初心者も挑戦！
機械学習をプログラムで実装してみよう

物理学科4年　　礒川 雄介

今回は、機械学習のもっとも基本的な解析方法である「重回帰分析」をpythonを使って実装してみます。

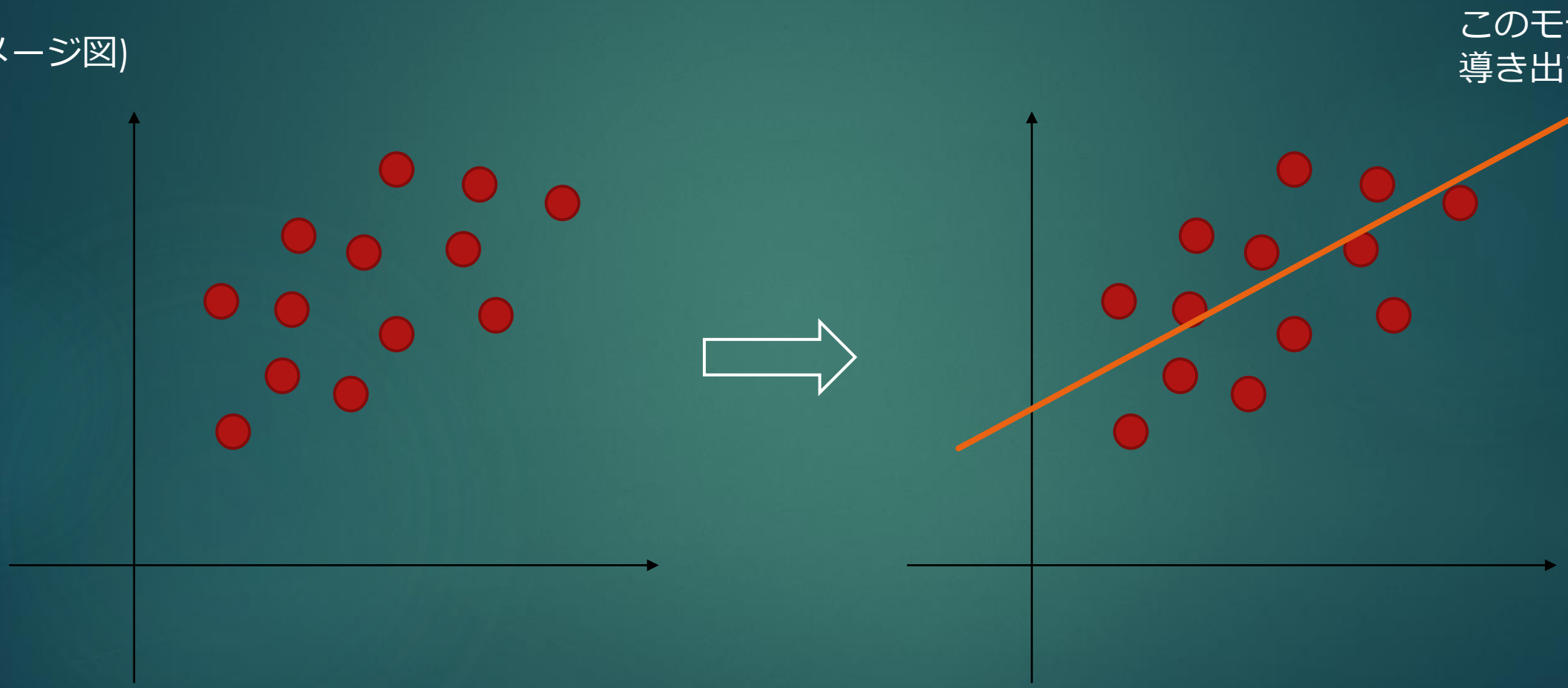
もし興味があれば、パソコンを使って手を動かしてみるとより理解が深まると思います。

Googleアカウントを持っていれば環境構築を必要とせずにコードが書ける「Google Colaboratory」を利用するのがおすすめです。

https://colab.research.google.com/notebooks/welcome.ipynb?hl=ja

回帰とは、教師あり学習といわれる機械学習の一種で、簡単に言うと、いくつかのデータからデータの特徴を表すモデルを導き出し、値の予測をできるようにするもの。

(イメージ図)



今回はその中でも、重回帰分析の実装を行っていきます。

機械学習を実装する上で必要となるライブラリが豊富にある「scikit-learn」というパッケージを利用していますので確認してみてください。

https://scikit-learn.org/stable/index.html

Machine Learning.ipynb

ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました

コメント 共有

目次

- 重回帰分析の実装
- 過学習の抑制
- PLSの実装
- セクション

+ コード + テキスト

RAM ディスク 編集

重回帰分析の実装

必要なライブラリをインポートする。 numpy…科学技術計算をする際の基本的なツール。高レベルの数学関数が用意されている。 pandas…データを変換したり解析したりするためのライブラリ。 matplotlib…科学技術計算向けのグラフ描画ライブラリ。 seaborn…データ可視化ライブラリ。

```
[1] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
```

scikit-learnという機械学習に必要なライブラリから、ボストンの住宅価格のサンプルデータを使用する。

```
[2] from sklearn.datasets import load_boston
```

サンプルデータをdatasetという変数に格納

```
[3] dataset = load_boston()
```

datasetから入力変数をx、目標変数をtとして値を取る。

```
[4] x,t = dataset.data, dataset.target
    columns = dataset.feature_names
```

xの中身と形を確認してみる。

0 秒 完了時間: 8:30

[4]
```
x,t = dataset.data, dataset.target
columns = dataset.feature_names
```

xの中身と形を確認してみる。

[5]
```
type(x), x.shape
```

```
(numpy.ndarray, (506, 13))
```

tの中身と形を確認してみる

[6]
```
type(t), t.shape
```

```
(numpy.ndarray, (506,))
```

パラメータ名をcolumnsで確認してみる。

[7]
```
columns
```

```
array(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD',
       'TAX', 'PTRATIO', 'B', 'LSTAT'], dtype='<U7')
```

pandasで扱える形に変換する。head()メソッドでDataFrameの先頭5行目までを見てみる。

[8]
```
df = pd.DataFrame(x, columns=columns)
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14

pandasで扱える形に変換する。head()メソッドでDataFrameの先頭5行目までを見てみる。

```
[8] df = pd.DataFrame(x, columns=columns)
    df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

Targetというcolumn名を追加する。

```
[9] df['Target'] = t
    df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	Target
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

numpy.ndarrayに変換する。効率性などからndarrayという独自のデータ構造を使う。valuesメソッドを使うことで変換できる。

numpy.ndarrayに変換する。効率性などからndarrayという独自のデータ構造を使う。valuesメソッドを使うことで変換できる。

```
[10] t = df['Target'].values
```

0秒

dropメソッドを使って、Targetだけを抜き取る。

```
[11] x = df.drop(labels=['Target'], axis=1).values
```

0秒

scikit-learnのtrain_test_split()関数を使うと、NumPy配列ndarrayやリストなどを二分割できる。機械学習においてデータを訓練用（学習用）とテスト用に分割してホールドアウト検証を行う際に用いる。

```
[12] from sklearn.model_selection import train_test_split
```

0秒

入力変数と目標変数をtest_size=0.3とすることによって7:3で分ける。random_state=0とすることによって、実行のたびに結果が変わってしまうことを防いでいる。x_train,t_trainがx,tの学習データ、x_test,t_testがx,tのテストデータ。

```
[13] x_train, x_test, t_train, t_test = train_test_split(x, t, test_size=0.3, random_state=0)
```

0秒

lenで確認すると、一番最初に取得したデータ数506の3割分のデータ数152が取得できたとわかる。

```
[14] len(x_test)

     152
```

0秒

重回帰分析のアルゴリズムを宣言するために LinearRegressionをインポートする。

```
[15] from sklearn.linear_model import LinearRegression
```

0秒

重回帰分析のアルゴリズムを宣言するために LinearRegressionをインポートする。

```
[15] from sklearn.linear_model import LinearRegression
```

0秒

modelというインスタンスを作成。インスタンスとはクラスから作成される個々のモノのこと。それを重回帰分析するアルゴリズムを宣言。

```
[16] model = LinearRegression()
```

0秒

インスタンスの中身を確認してみる。

```
[17] model
```

0秒

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

fitメソッドでモデルの学習をする。

```
[18] model.fit(x_train, t_train)
```

0秒

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

これで学習が完了。学習後のパラメータを確認する。

```
[19] model.coef_
```

0秒

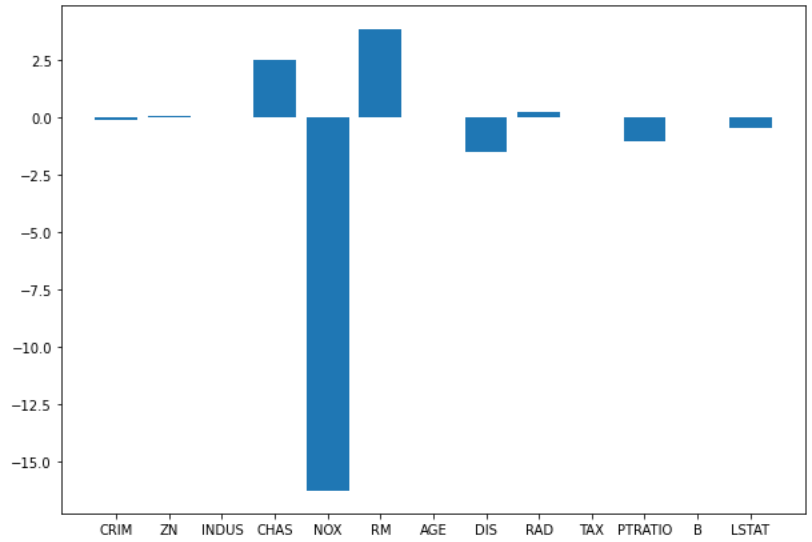
```
array([-1.21310401e-01,  4.44664254e-02,  1.13416945e-02,  2.51124642e+00,
       -1.62312529e+01,  3.85906801e+00, -9.98516565e-03, -1.50026956e+00,
        2.42143466e-01, -1.10716124e-02, -1.01775264e+00,  6.81446545e-03,
       -4.86738066e-01])
```

13個のパラメータがあるとわかった。パラメータの大きさをイメージしやすいようにmatplotlibでパラメータをヒストグラムで可視化してみ

13個のパラメータがあるとわかった。パラメータの大きさをイメージしやすいようにmatplotlibでパラメータをヒストグラムで可視化してみる。

```
[20] plt.figure(figsize=(10,7))
     plt.bar(x=columns, height=model.coef_)
```

```
<BarContainer object of 13 artists>
```



バイアスの大きさを確認。

```
[21] model.intercept_
```

```
37.93710774183309
```

バイアスの大きさを確認。

```
[21] model.intercept_
```

```
37.93710774183309
```

学習の精度を検証。 決定係数という指標をscoreというメソッドで出力できる。返ってきた値が1に近ければ近いほど良い学習モデルであると言える。

```
[22] print(f'train score: {model.score(x_train, t_train)}')
     print(f'test_score: {model.score(x_test, t_test)}')
```

```
train score: 0.7645451026942549
test_score: 0.6733825506400171
```

新たなデータを加えて推論を行っていく。テストデータを追加して、predictメソッドで学習したモデルから値を予測。

```
[23] y = model.predict(x_test)
```

予測値と目標値を比較。大幅にずれてしまっている！ これは「過学習」が起こってしまっている可能性が高い。過学習とは、手元のデータにフィットしすぎて、予測したいデータに全く当てはまらなくなってしまうこと。

```
[24] print(f'予測値: {y[1]}')
     print(f'目標値: {t_test[1]}')
```

```
予測値: 23.751631640748066
目標値: 50.0
```

この過学習を抑制しなければならない。

```
[24]
```

ここからは過学習を抑制するための手法を紹介します。以下のURLからサンプルデータをダウンロードしてみてください。

https://firestorage.jp/download/23ee31b482dc937cfc054051273e0c0f53942b00

予測値: 23.751631640748066
目標値: 50.0

この過学習を抑制しなければならない。

過学習の抑制

ここからは過学習を抑制する方法を紹介する。事前に用意したサンプルデータをアップロードして読み込む。

```
[ ] df = pd.read_csv('regression_pls.csv')
    df.head()
```

	Target	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	x19	x20	x
0	1.58	59.068	54.028	59.037114	24	0	0.213790	-0.369921	0.369921	0.213790	9.222222	-0.333333	9.222222	0.333333	2.803039	29.019550	3.577350	2.485599	2.485599	1.732051	0.992799	0.9927
1	1.34	46.073	40.025	46.053098	20	0	-0.001725	-0.271722	0.271722	0.001725	4.597222	1.652778	4.597222	1.652778	1.632993	2.754888	2.707107	2.077350	2.077350	1.414214	0.788675	0.7886
2	1.22	60.052	56.020	60.021129	24	0	0.299685	-0.481433	0.481433	0.299685	9.000000	-0.833333	9.000000	0.833333	2.803039	27.019550	3.577350	2.355462	2.355462	1.732051	0.927731	0.9277
3	1.15	71.123	62.051	71.073499	30	0	-0.004845	-0.316731	0.316731	0.004845	3.222222	1.250000	3.222222	1.250000	2.083333	15.219281	3.535534	3.328427	3.328427	2.500000	2.207107	2.2071
4	1.12	76.055	72.023	76.027277	30	0	0.335391	-0.349891	0.349891	0.335391	9.229167	-0.939815	9.229167	0.939815	2.847379	42.912609	4.284457	2.432812	2.432812	2.270056	0.966406	0.9664

5 rows × 197 columns

```
[ ] df.shape
```

(1290, 197)

```
[ ] t = df['Target'].values
    x = df.drop('Target', axis=1).values
```

(1290, 197)

```
[ ] t = df['Target'].values
    x = df.drop('Target', axis=1).values
```

```
[ ] t.shape
```

(1290,)

```
[ ] x.shape
```

(1290, 196)

ここからは先ほどと同じ手順。

```
[ ] x_train, x_test, t_train, t_test = train_test_split(x, t, test_size=0.3, random_state=0)
```

```
[ ] model = LinearRegression()
```

```
[ ] model.fit(x_train, t_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

先ほどと同じように精度を検証。

```
[ ] print(f'train score: {model.score(x_train, t_train)}')
    print(f'test_score: {model.score(x_test, t_test)}')
```

```
train score: 0.9365472506823411
test_score: 0.003576724162620226
```

先ほどと同じように精度を検証。

```
print(f'train score: {model.score(x_train, t_train)}')
print(f'test_score: {model.score(x_test, t_test)}')
```

```
train score: 0.9365472506823411
test_score: 0.003576724162620226
```

test_scoreの値がとても低い。つまり精度がかなり低い。ここでも「過学習」が起こってしまっているとわかる。過学習の原因として、多重共線性という相関の強いパラメータが入っていることによってモデルの予測精度に影響を及ぼす現象が起こってしまっていることが考えられる。なので、相関関係を確認してみる。corrメソッドでパラメータの相関係数を見ることができる。

```
df.corr()
```

	Target	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15	x16	x17	x18	
Target	1.000000	-0.642326	-0.648078	-0.640489	-0.524453	NaN	0.111829	-0.360696	0.357026	0.113189	0.120581	0.000857	0.120581	-0.041054	0.242124	-0.580994	-0.542549	-0.507630	-0.642483	-0.
x1	-0.642326	1.000000	0.997571	0.999978	0.908895	NaN	0.322508	-0.117193	0.134074	0.298204	0.378471	-0.446958	0.378471	-0.325579	-0.379325	0.826032	0.917214	0.839511	0.958681	0.
x2	-0.648078	0.997571	1.000000	0.997252	0.883891	NaN	0.322631	-0.097297	0.115794	0.294947	0.363786	-0.450551	0.363786	-0.319195	-0.357903	0.824275	0.896737	0.804689	0.939308	0.
x3	-0.640489	0.999978	0.997252	1.000000	0.910855	NaN	0.324352	-0.120477	0.137237	0.300415	0.380881	-0.448158	0.380881	-0.326151	-0.380865	0.827493	0.919039	0.842162	0.959551	0.
x4	-0.524453	0.908895	0.883891	0.910855	1.000000	NaN	0.385792	-0.284647	0.293981	0.382603	0.506428	-0.469375	0.506428	-0.377003	-0.457913	0.857233	0.995378	0.979624	0.967419	0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
x192	0.007788	0.001304	0.005912	0.001447	-0.010834	NaN	0.024306	-0.015943	0.014061	0.025697	-0.019591	-0.014460	-0.019591	0.018263	-0.009835	0.043685	-0.009241	-0.019158	-0.007010	0.
x193	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
x194	0.002448	-0.015226	-0.015106	-0.015093	-0.016378	NaN	-0.041735	0.036258	-0.038957	-0.042337	-0.071180	0.060033	-0.071180	0.094576	-0.008548	-0.005929	-0.019135	-0.011117	-0.007802	-0.
x195	-0.113820	-0.038657	-0.062823	-0.038138	0.027813	NaN	0.009401	-0.077525	0.070936	0.019969	0.017027	0.055359	0.017027	0.021115	0.022769	-0.176103	-0.009511	0.057212	0.024318	-0.
x196	0.043600	0.027857	0.027773	0.028359	0.055553	NaN	0.242888	0.000282	0.023008	0.227033	0.174731	-0.160939	0.174731	-0.052840	-0.019068	0.011104	0.060826	0.040373	0.018554	0.

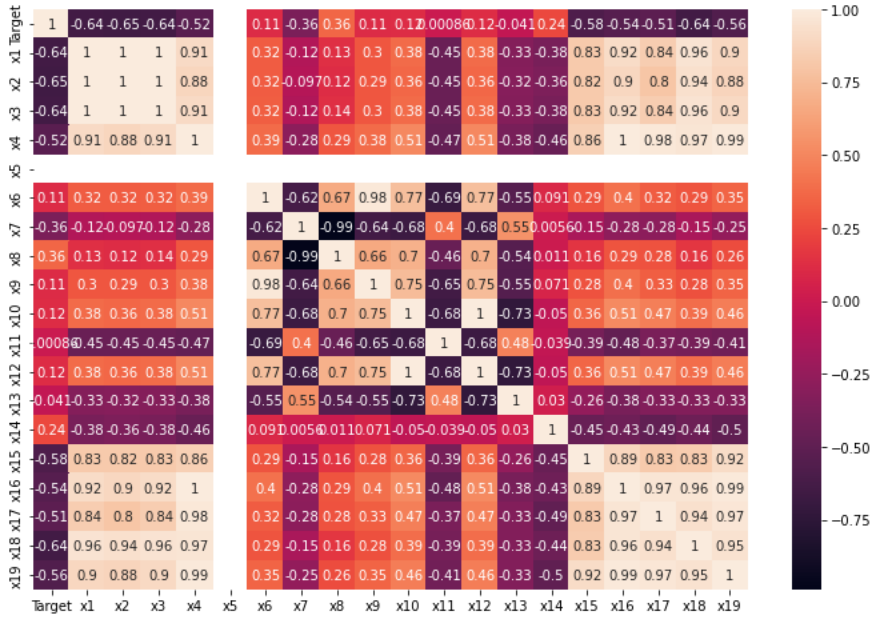
197 rows × 197 columns

数値だと少し見づらいのでヒートマップを作成する。相関係数が高いとベージュ、低いと赤黒い色に近づき、直感的にわかりやすい。パラメータが非常に多いので、20行20列に凝縮して作成。

```
df_corr = df.corr()
```

```
plt.figure(figsize=(12,8))
sns.heatmap(df_corr.iloc[:20, :20], annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f39bb194e10>
```



ラベルを指定して、実際に相関係数が高いところの値を実際に確認してみる。

Machine Learning.ipynb

ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました

コメント 共有

目次

- 重回帰分析の実装
- 過学習の抑制
- PLSの実装
- セクション

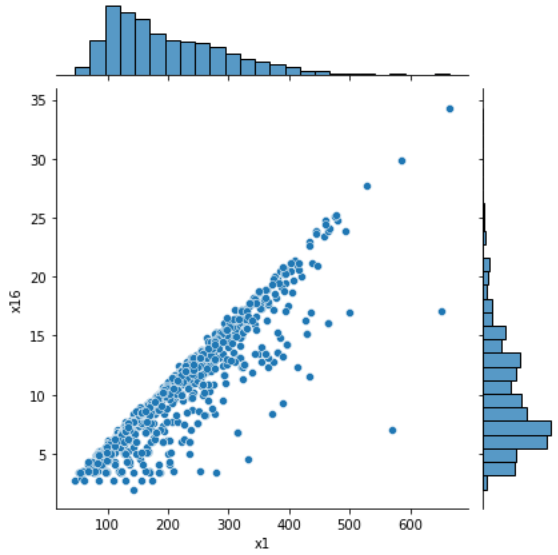
+ コード + テキスト

RAM ディスク 編集

ラベルを指定して、実際に相関係数が高いところの値を実際に確認してみる。

```
[ ] sns.jointplot(x='x1', y='x16', data=df)
```

```
<seaborn.axisgrid.JointGrid at 0x7f39baa03590>
```



非常に強い相関関係が見て取れる。 ここで別のアルゴリズムで 解析 をしてみる。

PLSの実装

[ ] ↳ 8 個のセルが非表示

0 秒 完了時間: 8:30

ここから「PLS」というアルゴリズムを実装してみます。
PLS(Partial least Squares)とは、データを直に使わずにスコアを計算しPLSを利用することにより、多重共線性の問題を回避し、線型回帰モデルを構築することができます。

PLSの実装

PLSのアルゴリズムを宣言するためPLSRegressionをインポート。

```
[ ] from sklearn.cross_decomposition import PLSRegression
```

n_componentsを自分で設定して、パラメータを凝縮する。このように人間が設定するパラメータをハイパーパラメータという。

```
[ ] pls = PLSRegression(n_components=11)
```

先ほどと同じようにモデルを学習。

```
[ ] pls.fit(x_train, t_train)
```

```
PLSRegression(copy=True, max_iter=500, n_components=11, scale=True, tol=1e-06)
```

精度を検証すると、かなり良い値が取れた！test_scoreはn_componentを変更することで値を調整することができる。

```
[ ] print(f'train score: {pls.score(x_train, t_train)}')
    print(f'test_score: {pls.score(x_test, t_test)}')
```

```
train score: 0.9157705245807128
test_score: 0.8839475118318957
```

(参考文献)

1.アンドレアス・C・ミュラー、サラ・グイド,Pythonではじめる機械学習,オライリー・ジャパン,2017-05-22

2.Data Viz Lab,過学習とは？初心者向けに理由から解決法までわかりやすく解説, 2021-7-14,閲覧日 2021-10-19,https://data-viz-lab.com/overfitting

3. Animesh Agarwal, Towards Data Science, Linear Regression on Boston Housing Dataset,2018-10-5,閲覧日 2021-10-19,https://towardsdatascience.com/linear-regression-on-boston-housing-dataset-f409b7e4a155

4.β short Lab,部分的最小２乗法（PLS）のメモ,閲覧日 2021-10-19,https://betashort-lab.com/%E3%83%87%E3%83%BC%E3%82%BF%E3%82%B5%E3%82%A4%E3%82%A8%E3%83%B3%E3%82%B9/%E7%B5%B1%E8%A8%88%E5%AD%A6/%E9%83%A8%E5%88%86%E7%9A%84%E6%9C%80%E5%B0%8F%EF%BC%92%E4%B9%97%E6%B3%95%EF%BC%88pls%EF%BC%89%E3%81%AE%E3%83%A1%E3%83%A2/