその結果を表示した．今回は成分数 8 でク

```
024 # データのスプリット
025 from sklearn.model_selection import train_test_split
026 train, test = train_test_split(data, train_size=0.6,
    random_state=12)
027 pyplot.violinplot([data.index, train.index, test.index])
028 pyplot.show()
```

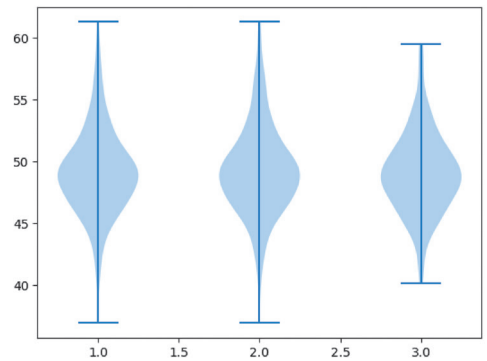

図 5　データのスプリット
左からそれぞれ，全データセット，トレーニングセット，テストセットにおける目的変数（セタン価）のバイオリンプロット．

```
029 # グリッドサーチによるハイパーパラメータの決定
030 from sklearn.cross_decomposition import PLSRegression
031 from sklearn.model_selection import GridSearchCV
032 p1 = numpy.arange(1, 21, 1)
033 parm = {"n_components": p1}
034 search = GridSearchCV(PLSRegression(), parm,
    cv=5).fit(data.values, data.index)
035 print(search.best_estimator_, search.best_score_)
036 pyplot.scatter(p1, search.cv_results_["mean_test_score"])
037 pyplot.show()
038 model = search.best_estimator_.fit(train.values, train.index)
PLSRegression(n_components=8) 0.5048790652445199
```

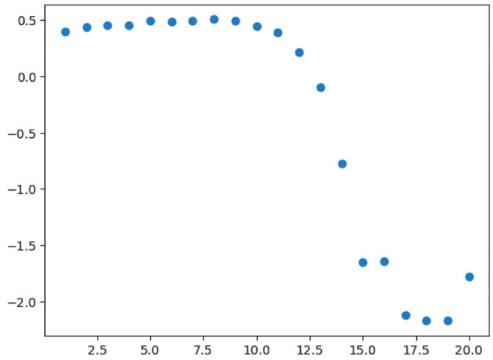

図 6　グリッドサーチによるハイパーパラメータの決定

ロスバリデーションのスコア（決定係数の平均）が最大値 0.504… となっている．36〜37 行は横軸を成分数，縦軸をクロスバリデーションのスコアとしたプロットであり，図 6 を見ると成分数 8 までは緩やかにスコアが増加し，それ以降は急激に減少していることから，成分数 9 以上で過学習が起こっていると判断できる．39 行ではグリッドサーチで得られた最適なハイパーパラメータ（ここでは成分数 8）を使ってトレーニングセットで回帰モデルを構築し，得られた結果を変数 model に代入した．

5　回帰結果の評価

得られた回帰モデルにスペクトルデータを入力してセタン価を予測してみよう（図 7）．ここでは 40 行のように，153 本あるテストセットの 1 本目のスペクトルを取り出し，spec とした．41 行では model.predict メソッドにこのスペクトルデータを入力して結果を出力した．ただしこの方法では，Python で得られた回帰モデルを測定装置に組み込む際に，Python で実装しなければならなくなる．

そこで次に，得られた回帰モデルの実体である回帰係数を用いて同様の予測を行ってみよう．model の回帰係数は model.coef_ で取り出すことができる．ただし，回帰係数 model.coef_ にスペクトルデータ spec を直接当てはめても正しい予測値は得られないことに注意が必要である．これは目的変数もオートスケーリングされているためであり，正しく予測するには 42 行のようにしてオートスケーリングを元に戻さなければならない．

```
039 # 回帰係数を用いた予測
040 spec = test.iloc[0].values
041 print(model.predict([spec])[0][0])
042 print(((spec - model._x_mean) / model._x_std @ model.coef_[0] +
    model._y_mean)[0])
```

```
53.10374305917401
53.10374305917401
```

図 7　回帰係数を用いた予測

続いて回帰の結果をプロットしてみよう（図 8）．44 行と 45 行でそれぞれ，得られた回帰モデルにトレーニングセットとテストセットを当てはめ，キャリブレーションとバリデーションの結果を計算した．46 行で作図する領域を準備し，47 行で（横軸）=（縦軸）となる対角線をプロットしている．ここで横軸はセタン価の真値，縦軸は PLS 回帰によるセタン価の予測値である．48 行によってキャリブレーションの結果が青色に，49 行によってバリデーションの結果がオレンジ色に，それぞれプロットされた．

この回帰の結果を評価してみよう．ここでは二乗平均平方根誤差（root-mean-square error, RMSE）と決定係数 R^2 を計算してみる．二乗平均誤差（mean-square-error，MSE）は sklearn.metrics.mean_squared_error 関数で計算できるので，RMSE はこれの平方根を計算すればよい．決定係数は sklearn.metrics.r2_score 関数で計算できる．図 9 にキャリブレーションとバリデーションにおけるそれぞれの RMSE と決定係数の計算結果を示した．

```
043 # 回帰結果のプロット
044 calibration = model.predict(train.values)
045 validation = model.predict(test.values)
046 pyplot.figure(figsize=(4, 4))
047 pyplot.plot([data.index.min(), data.index.max()],
    [data.index.min(), data.index.max()])
048 pyplot.scatter(train.index, calibration)
049 pyplot.scatter(test.index, validation)
050 pyplot.show()
```

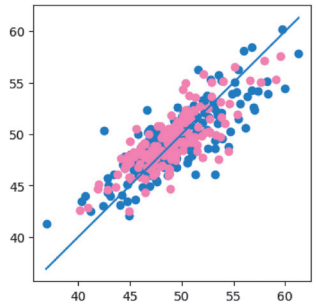

図 8　回帰結果のプロット

横軸はセタン価の真値，縦軸は PLS 回帰によるセタン価の予測値．青色はキャリブレーションの結果，オレンジ色はバリデーションの結果．

```
051 # RMSE と決定係数の計算
052 from sklearn.metrics import mean_squared_error, r2_score
053 print("calibration")
054 print("RMSE =", numpy.sqrt(mean_squared_error(train.index,
    calibration)))
055 print("R^2 =", r2_score(train.index, calibration))
056 print("")
057 print("validation")
058 print("RMSE =", numpy.sqrt(mean_squared_error(test.index,
    validation)))
059 print("R^2 =", r2_score(test.index, validation))
```

```
calibration
RMSE = 2.1063809198962775
R^2 = 0.6649517526812241

validation
RMSE = 2.0962595727739464
R^2 = 0.6012617326815684
```

図 9　RMSE と決定係数の計算

今回は図 3 で 960−1120 nm の領域を選んだが，1200−1400 nm の領域など，他の波長領域を選んでみる，図 4 の二次微分で窓の大きさを 9 としたが他の値にしてみる，といった工夫で，よりよい回帰モデルが得られるかもしれないので挑戦してみてほしい．二次微分における窓の大きさをハイパーパラメータとして PLS 回帰モデルをチューニングするには，パイプラインと呼ばれる計算テクニックが必要となるが，紙面の都合により説明は割愛する．実装するには sklearn.pipeline.make_pipeline 関数を用いればよい[2]．二次微分における窓の大きさと次元削減における成分数のように，二つのハイパーパラメータを同時に最適化するには，二次元のグリッドサーチを行えばよく，さらに高次のグリッドサーチも GridSearchCV クラスで実装が可能である．

6 ま と め

本稿では Python を用いて PLS 回帰の計算を行う方法を紹介した．説明は必要最低限にとどめており，まずはここまでの技術をしっかりと修得してほしい．データの前処理とモデルのチューニングは腕の見せ所であり，工夫することでよりロバストな学習器を得ることができる．

学習器に入力する機器分析データや教師データは人が測定することになるが，それらの誤差よりも計算による推定値の誤差が小さくなることは原理的にあり得ない．機器分析のプロは，サンプルの前処理，機器分析の最適化，計算データの前処理，計算の最適化の四つのプロセスに精通している必要があり，分析技術だけでなく，計算技術も鍛錬してほしい．

文　献

1）金子弘昌：“化学のための Python によるデータ解析・機械学習入門”，(2019)，(オーム社)．
2）森田成昭：“Python で始める機器分析データの解析とケモメトリックス”，(2022)，(オーム社)．
3）S. Wold, M. Sjöström, L. Eriksson : *Chemometrics Intellig. Lab. Syst.*, **58**, 109（2001）．
4）永田　靖，棟近雅彦：“多変量解析法入門”，(2001)，(サイエンス社)．
5）A. C. Müller, S. Guido, 中田　秀：“Python ではじめる機械学習：scikit-learn で学ぶ特徴量エンジニアリングと機械学習の基礎”，(2017)，(オライリー・ジャパン)．
6）宮下芳勝，佐々木慎一：“ケモメトリックス 化学パターン認識と多変量解析”，(1995)，(共立出版)．
7）尾崎幸洋，宇田明史，赤井俊雄：“化学者のための多変量解析 ケモメトリックス入門”，(2002)，(講談社サイエンティフィク)．
8）長谷川健：“スペクトル定量分析”，(2005)，(講談社サイエンティフィク)．
9）森田成昭：日本結晶成長学会誌，**49**, 49（2022）．
10）森田成昭：ぶんせき（*Bunseki*），**2020**, 290.
11）〈www.anaconda.com〉(2023 年 11 月 27 日確認)．
12）池内孝啓，片柳薫子，岩尾エマはるか，@driller : “Python ユーザのための Jupyter［実践］入門”，(2017)，(技術評論社)．

森田 成昭（Morita Shigeaki）
大阪電気通信大学工学部（〒572−8530 大阪府寝屋川市初町 18−8）．東京農工大学大学院生物システム応用科学研究科博士後期課程修了．博士（学術）．《現在の研究テーマ》分子分光とデータ解析．《主な著書》“Python で始める機器分析データの解析とケモメトリックス”，(オーム社)．《趣味》ジャズドラム．
E-mail : smorita@isc.osakac.ac.jp