

機械学習レポート

機械学習とモデリングプロセス

➤ 機械学習

- コンピュータプログラムは、タスクTを性能指標Pで測定。その性能が経験Eにより改善される場合、タスクTおよび性能指標Pに関して経験Eから学習すると言われている（トム・ミッチェル 1997）
- 人がプログラムするのは認識の仕方ではなく学習の仕方

➤ モデリングプロセスで最も重要なことは問題の設定。解き方(手段)として必ずしも機械学習が最適ではないことも考慮する。

モデリングプロセス

プロセス		概要
1	問題設定	どのような課題を機械学習に解決させるか
2	データ設定	どのようなデータを扱うか
3	データの前処理	モデルに学習させられるようにデータを変換
4	機械学習モデルの選定	どの機械学習モデルを利用するか
5	モデルの学習	パラメータの決め方
6	モデルの評価	ハイパーパラメータの選定モデル精度を測る

重要

線形回帰モデル 要点

- 線形回帰モデルは単回帰モデルと重回帰モデルがある。
 - 単回帰モデル ($m=1$ の場合)
モデル式 : $y = \omega_0 + \omega_1 x_1 + \varepsilon$
※ y : 目的関数、 ω_0 : 切片、 ω_1 : 回帰係数、 x_1 : 説明変数、 ε : 誤差
 - 重回帰モデル ($m > 1$ の場合)
モデル式 : $y = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \varepsilon$
※ y : 目的関数、 ω_0 : 切片、 ω_1, ω_2 : 回帰係数、 x_1, x_2 : 説明変数、 ε : 誤差
- 線形回帰モデルのパラメータは最小二乗法で推定する

線形回帰モデル 実装演習結果キャプチャ

[illegible]

線形回帰モデル 実装演習結果キャプチャ

The screenshot displays a Google Colab environment with a notebook named "skl_regression.ipynb". The interface includes a top navigation bar with various icons and a sidebar on the left showing file explorer and search options. The main workspace contains two code cells. The first cell, labeled "#DESCR変数の中身を確認", imports the boston dataset from sklearn.datasets and prints its description. The output shows detailed information about the Boston house prices dataset, including the number of instances (506), attributes (13), and their descriptions. The second cell, labeled "#feature_names変数の中身を確認", prints the feature names of the dataset. The output lists the feature names: ["CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B", "LSTAT"].

線形回帰モデル 実装演習結果キャプチャ

受電トレイ (6.256) - spve × 学習要領-各種画面へ × レポート作成方法 × 推薦図書と試験対策 × オリエンテーション (PDF) × 機械学習 講義資料 × skl_ml - Googleドライブ × skl_regression.ipynb - × 【文字数カウント】 × + 受電 ×

← → ↺ 🏠 🔒 https://colab.research.google.com/drive/1d4c4hNnMEFIO-i1blfy1Y6CUFxb1Gt#scrollTo=8WwcaVdkMsNA 80% ☆ 📄 ⬇️ 📏 📄 🟢 ☰

skl_regression.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

+ コード + テキスト

RAM
ディスク

編集

```
[6] #data変数(説明変数)の中身を確認
print(boston['data'])

[[6.3200e+03 1.8000e+01 2.3100e+00 ... 1.5300e+01 3.9690e+02 4.9800e+00]
 [2.7310e+02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9690e+02 9.1400e+00]
 [2.7290e+02 0.0000e+00 7.0700e+00 ... 1.7800e+01 3.9283e+02 4.0300e+00]
 ...
 [6.0760e+02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 5.6400e+00]
 [1.0590e+01 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9345e+02 6.4800e+00]
 [4.7410e+02 0.0000e+00 1.1930e+01 ... 2.1000e+01 3.9690e+02 7.8800e+00]]

[7] #target変数(目的変数)の中身を確認
print(boston['target'])

[[24. 21.6 34.7 39.4 36.2 29.7 22.9 27.1 16.5 18.9 15. 18.9 21.7 20.4
 18.2 19.9 23.1 17.5 20.2 18.2 13.6 19.6 15.2 14.5 15.6 13.9 16.6 14.8
 18.4 21. 12.7 14.5 13.2 13.1 13.5 18.9 20. 21. 24.7 30.8 34.9 28.6
 25.9 24.7 21.2 19.3 20. 16.6 14.4 19.4 19.7 20.5 25. 23.4 18.9 35.4
 24.7 21.6 23.9 19.6 18.7 16. 22.2 25. 39. 23.5 19.4 22. 17.4 20.9
 24.2 21.7 22.9 23.4 24.1 21.4 20. 20.9 21.4 20.9 23.9 24.9 22.9
 23.9 26.6 22.5 22.2 23.6 28.7 22.6 22. 22.9 25. 20.6 28.4 21.4 38.7
 49.8 33.2 27.5 26.5 18.6 19.3 20.1 19.5 19.5 20.4 19.8 19.4 21.7 22.8
 18.8 18.7 18.5 18.3 21.2 19.2 20.4 19.9 22. 20.3 20.5 17.3 18.8 21.4
 15.7 16.2 18. 14.3 19.2 19.6 23. 18.4 15.6 18.1 17.4 17.1 13.3 17.8
 14. 14.4 12.4 15.6 11.9 10.9 15.6 14.6 17.9 15.4 21.5 19.6 15.3 19.4
 17. 15.6 13.1 41.3 24.9 23.3 27. 50. 50. 50. 22.7 25. 50. 23.8
 23.8 22.9 17.4 19.1 23.1 23.6 22.6 29.4 29.2 24.6 29.9 37.2 39.8 36.2
 37.9 32.5 26.4 29.6 50. 32. 29.8 34.9 37. 30.5 36.4 31.1 29.1 50.
 33.3 30.9 34.6 34.9 32.9 34.1 42.3 48.9 50. 22.6 24.4 22.5 24.4 20.
 21.7 19.9 22.4 28.1 23.7 25. 23.9 28.7 21.5 23. 28.7 21.7 27.5 30.1
 44.8 50. 37.6 31.6 48.7 31.5 24.3 31.7 41.7 48.3 29. 24. 25.1 31.5
 23.7 23.9 22. 20.1 22.2 23.7 17.6 18.5 24.3 20.5 24.5 26.2 24.4 24.8
 29.6 42.8 21.9 20.9 44. 50. 36. 30.1 39.8 42.1 48.8 31. 36.5 22.8
 39.7 50. 42.5 20.7 21.1 25.2 24.4 35.2 32.4 32. 33.2 39.1 29.1 35.1
 45.4 35.4 46. 50. 32.2 22. 20.1 23.2 22.3 24.8 28.5 37.3 27.9 23.9
 21.7 28.6 27.1 20.3 22.5 29. 24.8 22. 26.4 33.1 36.1 28.4 33.4 28.2
 22.8 20.9 16.1 22.1 19.4 21.6 23.8 16.2 17.8 19.8 23.1 21. 23.8 23.1
 20.4 18.5 25. 24.6 23. 22.2 19.9 22.6 19.8 17.1 19.4 22.2 20.7 21.1
 19.5 18.5 20.6 19. 18.7 32.7 16.5 23.9 31.2 17.5 17.2 23.1 24.5 26.6
 22.9 24.1 18.6 30.1 18.2 20.6 17.8 21.7 22.7 22.6 25. 19.9 20.8 16.8
 21.9 27.5 21.9 23.1 50. 50. 50. 50. 13.8 13.8 15. 13.9 13.3
 13.1 10.2 10.4 10.9 11.9 12.3 8.8 7.2 10.5 7.4 10.2 11.5 15.1 23.2
 9.7 13.8 12.7 13.1 12.5 8.5 5. 6.3 5.6 7.2 12.1 8.3 9.5 5.
 11.9 27.9 17.2 27.5 15. 17.2 17.9 16.3 7. 7.2 7.5 10.4 8.8 8.4
 16.7 14.2 20.8 13.4 11.7 8.3 10.2 10.9 11. 9.5 14.5 14.1 16.1 14.3
 11.7 13.4 9.6 8.7 8.4 12.8 10.5 17.1 18.4 15.4 10.8 11.8 14.9 12.6
 14.1 13. 13.4 15.2 16.1 17.8 14.9 14.1 12.7 13.5 14.9 20. 16.4 17.7
 19.5 20.2 21.4 19.9 13. 19.1 19.1 20.9 19.6 23.2 23.8 13.8 13.9
 16.7 12. 14.6 21.4 23. 23.7 25. 21.8 20.6 21.2 19.1 20.6 15.2 7.
 8.1 13.6 20.1 21.8 24.5 23.1 19.7 18.9 21.2 17.5 16.8 22.4 20.6 23.9
 22. 11.9]]
```

2. データフレームの作成

```
[8] # 説明変数らをDataFrameへ変換
df = DataFrame(data=boston.data, columns = boston.feature_names)

[9] # 目的変数をDataFrameへ追加
df['target'] = boston.target
```

0秒 完了時間: 1846

線形回帰モデル 実装演習結果キャプチャ

受信トレイ (6/26) × 学習履歴・各種画面への × レポート作成方法 × 推薦図書と試験対策 × オリエンテーション (PDF) × 機械学習 講義資料 × skl_ml - Google ドライブ × skl_regression.ipynb - C × 【文字数カウント】 × + - 閉

← → ↺ 閉 https://colab.research.google.com/drive/1d4c4hNnMEFIO-i1blifj1Y6CUFxb1Gt#scrollTo=8WwcaVdkMsNA 80% ☆ 閉 下 印刷 設定 三

skl_regression.ipynb ☆ ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変数を保存しました

+ コード + テキスト

RAM ディスク 編集

2. データフレームの作成

```
[8] # 説明変数らをDataFrameへ実装
df = DataFrame(data=boston.data, columns = boston.feature_names)

[9] # 目的変数をDataFrameへ追加
df['PRICE'] = np.array(boston.target)

[10] # 最初の5行を表示
df.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
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

線形単回帰分析

```
[11] #カラムを指定してデータを表示
df[['RM']].head()
```

	RM
0	6.575
1	6.421
2	7.185
3	6.998
4	7.147

```
[12] # 説明変数
data = df.loc[:, ['RM']].values

[13] #dataリストの表示(1-5)
data[0:5]
```

```
array([[6.575],
       [6.421],
       [7.185],
       [6.998],
       [7.147]])

[14] # 目的変数
```

0 秒 完了時間: 18:46

線形回帰モデル 実装演習結果キャプチャ

(6.256) - spvix

学習要領・各種画面への

レポート作成方法

推薦図書と試験対策

オリエンテーション (PDF)

機械学習 講義資料

skl_ml - Google ドライブ

skl_regression.ipynb - C X

【文字数カウント】

<

→

🏠

🔒

https://colab.research.google.com/drive/1d4c4hNnMEFIO-i1blifj1Y6CUfxb1Gt#scrollTo=8WwcaVdkMsNA

80%

☆

🔄

↓

📄

🖨️

⌵

☰

🔗 skl_regression.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

コメント共有設定K

+ コード + テキスト

RAM ディスク

✓ 編集 ↕

線形単回帰分析

[11] # ラムを指定してデータを表示
df[['RM']].head()

RM
0 6.575
1 6.421
2 7.185
3 6.998
4 7.147

[12] # 説明変数
data = df.loc[:, ['RM']].values

[13] # dataリストの表示(1-5)
data[0:5]

array([[6.575],
 [6.421],
 [7.185],
 [6.998],
 [7.147]])

[14] # 目的変数
target = df.loc[:, 'PRICE'].values

[15] target[0:5]

array([24. , 21.6, 34.7, 33.4, 36.2])

[16] ## sklearnモジュールからLinearRegressionをインポート
from sklearn.linear_model import LinearRegression

[17] # オブジェクト生成
model = LinearRegression()
Model.get_params()
Model = LinearRegression(fit_intercept=True, normalize=False, copy_X=True, n_jobs=1)

[18] # fit関数でパラメータ推定
model.fit(data, target)

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

[19] #予測
model.predict([[1]])

✓ 0秒完了時間: 18:46

線形回帰モデル 実装演習結果キャプチャ

受信トレイ (6,256) × 学習要領・各種画面への × レポート作成方法 × 推薦図書と試験対策 × オリエンテーション (PDF) × 機械学習 講義資料 × ski_ml - Google ドライブ × ski_regression.ipynb - C × 【文字数カウント】 × + - 閉

← → ↺ https://colab.research.google.com/drive/1d4c4hNnMEfIO-i1blifyJ1Y6CUFxb1Gt#scrollTo=8WwcaVdkMsNA 80% ☆

ski_regression.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

+ コード + テキスト

コメント 共有 K

RAM ディスク 編集

重回帰分析(2変数)

```
[20] #カラムを指定してデータを表示
df[['CRIM', 'RM']].head()

      CRIM      RM
0  0.00632  6.575
1  0.02731  6.421
2  0.02729  7.185
3  0.03237  6.998
4  0.06905  7.147

[21] #説明変数
data2 = df.loc[:, ['CRIM', 'RM']].values
#目的変数
target2 = df.loc[:, 'PRICE'].values

[22] #オブジェクト生成
model2 = LinearRegression()

[23] #fit関数でパラメータ推定
model2.fit(data2, target2)

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

model2.predict([[0.2, 7]])
array([28.43977562])

# 重回帰の回帰係数と切片を確認

[25] # 重回帰の回帰係数と切片を出力
print('推定された回帰係数: %.3f, 推定された切片: %.3f' % (model.coef_, model.intercept_))

推定された回帰係数: 9.102, 推定された切片: -34.671

[26] # 重回帰の回帰係数と切片を出力
print(model.coef_)
print(model.intercept_)

[9.10210898]
-34.67062077645857
```

警告: GPU ランタイムに接続していますが、GPU は使用されていません。 [標準ランタイムに切り替える](#)

0 秒 完了時間: 18:46

線形回帰モデル 実装演習結果キャプチャ

受信トレイ (6,256) × 学習要領・各種画面への × レポート作成方法 × 推薦図書と試験対策 × オリエンテーション (PDF) × 機械学習 講義資料 × skl_ml - Google ドライブ × skl_regression.ipynb - C × 【文字数カウント】 × + - 閉

← → ↺ 家 <https://colab.research.google.com/drive/1d4c4hNnMEfIO-t1blifJ1Y6Cufxb1Gt#scrollTo=8WwcaVdkMsNA> 70% ☆ 通知 ダウンロード 印刷 共有 設定

skl_regression.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

+ コード + テキスト
セブプレノ決証

1. 決定係数

決定係数

```
print('単回帰決定係数: %.3f, 重回帰決定係数: %.3f' % (model.score(data, target), model2.score(data2, target2)))
```

[27] # train_test_splitをインポート
from sklearn.model_selection import train_test_split

[28] # 70%を学習用、30%を検証用データにするよう分割
X_train, X_test, y_train, y_test = train_test_split(data, target,
test_size = 0.3, random_state = 666)
学習用データでパラメータ推定
model.fit(X_train, y_train)
推定したモデルの出力 (学習用、検証用モデル使用)
y_train_pred = model.predict(X_train)
y_test_pred = model.predict(X_test)

[29] # matplotlibをインポート
import matplotlib.pyplot as plt
Jupyterを利用していたら、以下のおまじないを書くくとnotebook上に図が表示
%matplotlib inline
学習用、検証用それぞれで残差をプロット
plt.scatter(y_train_pred, y_train_pred - y_train, c = 'blue', marker = 'd', label = 'Train Data')
plt.scatter(y_test_pred, y_test_pred - y_test, c = 'lightgreen', marker = 's', label = 'Test Data')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
凡例を左に表示
plt.legend(loc = 'upper left')
y = 0の直線を描く
plt.hlines(y = 0, xmin = -10, xmax = 50, lw = 2, color = 'red')
plt.xlim([10, 50])
plt.show()

● # 平均二乗誤差を評価するためのメソッドを呼び出し
from sklearn.metrics import mean_squared_error
学習用、検証用データに関して平均二乗誤差を出力
print('MSE Train: %.3f, Test: %.3f' % (mean_squared_error(y_train, y_train_pred), mean_squared_error(y_test, y_test_pred)))
学習用、検証用データに関してR^2を出力
print('R^2 Train: %.3f, Test: %.3f' % (model.score(X_train, y_train), model.score(X_test, y_test)))

MSE Train: 44.989, Test: 40.412
R^2 Train: 0.506, Test: 0.404

0秒 完了時間: 18:46

非線形回帰モデル

- 線形回帰モデルは説明変数に対する目的関数が線形のため、データの非線形に対応できない。
そこで非線形性に対応する方法が基底展開法である。基底関数によって説明変数を非線形変換し、線形回帰モデルで学習する方法。
- 基底展開法
 - ・回帰関数として、基底関数と呼ばれる既知の非線形関数とパラメータベクトルの線形結合を使用
 - ・未知のパラメータは最小二乗法や最尤法により推定する
$$y_i = f(x_i) + \varepsilon_i$$
$$y_i = \omega_0 + \text{sum}[i=1 \text{ to } m] \omega_j \phi_j(x_i) + \varepsilon_i$$
- 未学習と過学習
 - ・未学習：学習データに対して、十分小さな誤差とならない状態
 - ・過学習：学習データを忠実に再現してしまい、未知のデータに対する汎化性がなくなってしまった状態を指す
(= 訓練データに対する誤差(訓練誤差) は小さいにもかかわらず、テストデータに対する誤差(汎化誤差)が小さくならない状態)
- 過学習を防ぐ手法として正則化がある。
 - ・正則化とはモデルを学習する際に、複雑さが増すことに対するペナルティを設け、ペナルティを訓練誤差に加えた量が最も小さくなる学習モデルを求めること。
これにより汎化性能を高める。
 - ・複雑さの指標として、L1ノルム(L1正則化：ペナルティとして学習モデルのパラメータの絶対値の総和を用いる)、L2ノルム(L2正則化：ペナルティとして学習モデルのパラメータの二乗の総和を用いる)等がある。
 - ・L1正則化は特定のデータの重みを0にすることで不要なデータを削除する、L2正則化はデータの大きさに応じて0に近づけて滑らかなモデルとする特徴を持つ。

参考：

<https://qiita.com/fridericusgauss/items/3052ba024521f50bd666>

<https://qiita.com/kenta1984/items/91ab29fae8cd3920cf2b>

<https://qiho.jp/dev/serial/01/machine-learning/0009?page=3>

<https://www.techcrowd.jp/machinelearning/regularization/>

非線形回帰モデル 実装演習結果キャプチャ

out - Google ドライブ

skl_nonlinear regression.ipynb

Python 3.x - Pythonによる機械学

コルバック - Keras Documenta...

+

← → ↺ 🏠 🔒 https://colab.research.google.com/drive/1td_eyt1WK6Unt_5tXSY8lpt6yRismnnp#scrollTo=jrRZlwmCJznO 70% ☆ 📄 ⬇ 📄 📄 📄 📄

🔗 skl_nonlinear regression.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

📄 + コード + テキスト

🔍 📄 📄 📄

📄 train

📄 validation

🗨 コメント 👤 共有 ⚙️ K

✅ ディスク 📄 編集 ↶

Googleドライブのマウント

from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

+ コード + テキスト

[63]

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[64]

```
#seaborn設定
sns.set()
#背景変更
sns.set_style("darkgrid", {'grid.linestyle': '-'})
#大きさ(スクリーン変更)
sns.set_context("paper")
```

[65]

```
n=100

def true_func(x):
    z = 1-40*x+218*x**2-315*x**3+145*x**4
    return z

def linear_func(x):
    z = x
    return z
```

[66]

```
# 真の関数からノイズを伴うデータを生成


# 真の関数からデータ生成
data = np.random.rand(n).astype(np.float32)
data = np.sort(data)
target = true_func(data)

# ノイズを加える
noise = 0.5 * np.random.randn(n)
target = target + noise

# ノイズ付きデータを描画
plt.scatter(data, target)

plt.title('NonLinear Regression')
plt.legend(loc=2)
```

No handles with labels found to put in legend.
matplotlib.legend.Legend at 0x7fb6764861d0



📄 ディスク 29.61 GB 7利用可能

🔴 2 秒 完了時間 20:52

🟢 X

非線形回帰モデル 実装演習結果キャプチャ

Google Drive | skd_nonlinear_regression.ipynb | Python 3.x - Pythonによる機械学習 | コールバック - Keras Documentation

https://colab.research.google.com/drive/1td__eye1WK6Umr_5tX5Y8lpt6yRimsnp#scrollTo=IDZenfgcKHZ-

skd_nonlinear_regression.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

+ コード + テキスト


```
from sklearn.linear_model import LinearRegression

clf = LinearRegression()
data = data.reshape(-1,1)
target = target.reshape(-1,1)
clf.fit(data, target)

p_lin = clf.predict(data)

plt.scatter(data, target, label='data')
plt.plot(data, p_lin, color='darkorange', marker='-', linestyle='-', linewidth=1, markersize=6, label='linear regression')
plt.legend()
print(clf.score(data, target))
```

0.1814357346676134



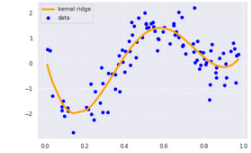
```
[68] from sklearn.kernel_ridge import KernelRidge

clf = KernelRidge(alpha=0.0002, kernel='rbf')
clf.fit(data, target)

p_kr_ridge = clf.predict(data)

plt.scatter(data, target, color='blue', label='data')
plt.plot(data, p_kr_ridge, color='orange', linestyle='-', linewidth=1, markersize=6, label='kernel ridge')
plt.legend()
plt.plot(data, p, color='orange', marker='o', linestyle='-', linewidth=1, markersize=6)

matplotlib.legend at 0x7fb6b5af6f50:
```



```
[69] #Ridge

from sklearn.metrics.pairwise import rbf_kernel
from sklearn.linear_model import Ridge

kx = rbf_kernel(X=data, Y=data, gamma=50)
```

ディスク 28.61 GB 71% 利用可能

2 秒 完了時間 20:32

非線形回帰モデル 実装演習結果キャプチャ

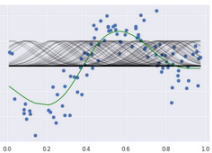
Google Drive | skl_nonlinear_regression.ipynb | Python 3.x - Pythonによる機械学習 | コールバック - Keras Documentation

https://colab.research.google.com/drive/1td_eyt1WK6Umr_5tXSY8lpt6yRmsmnp#scrollTo=IDZenfgcKHZ-

sklearn.metrics.pairwise import rbf_kernel
from sklearn.linear_model import Ridge
from sklearn.pipeline import Pipeline


```
ko = rbf_kernel(X_train, X_test, gamma=50)  
R = Ridge(alpha=10)  
R.fit(X_train, y_train)  
p_ridge = R.predict(X_test)  
plt.scatter(X_test, y_test, label='data')  
for i in range(len(X_test)):  
    plt.plot(X_test[i], color='black', linestyle='-', linewidth=1, markersize=5, label='rbf', alpha=0.2)  
plt.plot(X_test, p_ridge, color='green', linestyle='-', linewidth=1, markersize=5, label='ridge regression')  
plt.legend()  
print(R.score(X_test, y_test))
```

0.7408923324672383



```
[70] from sklearn.preprocessing import PolynomialFeatures  
from sklearn.pipeline import Pipeline
```

```
[71] PolynomialFeatures(degree=1)  
deg = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
for d in deg:  
    regr = Pipeline([  
        ('poly', PolynomialFeatures(degree=d)),  
        ('linear', LinearRegression())  
    ])  
    regr.fit(X_train, y_train)  
    # make predictions  
    p_poly = regr.predict(X_test)  
    # plot regression result  
    plt.scatter(X_test, y_test, label='data')  
    plt.plot(X_test, p_poly, label='polynomial of degree %d' % (d))
```



29.61 GB 71% 利用可能

2 秒 完了時間 20:52

非線形回帰モデル 実装演習結果キャプチャ

out - Google ドライブ

skl_nonlinear regression.ipynb

Python 3.x - Pythonによる機械学習

コルバ(ブック - Keras Documenta

← → ↺ 🏠 🔒 https://colab.research.google.com/drive/Ttd__eye1WK6Unr_5tXSY8lpt6yRmsmnp#scrollTo=IDZenfgcKHZ- 70% ☆ 🖨️ ⬇️ 📄 🌐 ⋮

skl_nonlinear regression.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

≡ ファイル

🔍 📁 📄 📂

<> > train

> validation

+ コード + テキスト

✓ 8MB ディスク

🔍 編集

[72]

```
from sklearn.metrics.pairwise import rbf_kernel
from sklearn.linear_model import Lasso

X = rbf_kernel(X=data, Y=data, gamma=5)
X = rbf_kernel(X, X)

lasso_cif = LinearRegression()
lasso_cif = Lasso(alpha=10000, max_iter=1000)
lasso_cif.fit(X, target)

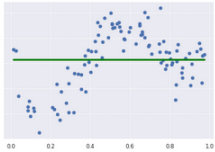
p_lasso = lasso_cif.predict(X)

plt.scatter(data, target)

plt.plot(data, p, color='green', marker='o', linestyle='-', linewidth=0.5, markersize=3)
plt.plot(data, p_lasso, color='green', linestyle='-', linewidth=0.5, markersize=3)

print(lasso_cif.score(X, target))
```

-0.881784197001252e+16



[73]

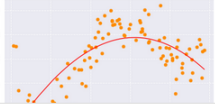
```
from sklearn import model_selection, preprocessing, linear_model, svm

# SVM=bf
cif_svr = svm.SVC(kernel='rbf', C=1e5, gamma=0.1, epsilon=0.1)
cif_svr.fit(data, target)
y_bf = cif_svr.fit(data, target).predict(data)

# plot

plt.scatter(data, target, color='darkorange', label='data')
plt.plot(data, y_bf, color='red', label='Support Vector Regression (SVM)')
plt.legend()
plt.show()
```

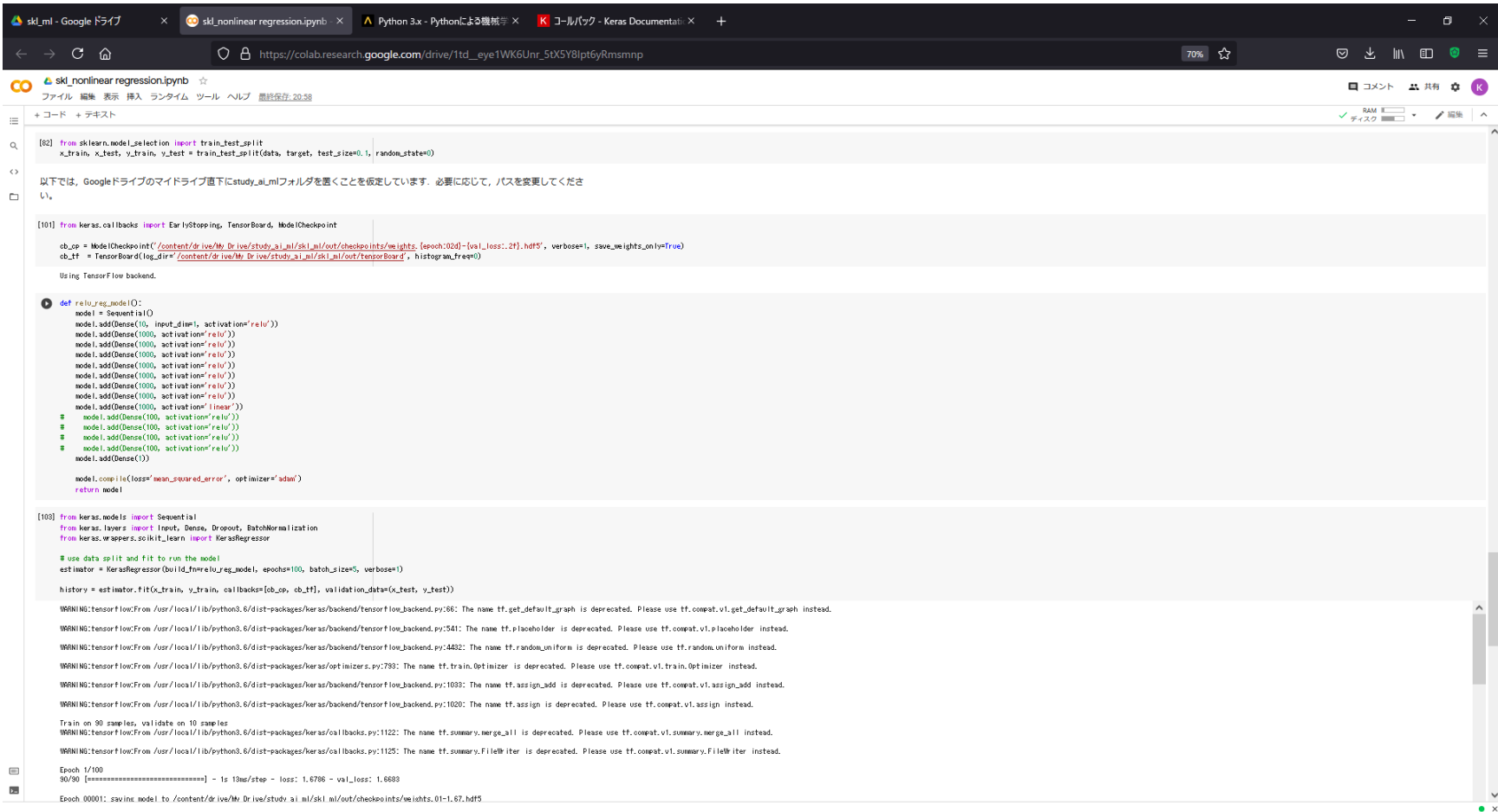
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
y = column_or_1d(y, warn=True)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
y = column_or_1d(y, warn=True)



ディスク 29.61 GB が利用可能

2 秒 完了時間 20:52

非線形回帰モデル 実装演習結果キャプチャ



ロジスティック回帰モデル

➤ ロジスティック線形回帰モデル

- ・分類問題（ある入力からクラスに分類する問題）を解くための教師あり機械学習モデル
- ・入力とm次元パラメータの線形結合をシグモイド関数に入力
- ・出力は $y=1$ になる確率の値になる

$$y = \omega^T x + \omega_0$$

➤ シグモイド関数は入力は実数、出力は必ず0～1の値

$$1 / (1 + \exp(-ax))$$

➤ シグモイド関数の性質

- ・シグモイド関数の微分はシグモイド関数自身で表現することが可能
- ・尤度関数の微分を行う際にこの事実を用いると計算が容易

➤ 最尤推定

- ・ロジスティック回帰モデルではベルヌーイ分布（確率 p で1、確率 $1-p$ で0を取る離散確率分布）を用いる。

$$P(y) = p^y (1-p)^{1-y}$$

➤ 同時確率

- ・あるデータが得られたとき、それが同時に得られる確率
- ・確率変数は独立であることを仮定すると、それぞれの確率の掛け算となる

➤ 尤度関数

- ・データは固定、パラメータを変化させる
- ・尤度関数を最大化するようなパラメータを選ぶ推定方法を最尤推定という

参考：

https://qiita.com/ONE_shoT/items/b702ab482466df6e5569

<https://waidai-csc.jp/updata/2018/08/seminar-igaku-20171222.pdf>

ロジスティック回帰モデル

- 勾配降下法
 - ・反復学習によりパラメータを逐次的に更新するアプローチのひとつ
- 確率的勾配降下法
 - ・データを一つずつランダムに選んでパラメータを更新
- 混同行列は下表のとおり
 - ・正解率： $(TP + TN) / (TP + FP + FN + TN)$ 、精度(再現率)： $TP / (TP + FP)$ 、真陽性率(適合率)： $TP / (TP + FN)$ 、真陰性率： $TN / (FP + TN)$ 、偽陰性率： $FN / (TP + FN)$ 、偽陰性率： $FP / (FP + TN)$
 - ・F値とは再現率と適合率の調和平均（逆数の平均の逆数）

		機械学習モデルの予測	
		Positive	Negative
実際のクラス	Positive	True Positive 真陽性	False Negative 偽陽性（間違えてNegativeと判別した個数）
	Negative	False Positive 偽陰性（間違えてpositiveと判別した個数）	True Negative 真陰性

ロジスティック回帰モデル 実装演習結果キャプチャ

機械学習 講義資料

skl_ml - Google ドライブ

skl_logistic_regression.ipynb - C

+

← → ↺ 🏠 🔒 https://colab.research.google.com/drive/1Xy-iqux79J7tMIM_uj4tW3Un_by4UGi#scrollTo=fp28fivH8fx 70% ☆ 🗨 📄 🌐 ☰

skl_logistic_regression.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

+ コード + テキスト

RAM 100% ディスク 100% 編集

コメント 共有 設定 K

編集するにはダブルクリックするか Enter キーを押してください

Googleドライブのマウント

```
[1] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

0. データ表示

```
[2] #from モジュール名 import クラス名 (もしくは関数名や変数名)
import pandas as pd
from pandas import DataFrame
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#matplotlibをinlineで表示するためのおまじない (plt.show() なくしていい)
%matplotlib inline
```

以下では、Googleドライブのマイドライブ直下にstudy_al_mlフォルダを置くことを仮定しています。必要に応じて、パスを変更してください。

```
[3] # titanic_data.csvファイルの読み込み
titanic_df = pd.read_csv('/content/drive/My Drive/study_al_ml/data/titanic_train.csv')
```

```
[4] # ファイルの先頭部を表示し、データセットを確認する
titanic_df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cummings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

1. ロジスティック回帰

不要なデータの削除・欠損値の補完

```
[5] #予測に不要と考えるからうをドロップ (本当はこの情報もしっかり使うべきだと思っています)
titanic_df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

#一番カラムをドロップしたデータを表示
titanic_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

7秒 完了時間: 14:55

ロジスティック回帰モデル 実装演習結果キャプチャ

機械学習 講義資料 × skl_ml - Google ドライブ × skl_logistic_regression.ipynb - C × +

← → ↺ 🏠 🔒 https://colab.research.google.com/drive/1Xy-ixdx79J7tMIM_uj4rW3Un_by4UGi#scrollTo=fp28IfivH8fx 70% ☆ 🖨 📄 🌐 🍏 ☰

skl_logistic_regression.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

+ コード + テキスト コメント 共有 設定 K

RAM 100%
ディスク 70% 編集 ↶

4 5 0 3 Allen, Mr. William Henry male 35.0 0 0 373450 8.0500 NaN S

1. ロジスティック回帰

不要なデータの削除・欠損値の補完

#字測に不要と考えるからうをドロップ (本当はこの情報もしっかり使おうべきだと思います)
titanic_df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

#→ 暫かラムをドロップしたデータを表示
titanic_df.head()

	Survived	Polclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

#nullを含んでいる行を表示
titanic_df[titanic_df.isnull().any(1)].head(10)

Survived Polclass Sex Age SibSp Parch Fare Embarked

5	0	3	male	NaN	0	0	8.4583	Q
17	1	2	male	NaN	0	0	13.0000	S
19	1	3	female	NaN	0	0	7.2250	C
26	0	3	male	NaN	0	0	7.2250	C
28	1	3	female	NaN	0	0	7.8792	Q
29	0	3	male	NaN	0	0	7.8958	S
31	1	1	female	NaN	1	0	146.5208	C
32	1	3	female	NaN	0	0	7.7500	Q
36	1	3	male	NaN	0	0	7.2292	C
42	0	3	male	NaN	0	0	7.8958	C

#Ageカラムのnullを中央値で補完
titanic_df['AgeFill'] = titanic_df['Age'].fillna(titanic_df['Age'].mean())

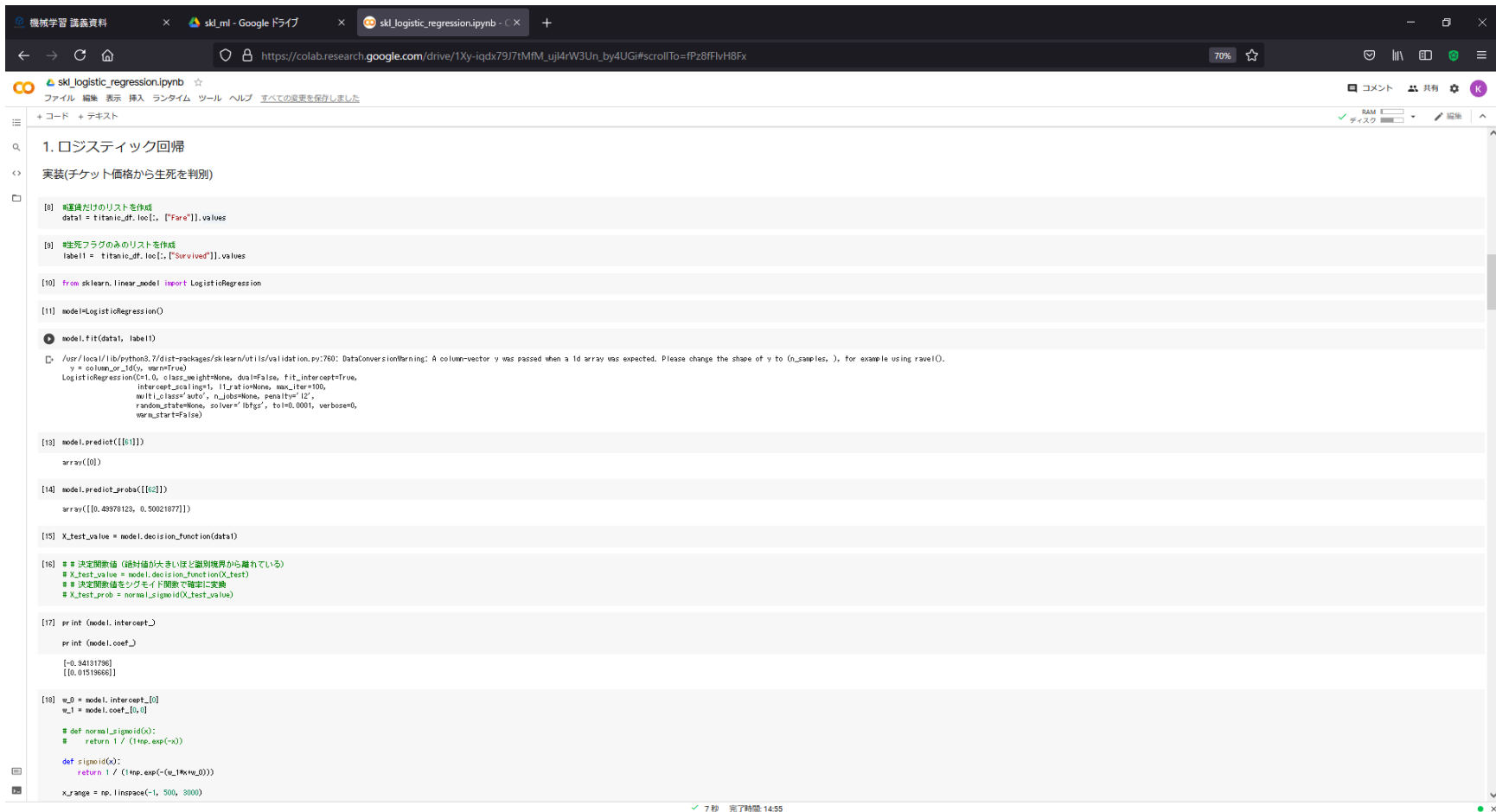
#再度nullを含んでいる行を表示 (Ageのnullは補完されている)
titanic_df[titanic_df.isnull().any(1)]

#titanic_df.dtypes

	Survived	Polclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeFill
5	0	3	male	NaN	0	0	8.4583	Q	29.699118
17	1	2	male	NaN	0	0	13.0000	S	29.699118
19	1	3	female	NaN	0	0	7.2250	C	29.699118

7秒 完了時間: 14:55

ロジスティック回帰モデル 実装演習結果キャプチャ



```
skl_logistic_regression.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
+ コード + テキスト
コメント 共有
RAM 100%
ディスク 70%
1. ロジスティック回帰
実装(チケット価格から生死を判別)
[0] # 生存だけのリストを作成
data1 = titanic_df.loc[:, ["Fare"]].values

[0] # 生死フラグのみのリストを作成
label1 = titanic_df.loc[:, ["Survived"]].values

[10] from sklearn.linear_model import LogisticRegression

[11] model=LogisticRegression()

[12] model.fit(data1, label1)

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
y = column_or_1d(y, warn=True)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, l1_regularization=0.0, l2_regularization=1.0,
multi_class='auto', n_jobs=None, penalty='l2',
random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
warm_start=False)

[13] model.predict([0])

array([0])

[14] model.predict_proba([0])

array([[0.49978123, 0.50021877]])

[15] X_test_value = model.decision_function(data1)

[16] ### 決定関数値 (値が値が大きいほど識別境界から離れている)
### X_test_value = model.decision_function(X_test)
### 決定関数値をシグモイド関数で確率に変換
### X_test_prob = normal_sigmoid(X_test_value)

[17] print (model.intercept_)

print (model.coef_)

[-0.34131796]
[0.01519666]

[18] w_0 = model.intercept_[0]
w_1 = model.coef_[0,0]

def normal_sigmoid(x):
    return 1 / (1+np.exp(-x))

def sigmoid(x):
    return 1 / (1+np.exp(-(w_1*x+w_0)))

x_range = np.linspace(-1, 500, 3000)
```

7 秒 完了時間: 14:55

ロジスティック回帰モデル 実装演習結果キャプチャ

機械学習 講義資料 × skl_ml - Google ドライブ × skl_logistic_regression.ipynb - C × +

← → ↺ 🏠 🔒 https://colab.research.google.com/drive/1Xy-igdx79J7tMM_uj4rW3Un_by4UGi#scrollTo=fp28fIvH8Fx 70% ☆ 🗨️ 📄 📁 📌 📄

skl_logistic_regression.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

+ コード + テキスト

```
[16] # # 決定関数をシグモイド関数で置き換え
# X_test_prob = normal_sigmoid(X_test_value)

[17] print (model.intercept_)

print (model.coef_)

[-0.94191786]
[[0.01519668]]

[18] w_0 = model.intercept_[0]
w_1 = model.coef_[0,0]

# def normal_sigmoid(x):
#     return 1 / (1+np.exp(-x))

def sigmoid(x):
    return 1 / (1+np.exp(-(w_1*x+w_0)))

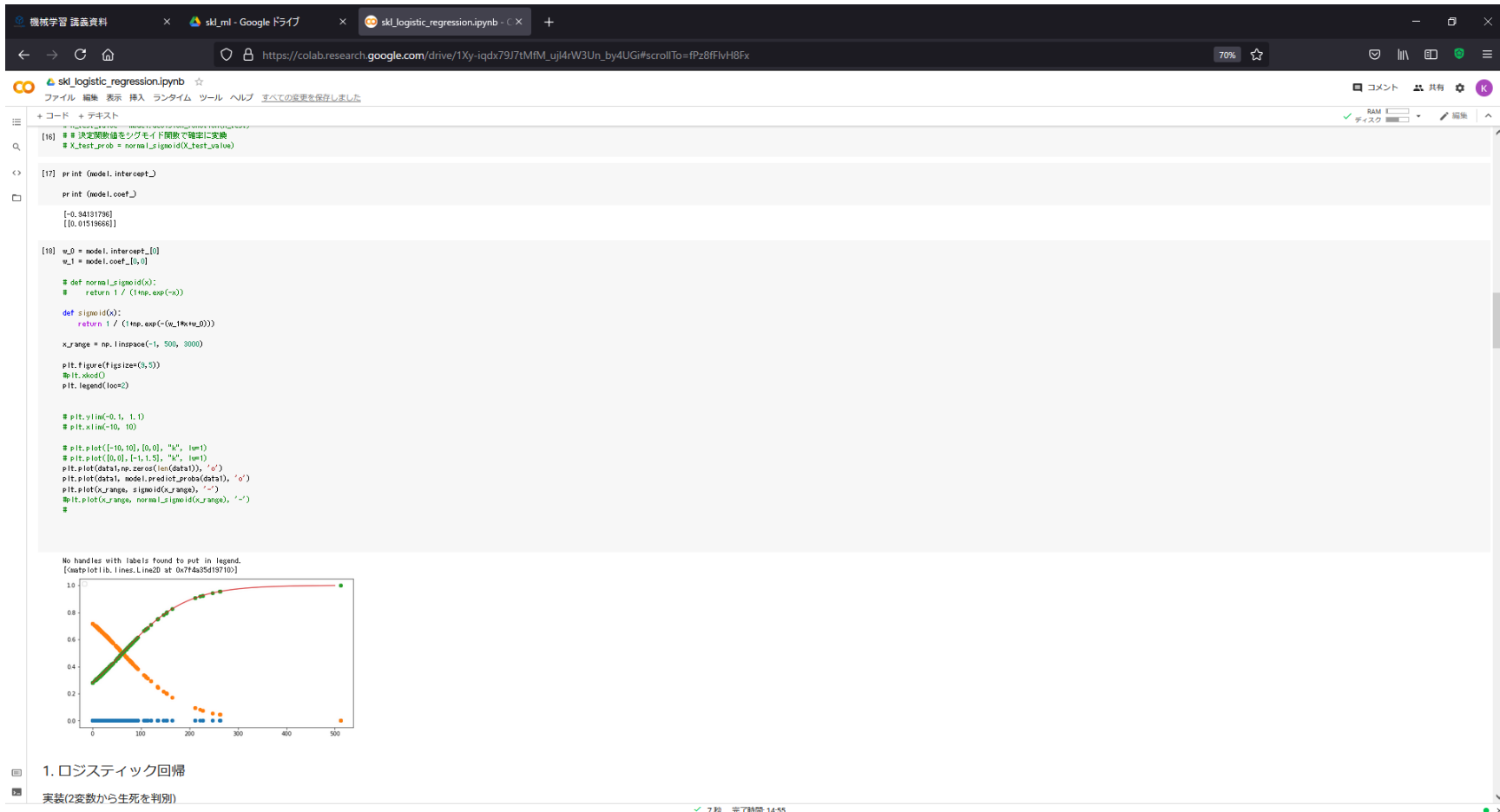
x_range = np.linspace(-1, 500, 3000)

plt.figure(figsize=(5,5))
plt.plot(x_range)
plt.legend(loc=2)

# plt.ylim(-0.1, 1.1)
# plt.xlim(-10, 10)

# plt.plot([-10,10],[0,0], "k", lw=1)
# plt.plot([0,0],[-1,1.5], "k", lw=1)
plt.plot(data[:,np.zeros(len(data)), 'x'])
plt.plot(data[:, model.predict_proba(data), 'o'])
plt.plot(x_range, sigmoid(x_range), "-")
plt.plot(x_range, normal_sigmoid(x_range), "-")

No handles with labels found to put in legend.
[Outplotlib.lines.Line2D at 0x7f4b35d19710]
```



1. ロジスティック回帰

実装(2変数から生死を判別)

✓ 7秒 完了時間 14:55

ロジスティック回帰モデル 実装演習結果キャプチャ

機械学習 講義資料 × skl_ml - Google ドライブ × skl_logistic_regression.ipynb - C × +

← → ↺ 🏠 🔒 https://colab.research.google.com/drive/1Xy-idx79J7tMIM_uj4tW3Un_by4UGi#scrollTo=fp28IfvH8fx 70% ☆ 🗨 📄 📁 🌐 ☰

skl_logistic_regression.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

+ コード + テキスト

コメント 共有 設定 K

RAM 100% ディスク 100%

🔍 🔖 🗑

1. ロジスティック回帰
実装(2変数から生死を判別)

[] # AgeFillの欠損値を埋めたので
titanic_df = titanic_df.drop(['Age'], axis=1)

[19] titanic_df['gender'] = titanic_df['Sex'].map({'female': 0, 'male': 1}).astype(int)

[20] titanic_df.head(3)

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeFill	Gender
0	0	3	male	22.0	1	0	7.2500	S	22.0	1
1	1	1	female	38.0	1	0	71.2833	C	38.0	0
2	1	3	female	26.0	0	0	7.9250	S	26.0	0

[21] titanic_df['Pclass_Gender'] = titanic_df['Pclass'] + titanic_df['Gender']

[22] titanic_df.head()

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	AgeFill	Gender	Pclass_Gender
0	0	3	male	22.0	1	0	7.2500	S	22.0	1	4
1	1	1	female	38.0	1	0	71.2833	C	38.0	0	1
2	1	3	female	26.0	0	0	7.9250	S	26.0	0	3
3	1	1	female	35.0	1	0	53.1000	S	35.0	0	1
4	0	3	male	35.0	0	0	8.0500	S	35.0	1	4

[23] titanic_df = titanic_df.drop(['Pclass', 'Sex', 'Gender', 'Age'], axis=1)

[24] titanic_df.head()

	Survived	SibSp	Parch	Fare	Embarked	AgeFill	Pclass_Gender
0	0	1	0	7.2500	S	22.0	4
1	1	1	0	71.2833	C	38.0	1
2	1	0	0	7.9250	S	26.0	3
3	1	1	0	53.1000	S	35.0	1
4	0	0	0	8.0500	S	35.0	4

[] # 重宝だよ!!!
境界線の様式
$w_1 \cdot x + w_2 \cdot y + w_0 = 0$
$\Rightarrow y = (-w_1 \cdot x - w_0) / w_2$

境界線 プロット
plt.plot([-2,2], map(lambda x: (-w_1 * x - w_0) / w_2, [-2,2]))

7秒 完了時間: 14.55

ロジスティック回帰モデル 実装演習結果キャプチャ

機械学習 講義資料

skl_ml - Google ドライブ

skl_logistic_regression.ipynb - C x

+

← → ↺ 🏠 🔒 https://colab.research.google.com/drive/1Xy-idx79j7tMiM_uj4rW3Un_by4UGi#scrollTo=fPz8BfivH8Fx 70% ☆ 📄 📏 📄 📄 📄 📄

skl_logistic_regression.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

+ コード + テキスト

RAM 100%
ディスク 100% 編集 ↗

```
[25] np.random.seed = 0

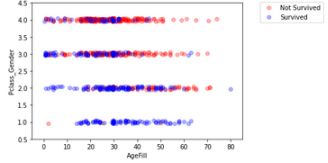
xmin, xmax = -5, 85
ymin, ymax = 0.5, 4.5

index_survived = titanic_df[titanic_df["Survived"]==0].index
index_not_survived = titanic_df[titanic_df["Survived"]==1].index

from matplotlib.colors import ListedColorMap
fig, ax = plt.subplots()
cm = plt.cm.Bu
cm_bright = ListedColorMap(['#FF0000', '#0000FF'])
so = ax.scatter(titanic_df.loc[index_survived, "AgeFill"],
                titanic_df.loc[index_survived, "Pclass_Sender"]*(np.random.rand(len(index_survived))-0.5)*0.1,
                color="r", label="Not Survived", alpha=0.3)
so = ax.scatter(titanic_df.loc[index_not_survived, "AgeFill"],
                titanic_df.loc[index_not_survived, "Pclass_Sender"]*(np.random.rand(len(index_not_survived))-0.5)*0.1,
                color="b", label="Survived", alpha=0.3)

ax.set_xlabel("AgeFill")
ax.set_ylabel("Pclass_Sender")
ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)
ax.legend(bbox_to_anchor=(1.4, 1.03))

<matplotlib.legend.Legend at 0x74a35db6950>
```



```
[26] #演習だけのリストを作成
data2 = titanic_df.loc[:, ["AgeFill", "Pclass_Sender"]].values

[27] data2

array([[22.      ,  4.      ],
       [58.      ,  1.      ],
       [26.      ,  3.      ],
       ...,
       [29.69011785,  3.      ],
       [26.      ,  2.      ],
       [32.      ,  4.      ]])

[28] #全行フラグのみのリストを作成
label2 = titanic_df.loc[:, ["Survived"]].values

[29] model2 = LogisticRegression()

[30] model2.fit(data2, label2)

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  y = column_or_1d(y, warn=True)
```

✓ 7秒 完了時間 14:55

ロジスティック回帰モデル 実装演習結果キャプチャ

機械学習 講義資料 x skl_ml - Google ドライブ x skl_logistic_regression.ipynb - C x +

← → ↺ 🏠 🔒 https://colab.research.google.com/drive/1Xy-idx797tMIM_uj4rW3Un_by4UGi#scrollTo=ukigMWE_H8FH 70% ☆ 🖨 📄 🗨 🍏 ☰

🔗 skl_logistic_regression.ipynb ☆
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ 全ての変更を保存しました

+ コード + テキスト

🔍 🔑 📁

```
[30] model2.fit(data2, label2)

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:760: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
y = column_or_1d(y, warn=True)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_regularization=0.0, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tolb=0.0001, verbose=0,
                    warm_start=False)

[31] model2.predict([[10, 1]])

array([1])

[32] model2.predict_proba([[10, 1]])

array([[0.03754748, 0.96245251]])
```

🔍 titanic_df.head(3)

	Survived	SibSp	Parch	Fare	Embarked	AgeF III	Pclass	Gender
0	0	1	0	7.2500	S	22.0		4
1	1	1	0	71.2833	C	38.0		1
2	1	0	0	79.250	S	26.0		3

```
[34] h = 0.02
xmin, xmax = -5, 95
ymin, ymax = 0.5, 4.5
xx, yy = np.meshgrid(np.arange(xmin, xmax, h), np.arange(ymin, ymax, h))
Z = model2.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
Z = Z.reshape(xx.shape)

fig, ax = plt.subplots()
levels = np.linspace(0, 1.0)
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
contour = ax.contourf(xx, yy, Z, cmap=cm, levels=levels, alpha=0.5)

so = ax.scatter(titanic_df.loc[index_survived, 'AgeF III'],
                titanic_df.loc[index_survived, 'Pclass_Gender']) * (np.random.rand(len(index_survived)) - 0.5) * 0.1,
                color='r', label='Not Survived', alpha=0.3)
so = ax.scatter(titanic_df.loc[index_not_survived, 'AgeF III'],
                titanic_df.loc[index_not_survived, 'Pclass_Gender']) * (np.random.rand(len(index_not_survived)) - 0.5) * 0.1,
                color='b', label='Survived', alpha=0.3)

ax.set_xlabel('AgeF III')
ax.set_ylabel('Pclass_Gender')
ax.set_xlim(xmin, xmax)
ax.set_ylim(ymin, ymax)
fig.colorbar(contour)

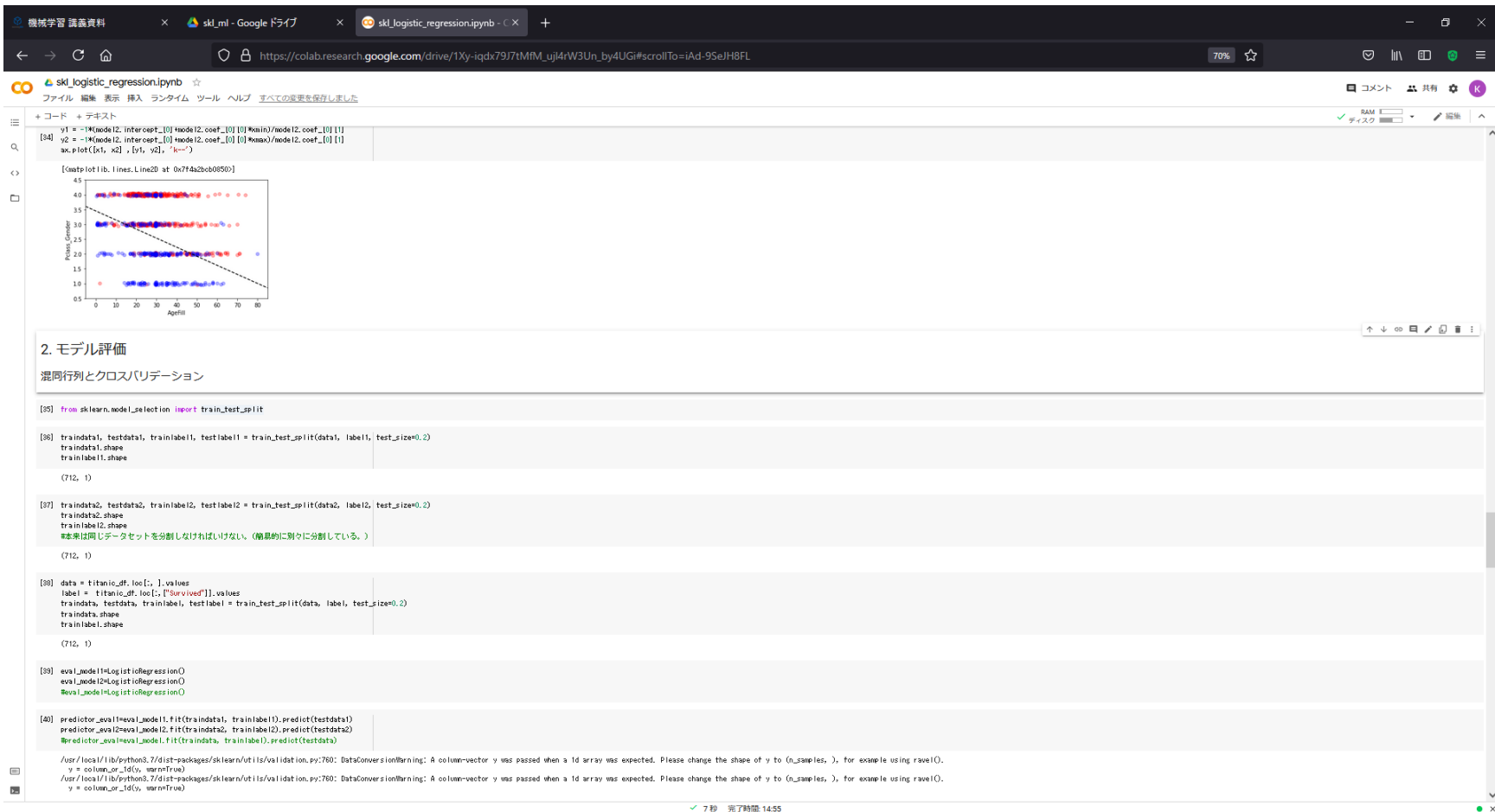
x1 = xmin
x2 = xmax
y1 = -1 * (model2.intercept_[0] + model2.coef_[0][0] * xmin) / model2.coef_[0][1]
y2 = -1 * (model2.intercept_[0] + model2.coef_[0][0] * xmax) / model2.coef_[0][1]
ax.plot([x1, x2], [y1, y2], 'k--')

[Outplotlib, lines.Line2D at 0x7f4a2cb00500]
```

43

7 秒 完了時間: 14:55

ロジスティック回帰モデル 実装演習結果キャプチャ



ロジスティック回帰モデル 実装演習結果キャプチャ

機械学習 講義資料 × sk_ml - Google ドライブ × sk_logistic_regression.ipynb - C × +

← → ↺ 🏠 🔒 https://colab.research.google.com/drive/1Xy-idx79J7tMIM_uj4rW3Un_by4UGi#scrollTo=u7Q4CjPLH8Fc 70% ☆ 🖨️ 📄 🗨️ 🌐 ☰

sk_logistic_regression.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

+ コード + テキスト

RAM 100% ディスク 100%

```
[41] eval_model1.score(traindata1, trainlabel1)
0.6741573033707865

[42] eval_model1.score(testdata1, testlabel1)
0.6759776536312848

[43] eval_model2.score(traindata2, trainlabel2)
0.7626404494382022

[44] eval_model2.score(testdata2, testlabel2)
0.7932960893854748

[45] from sklearn import metrics
print(metrics.classification_report(testlabel1, predictor_eval1))
print(metrics.classification_report(testlabel2, predictor_eval2))

      precision    recall  f1-score   support

      0       0.68       0.91       0.78       113
      1       0.64       0.27       0.38        66

 accuracy: 0.66
 macro avg: 0.66   0.59   0.58   179
 weighted avg: 0.67   0.68   0.63   179

      precision    recall  f1-score   support

      0       0.87       0.82       0.85       125
      1       0.64       0.72       0.68        54

 accuracy: 0.79
 macro avg: 0.76   0.77   0.76   179
 weighted avg: 0.80   0.79   0.80   179

[46] from sklearn.metrics import confusion_matrix
confusion_matrix1=confusion_matrix(testlabel1, predictor_eval1)
confusion_matrix2=confusion_matrix(testlabel2, predictor_eval2)

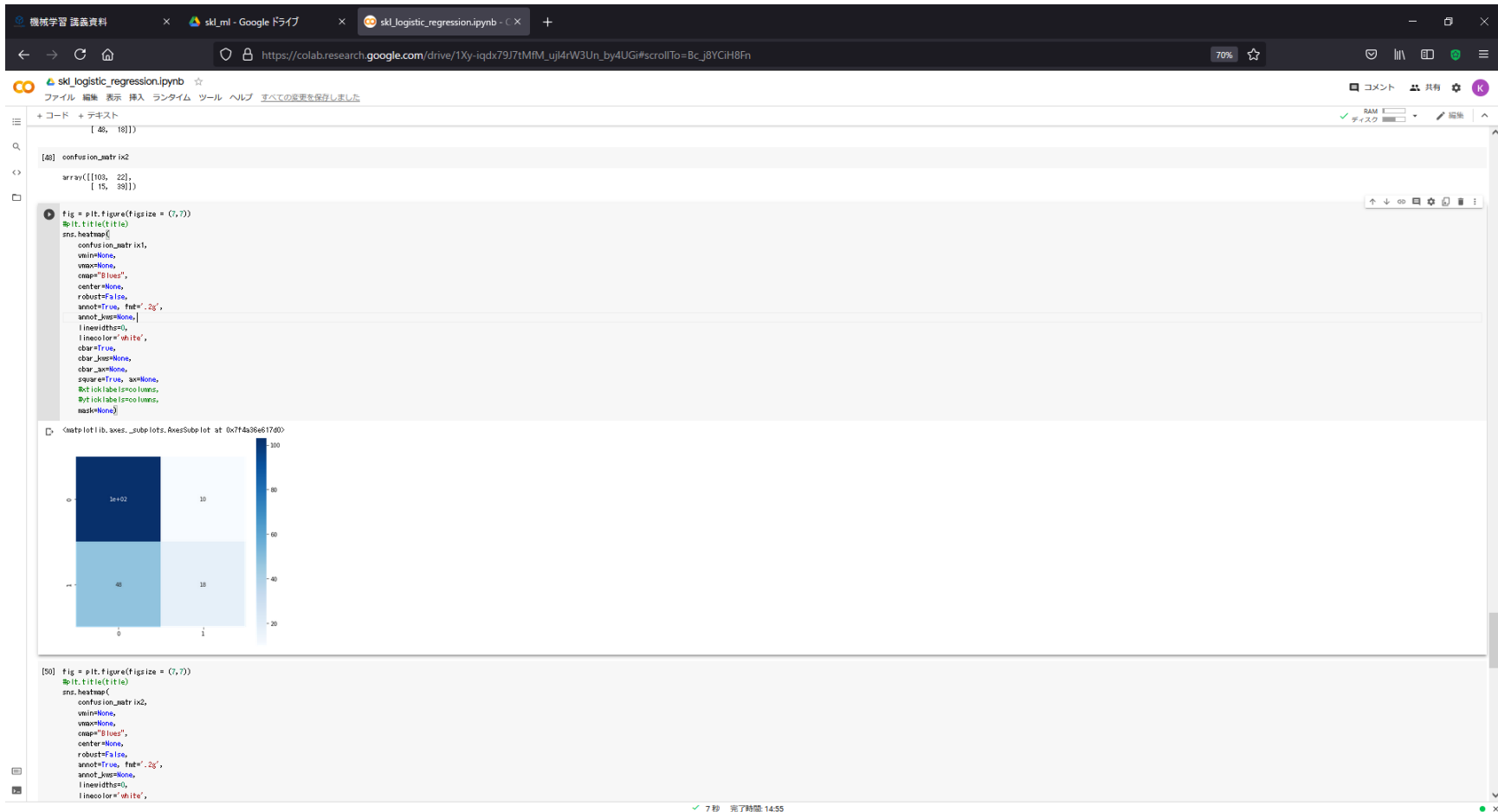
[47] confusion_matrix1
array([[103, 10],
       [ 48, 18]])

[48] confusion_matrix2
array([[103, 22],
       [ 15, 39]])

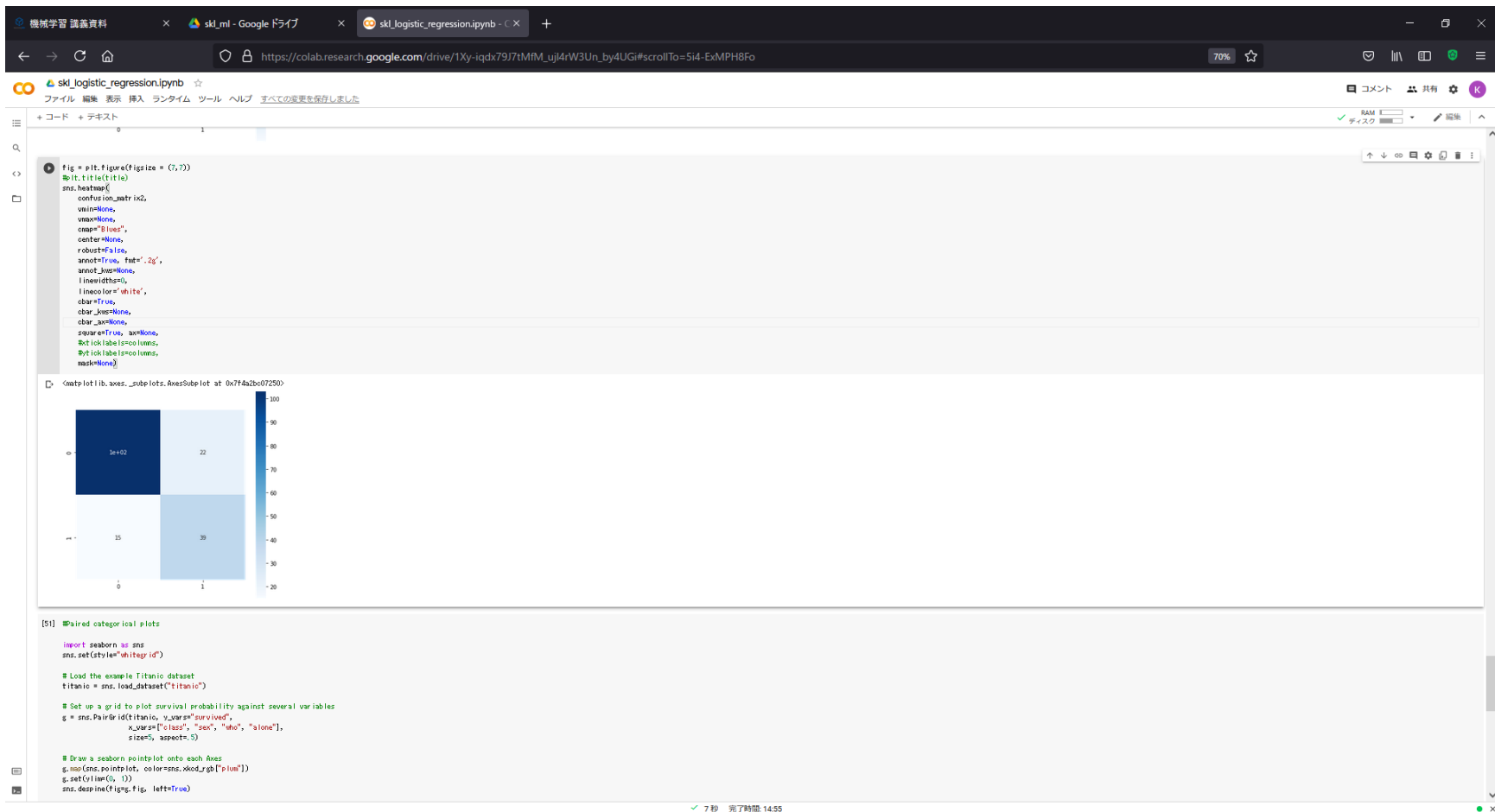
[49] fig = plt.figure(figsize = (7,7))
plt.title(title)
sns.heatmap(
    confusion_matrix1,
    vmin=None,
    vmax=None,
    cmap='Blues',
    center=None,
    annot=True,
```

7 秒 完了時間 14:55

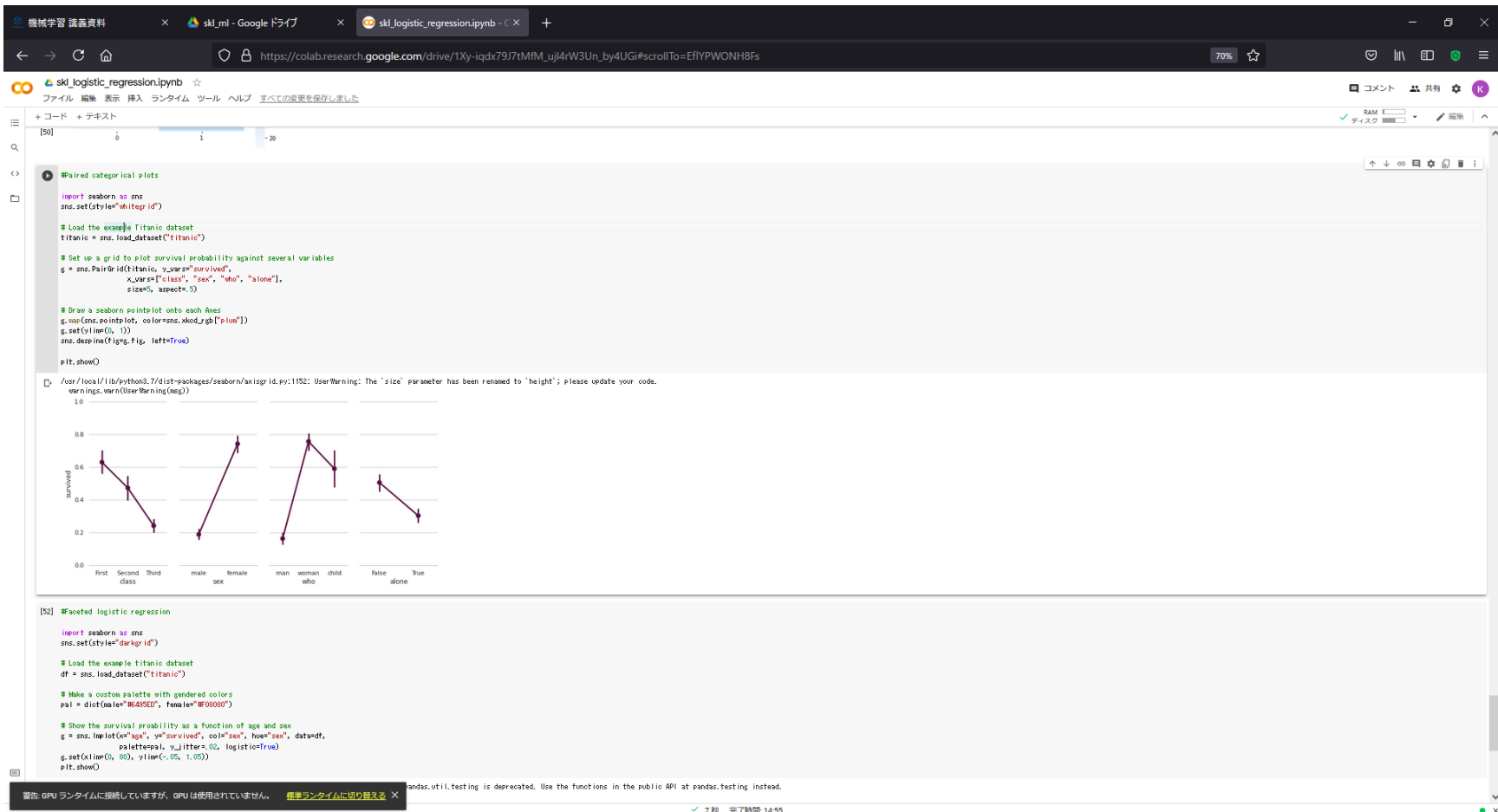
ロジスティック回帰モデル 実装演習結果キャプチャ



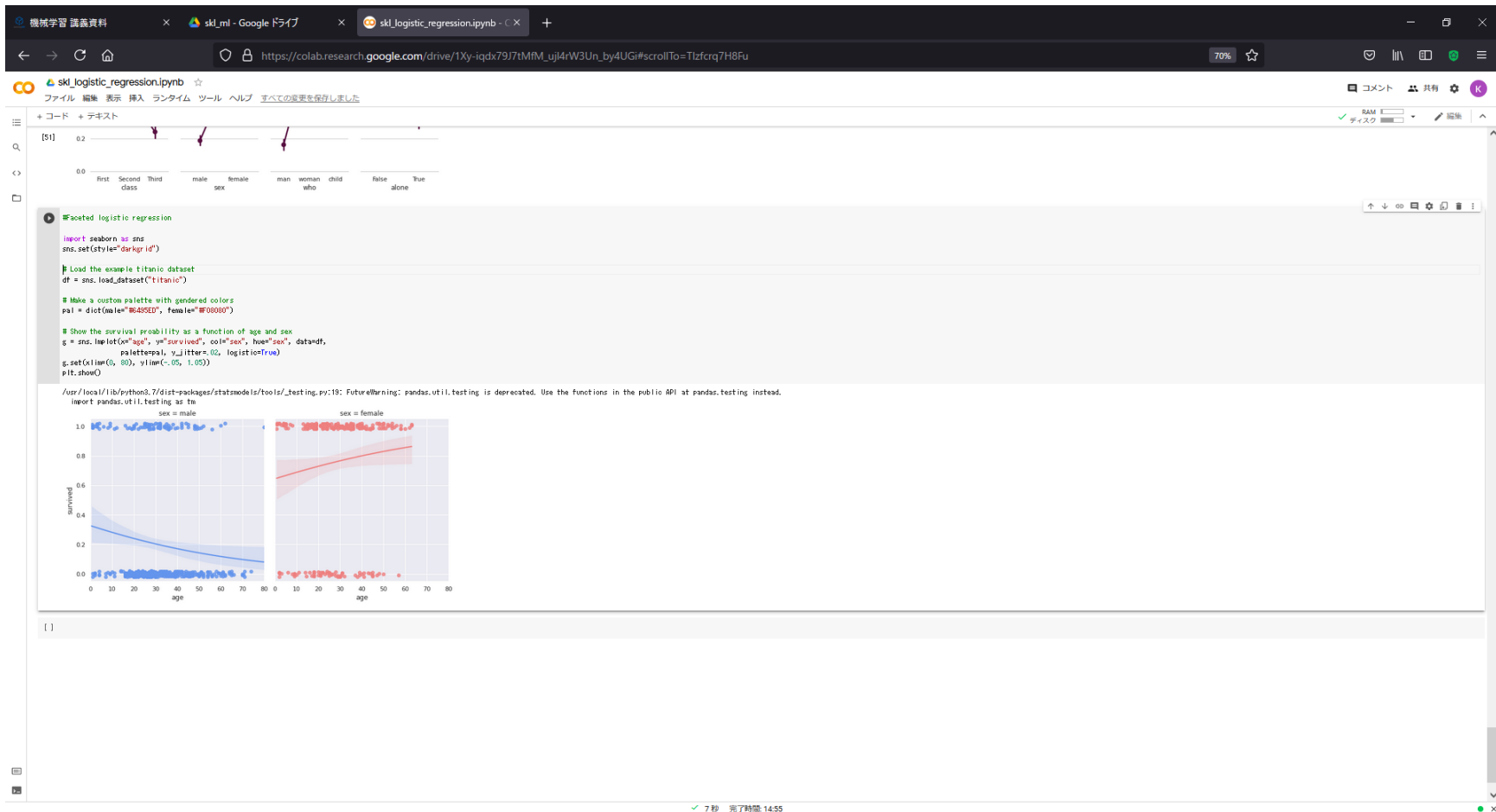
ロジスティック回帰モデル 実装演習結果キャプチャ



ロジスティック回帰モデル 実装演習結果キャプチャ



ロジスティック回帰モデル 実装演習結果キャプチャ



主成分分析

- 主成分分析は多変量データの持つ構造をより少数個の指標に圧縮する（次元の縮約）
 - ・変量の個数を減らすに伴う、情報の損失をなるべく小さくしたい
 - ・少数変数を利用した分析や可視化が実現できる（視覚化により、データが持つ情報を解釈しやすくなる）
- 主成分を与える変換は、第一主成分の分散を最大化し、続く主成分はそれまでに決定した主成分と直交する条件の下で、分散を最大化するように求める。
- 寄与率（第k主成分が持つ情報量の割合）：
 - 第1～次元分の主成分の分散は、元のデータの分散と一致
 - ・2次元のデータを2次元の主成分で表示したとき、固有値の和と元のデータの分散が一致
 - ・第k主成分の分散は主成分に対応する固有値

主成分分析 実装演習結果キャプチャ

skl_pca.ipynb

ファイル編集表示挿入ランタイムツールヘルプ

目次

Googleドライブのマウント
sys.pathの設定
セクション

+コード+テキスト

[] <https://ekke.hateblo.jp/entry/2017/08/11/230000>を参考に利用しています。

Googleドライブのマウント

```
[1] from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive
```

sys.pathの設定

以下では、Googleドライブのマイドライブ直下にstudy_mlフォルダを書くことを仮定しています。必要に応じて、パスを変更してください。

```
[3] cancer_df = pd.read_csv('/content/drive/My Drive/study_ml/data/cancer.csv')

[4] print('cancer df shape: {}'.format(cancer_df.shape))

cancer df shape: (569, 33)

[5]
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fractal_dimension_mean	radius_se	texture_se	perimeter_se	area_se	smoothness_se	compactness_se	concavity_se	concave points_se
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	1.0950	0.9053	8.589	153.40	0.006399	0.04904	0.05373	0.01860
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	0.5435	0.7339	3.398	74.08	0.005225	0.01308	0.01860	0.03832
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	0.7456	0.7869	4.585	94.03	0.006150	0.04005	0.03832	0.05661
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	0.4956	1.1560	3.445	27.23	0.009110	0.07458	0.05661	0.05688
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	0.7572	0.7813	5.438	94.44	0.011490	0.02461	0.05688	0.05198
...
564	926424	M	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	1.1760	1.2560	7.673	158.70	0.010300	0.02891	0.05198	0.03950
565	926682	M	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	0.7655	2.4630	5.203	99.04	0.005769	0.02423	0.03950	0.04730
566	926954	M	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	0.4564	1.0750	3.425	48.55	0.005903	0.03731	0.04730	0.07117
567	927241	M	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	0.7260	1.5950	5.772	86.22	0.006522	0.06158	0.07117	0.00000
568	92751	B	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	0.3857	1.4280	2.548	19.15	0.007189	0.00466	0.00000	

569 rows x 33 columns

0秒 完了時間 8.41

主成分分析 実装演習結果キャプチャ

sklearn_pca.py

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

目次

Googleドライブのマウント

sys.pathの追加

セクション

code

text

6

cancer_df.drop('Unnamed: 0', axis=1, inplace=True)

cancer_df

id diagnosis radius_mean texture_mean perimeter_mean area_mean smoothness_mean compactness_mean concavity_mean concave points_mean symmetry_mean fractal_dimension_mean radius_se texture_se perimeter_se area_se smoothness_se compactness_se concavity_se

0 842302 M 17.99 10.38 122.80 1001.0 0.11840 0.27760 0.30010 0.14710 0.2419 0.07871 1.0950 0.9053 8.589 153.40 0.006399 0.04904 0.05373

1 842517 M 20.57 17.77 132.90 1326.0 0.08474 0.07864 0.08690 0.07017 0.1812 0.05667 0.5435 0.7339 3.398 74.08 0.005225 0.01308 0.01860

2 8430903 M 19.69 21.25 130.00 1203.0 0.10960 0.15990 0.19740 0.12790 0.2069 0.05999 0.7456 0.7869 4.585 94.03 0.006150 0.04006 0.03832

3 84348301 M 11.42 20.38 77.58 386.1 0.14250 0.28390 0.24140 0.10520 0.2597 0.09744 0.4956 1.1560 3.445 27.23 0.009110 0.07458 0.05661

4 84358402 M 20.29 14.34 135.10 1297.0 0.10030 0.13280 0.19800 0.10430 0.1809 0.05883 0.7572 0.7813 5.438 94.44 0.011490 0.02461 0.05688

...

564 926424 M 21.56 22.39 142.00 1479.0 0.11100 0.11590 0.24390 0.13890 0.1726 0.05623 1.1760 1.2560 7.673 158.70 0.010300 0.02891 0.05198

565 926682 M 20.13 28.25 131.20 1261.0 0.09780 0.10340 0.14400 0.09791 0.1752 0.05533 0.7655 2.4630 5.203 99.04 0.005769 0.02423 0.03950

566 926954 M 16.60 28.08 108.30 858.1 0.08455 0.10230 0.09251 0.05302 0.1590 0.05648 0.4564 1.0750 3.425 48.55 0.005903 0.03731 0.04730

567 927241 M 20.60 29.33 140.10 1265.0 0.11780 0.27700 0.35140 0.15200 0.2397 0.07016 0.7260 1.5950 5.772 86.22 0.006522 0.06158 0.07117

568 92751 B 7.76 24.54 47.92 181.0 0.05263 0.04362 0.00000 0.00000 0.1587 0.05884 0.3857 1.4280 2.548 19.15 0.007189 0.00466 0.00000

569 rows x 32 columns

diagnosis: 診断結果 (良性がB / 悪性がM) ・説明変数は3列以降、目的変数を2列目としロジスティック回帰で分類

7

目的変数の抽出

y = cancer_df.diagnosis.apply(lambda d: 1 if d == 'M' else 0)

8

説明変数の抽出

X = cancer_df.loc[:, 'radius_mean':]

9

学習用とテスト用でデータを分割

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

標準化

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)

ロジスティック回帰で学習

logistic = LogisticRegressionCV(solver='lbfgs', random_state=0)

logistic.fit(X_train_scaled, y_train)

検証

print('Train score: {:.3f}'.format(logistic.score(X_train_scaled, y_train)))

print('Test score: {:.3f}'.format(logistic.score(X_test_scaled, y_test)))

print('Confusion matrix: \n{}'.format(confusion_matrix(y_train, y_test)))

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

Please also refer to the documentation for alternative solver options:

https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

extra_warning_msg='LOGISTIC_SOLVER_CONVERGENCE_MSG')

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:340: ConvergenceWarning: lbfgs failed to converge (status=1):

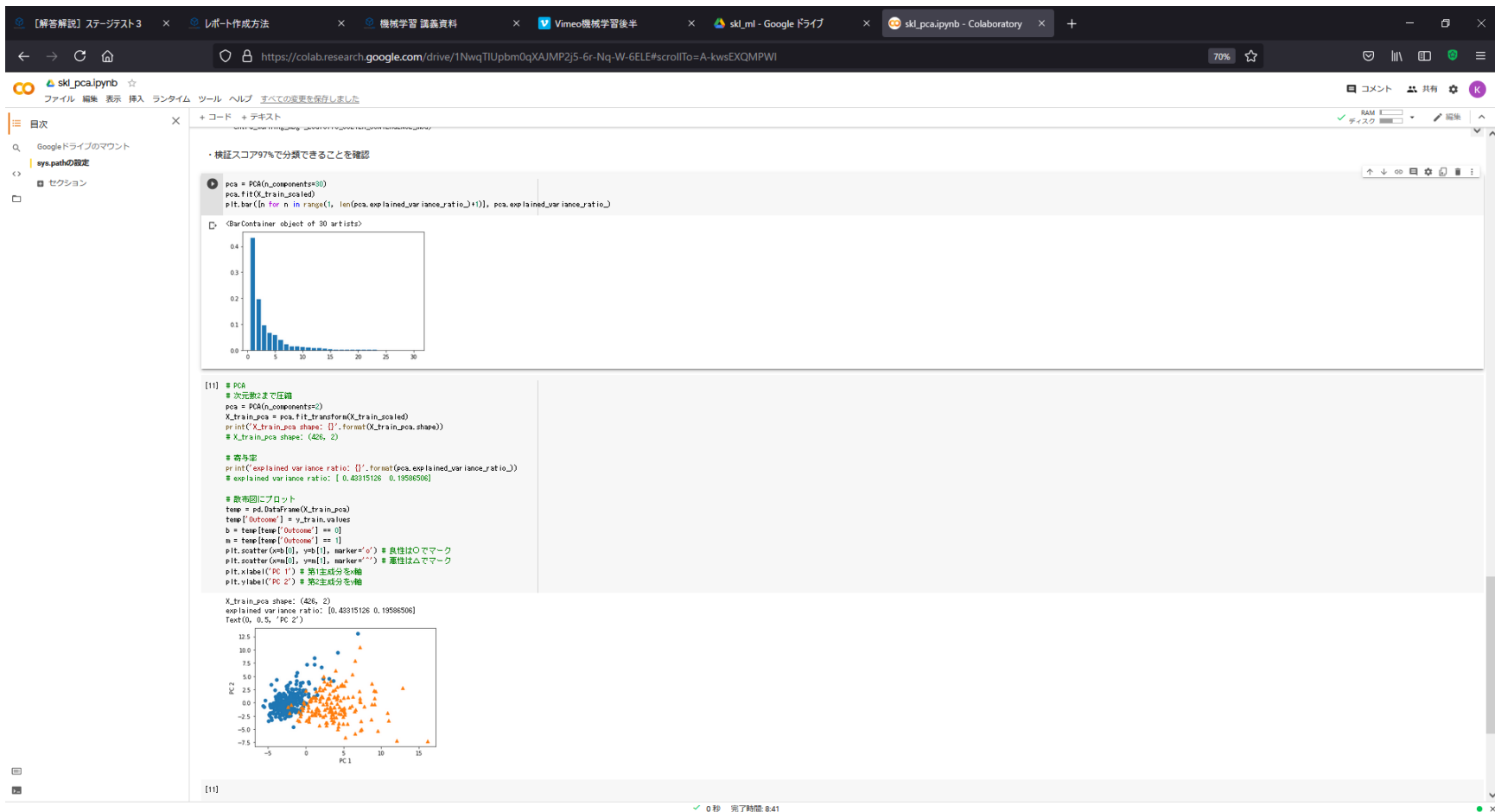
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:

<https://scikit-learn.org/stable/modules/preprocessing.html>

0秒 完了時間: 8:41

主成分分析 実装演習結果キャプチャ



アルゴリズム

- k近傍法（教師あり学習の分類問題）
 - ・最近傍のデータをk個とってきて、それらが最も多く所属するクラスに識別する
 - ・kの値を変化させると結果も変わる
- k-means（教師なし学習でクラスタリング(特徴の似ているもの同士をグループ化する)手法)
- k近傍法とk-meansの両方に当てはまる特徴として、いずれのアルゴリズムも距離計算を行うことがあげられる

アルゴリズム(k近傍法) 実装演習結果キャプチャ

np_knn.ipynb

ファイル 編集 表示 挿入 ランタイム ツール ヘルプ

目次

k近傍法

訓練データ生成

学習

予測

RAM 1 GB

ディスク 10 GB

コメント

共有

設定

ヘルプ

https://colab.research.google.com/drive/1HuV56L4kmLTn49EYEsVxsDfPuUCgzZg#scrollTo=ETfJRzAt4IM

70%

☆

🔄

📄

🌱

☰

np_knn.ipynb

+ コード + テキスト

▼ k近傍法

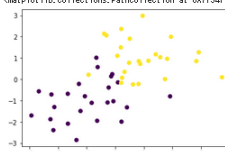
```
[1] %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

▼ 訓練データ生成

```
[2] def gen_data():
    x0 = np.random.normal(size=50).reshape(-1, 2) - 1
    x1 = np.random.normal(size=50).reshape(-1, 2) + 1
    x_train = np.concatenate([x0, x1])
    y_train = np.concatenate([np.zeros(50), np.ones(50)]).astype(np.int)
    return x_train, y_train

[3] X_train, y_train = gen_data()
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)

<matplotlib.collections.PathCollection at 0x7f54f1b23710>
```



▼ 学習

既に訓練ステップはない

▼ 予測

予測するデータ点との、距離が最も近いk個の、訓練データのラベルの最頻値を割り当てる

```
[4] def distance(x1, x2):
    return np.sum((x1 - x2)**2, axis=1)

def knn_predict(n_neighbors, x_train, y_train, X_test):
    y_pred = np.empty(len(X_test), dtype=y_train.dtype)
    for i, x in enumerate(X_test):
        distances = distance(x, X_train)
        nearest_index = distances.argsort()[:n_neighbors]
        mode, _ = stats.mode(y_train[nearest_index])
        y_pred[i] = mode
```

アルゴリズム(k近傍法) 実装演習結果キャプチャ

【解答解説】 ステージテスト 3

レポート作成方法

機械学習 講義資料

Vimeo機械学習後半

np_ml - Google ドライブ

np_knn.ipynb - Colaboratory

np_svm.ipynb - Colaboratory

+

← → 🏠 🔒 <https://colab.research.google.com/drive/THuV56L4kmLTn49EYEsSVxsDFPuUCgZg#scrollTo=ETfjRsZAt4IM> 70% ☆ 🖨 📄 🗨 🍏

np_knn.ipynb ☆
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ

📄 目次 + コード + テキスト

🔍 k近傍法
📁 訓練データ生成
📁 学習
📁 予測
📁 numpy実装
📁 セクション

📄 コメント 共有 ⚙️ K

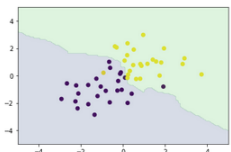
🟢 RAM 100% 🟢 ディスク 100% 📄 編集 ^

▼ 予測

予測するデータ点との、距離が最も近いk個の、訓練データのラベルの最頻値を割り当てる

```
[4] def distance(x1, x2):  
    return np.sum((x1 - x2)**2, axis=1)  
  
def knn_predict(n_neighbors, x_train, y_train, X_test):  
    y_pred = np.empty(len(X_test), dtype=y_train.dtype)  
    for i, x in enumerate(X_test):  
        distances = distance(x, X_train)  
        nearest_index = distances.argsort()[n_neighbors]  
        mode, _ = stats.mode(y_train[nearest_index])  
        y_pred[i] = mode  
    return y_pred  
  
def plt_result(x_train, y_train, y_pred):  
    x0, x1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))  
    xx = np.array([x0, x1]).reshape(2, -1).T  
    plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)  
    plt.contourf(x0, x1, y_pred.reshape(100, 100).astype(dtype=np.float), alpha=0.2, levels=np.linspace(0, 1, 2))
```

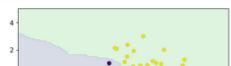
```
[5] n_neighbors = 3  
  
x0, x1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))  
X_test = np.array([x0, x1]).reshape(2, -1).T  
  
y_pred = knn_predict(n_neighbors, X_train, y_train, X_test)  
plt_result(X_train, y_train, y_pred)
```



▼ numpy実装

```
[6] x0, x1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))  
xx = np.array([x0, x1]).reshape(2, -1).T
```

```
[7] from sklearn.neighbors import KNeighborsClassifier  
kne = KNeighborsClassifier(n_neighbors=n_neighbors).fit(X_train, y_train)  
plt_result(X_train, y_train, kne.predict(xx))
```



🟢 0 秒 完了時間 8:47

🟢 X

アルゴリズム(k近傍法) 実装演習結果キャプチャ

【解答解説】ステージテスト3 × レポート作成方法 × 機械学習 講義資料 × Vimeo機械学習後半 × np_ml - Google ドライブ × np_knn.ipynb - Colaboratory × np_svm.ipynb - Colaboratory × +

← → ↺ 🏠 <https://colab.research.google.com/drive/1HuV56L4kmLtn49EYEeSVxsDfPuUCgzZg#scrollTo=ETfjRsZA4IM> 70% ☆

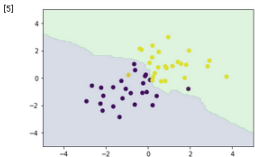
np_knn.ipynb ☆
ファイル 編集 表示 挿入 ランタイム ツール ヘルプ すべての変更を保存しました

📄 コメント 👤 共有 ⚙️ K

📄 目次 × + コード + テキスト

🔍 k近傍法
🔍 訓練データ生成
🔍 学習
🔍 予測
📄 numpy実装
📄 セクション

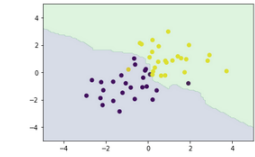
[5]



numpy実装

```
[6] x00, x01 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))  
xx = np.array([x00, x01]).reshape((-1), T)
```

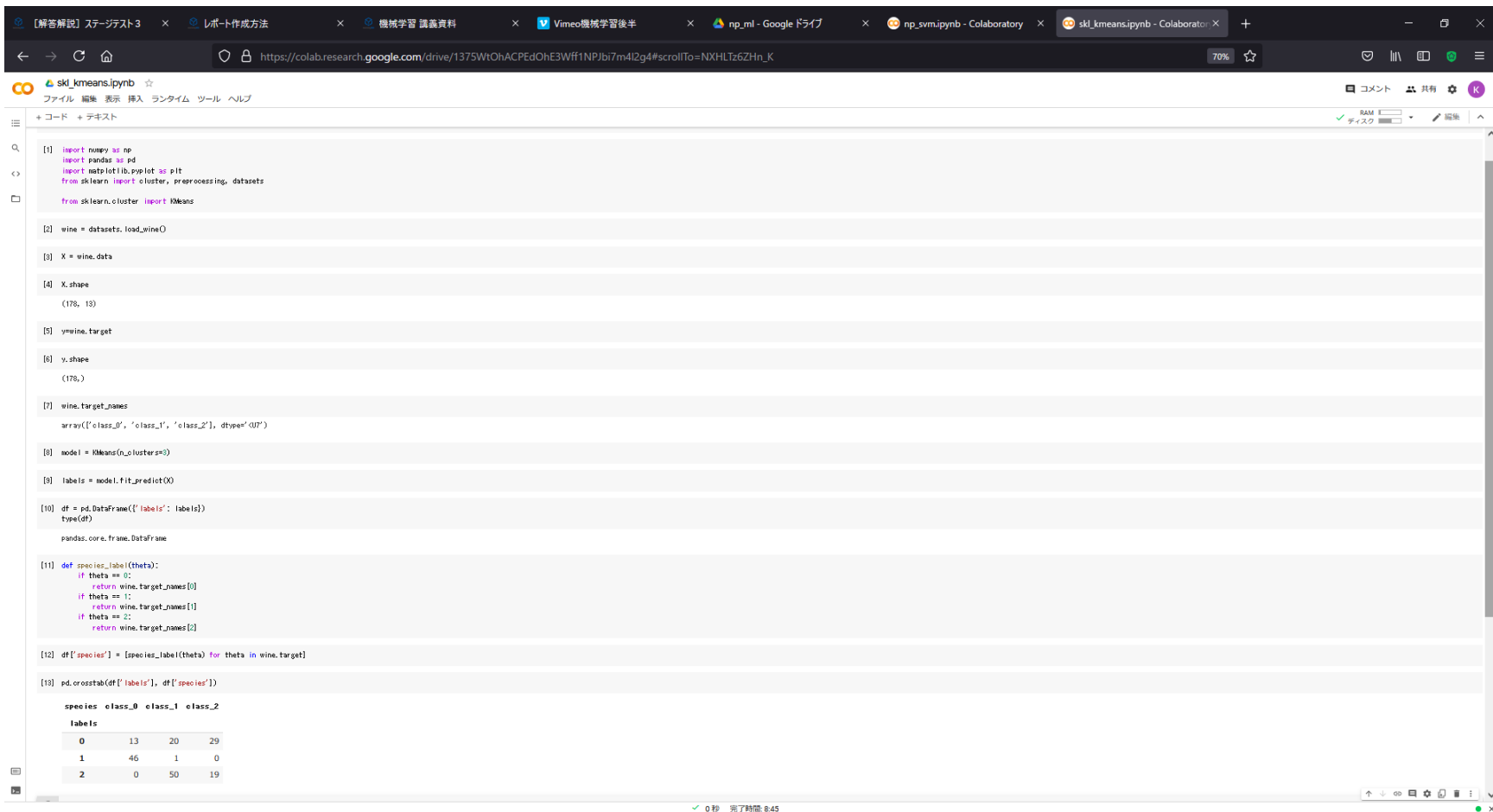
```
[7] from sklearn.neighbors import KNeighborsClassifier  
knc = KNeighborsClassifier(n_neighbors=5, neighbors_type='distance')  
knc.fit(X_train, y_train)  
plt_resut(X_train, y_train, knc.predict(xx))
```



0

0 秒 完了時間 8:47

アルゴリズム(k-means) 実装演習結果キャプチャ



The screenshot shows a Google Colab notebook titled 'skl_kmeans.ipynb'. The notebook contains Python code for loading the wine dataset, preprocessing it, and performing k-means clustering. The code is executed in a Jupyter environment, and the output of the final cell is displayed as a table.

```
[1] import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import cluster, preprocessing, datasets

from sklearn.cluster import KMeans

[2] wine = datasets.load_wine()

[3] X = wine.data

[4] X.shape

(178, 13)

[5] y = wine.target

[6] y.shape

(178,)

[7] wine.target_names

array(['class_0', 'class_1', 'class_2'], dtype=object)

[8] model = KMeans(n_clusters=3)

[9] labels = model.fit_predict(X)

[10] df = pd.DataFrame({'labels': labels})
type(df)

pandas.core.frame.DataFrame

[11] def species_label(theta):
    if theta == 0:
        return wine.target_names[0]
    if theta == 1:
        return wine.target_names[1]
    if theta == 2:
        return wine.target_names[2]

[12] df['species'] = [species_label(theta) for theta in wine.target]

[13] pd.crosstab(df['labels'], df['species'])
```

species	class_0	class_1	class_2
0	13	20	29
1	46	1	0
2	0	50	19

0 秒 完了時間 8:45

- SVM（サポートベクトルマシン）は1990年代に提案され、2000年代前半に急速に発展した手法
 - ・信号処理医療アプリケーションや自然言語処理、音声および画像認識などの多くの分類と回帰の問題に使用される教師あり学習アルゴリズム
 - ・目的は、あるクラスのデータ点を、別のクラスのデータ点から、可能な限り分離する超平面を見つけること。
境界と各クラスのデータ点との最短距離をマージンと呼び、マージンが最大になるよう境界を学習する。
 - ・マージン上にあるデータ点をサポートベクトルと呼ぶ。
- カーネルトリック：高次元に写像しながら実際には写像された空間での特徴の計算を避けて、カーネルの計算のみで最適な識別関数を構成するテクニック。
多項式カーネル、ガウスカーネル、シグモイドカーネルなどが使われている。

SVM 実装演習結果キャプチャ

【解き解説】 ステージテスト3 × 【レポート作成方法 × 機械学習 講義資料 × Vimeo機械学習後半 × np_ml - Google ドライブ × np_svm.py - Colaboratory × +

https://colab.research.google.com/drive/1YEK14wRtOH8Q-zCZnZ-Wfmxk2A8WO4T#scrollTo=LR-opvoJHgj

np_svm.py

サポートベクターマシン(SVM)

```
In [ ]: %matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

訓練データ生成①（線形分離可能）

```
[2]: def gen_data():
    x0 = np.random.normal(size=50).reshape(-1, 2) - 2.
    x1 = np.random.normal(size=50).reshape(-1, 2) + 2.
    X_train = np.concatenate([x0, x1])
    y0_train = np.zeros(50), np.ones(50)].astype(np.int)
    return X_train, y0_train

[3]: X_train, y0_train = gen_data()
plt.scatter(X_train[:, 0], X_train[:, 1], c=y0_train)
```

学習

特徴空間上で線形モデル $y(\mathbf{x}) = \mathbf{w}\phi(\mathbf{x}) + b$ を用い、その正負によって2値分類を行うことを考える。

サポートベクターマシンではマージンの最大化を行うが、それは結局以下の最適化問題を解くことと同じである。

ただし、訓練データを $X = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^T, t = [t_1, t_2, \dots, t_n]^T (t_i = \{-1, +1\})$ とする。

$$\min_{\mathbf{w}, b} \quad \frac{1}{2} \|\mathbf{w}\|^2$$
$$\text{subject to} \quad t_i (\mathbf{w}\phi(\mathbf{x}_i) + b) \geq 1 \quad (i = 1, 2, \dots, n)$$

ラグランジュ乗数法を使うと、上の最適化問題はラグランジュ乗数 $\alpha \geq 0$ を用いて、以下の目的関数を最小化する問題となる。

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^n \alpha_i t_i (\mathbf{w}\phi(\mathbf{x}_i) + b - 1) \quad \cdots (1)$$

目的関数が最小となるのは、 \mathbf{w}, b に関して偏微分した値が0となるときので、

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^n \alpha_i t_i \phi(\mathbf{x}_i) = 0$$
$$\frac{\partial L}{\partial b} = \sum_{i=1}^n \alpha_i t_i = 0$$

これを式(1) に代入することで、最適化問題は結局以下の目的関数の最大化となる。

SVM 実装演習結果キャプチャ

npsvm.ipynb

ファイル編集表示ランタイムツールヘルプすべての変更を保存しました

+コード+テキスト

コメントRAMディスク編集

学習

特徴空間上で線形なモデル $y(\boldsymbol{x}) = w\phi(\boldsymbol{x}) + b$ を用い、その正負によって2値分類を行うことを考える。
サポートベクターマシンではマージンの最大化を行うが、それは結局以下の最適化問題を解くことと同じである。

ただし、訓練データを $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n]^T, t = [t_1, t_2, \dots, t_n]^T (t_i \in \{-1, +1\})$ とする、

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

subject to $t_i(w\phi(\boldsymbol{x}_i) + b) \geq 1 \quad (i=1, 2, \dots, n)$

ラグランジュ乗数法を使うと、上の最適化問題はラグランジュ乗数 $a(\geq 0)$ を用いて、以下の目的関数を最小化する問題となる、

$$L(w, b, a) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n a_i t_i (w\phi(\boldsymbol{x}_i) + b - 1) \quad \cdots(1)$$

目的関数が最小となるのは、 w, b に関して偏微分した値が0となるときので、

$$\begin{aligned}\frac{\partial L}{\partial w} &= w - \sum_{i=1}^n a_i t_i \phi(\boldsymbol{x}_i) = 0 \\ \frac{\partial L}{\partial b} &= \sum_{i=1}^n a_i t_i = a^T t = 0\end{aligned}$$

これを式(1) に代入することで、最適化問題は結局以下の目的関数の最大化となる、

$$\begin{aligned}L(a) &= \sum_{i=1}^n a_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n a_i a_j t_i t_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) \\ &= a^T \mathbf{1} - \frac{1}{2} a^T H a\end{aligned}$$

ただし、行列 H の*i,j*成分は $H_{ij} = t_i t_j \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) = t_i t_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ である。また制約条件は、 $a^T t = 0 (\frac{1}{2} \|a^T t\|^2 = 0)$ である。

この最適化問題を最急降下法で解く、目的関数と制約条件を a で微分すると、

$$\begin{aligned}\frac{dL}{da} &= \mathbf{1} - H a \\ \frac{d}{da} (\frac{1}{2} \|a^T t\|^2) &= (a^T t) t\end{aligned}$$

なので、 a を以下の二式で更新する、

$$\begin{aligned}a &\leftarrow a + \eta_1 (1 - H a) \\ a &\leftarrow a - \eta_2 (a^T t) t\end{aligned}$$

```
[4] t = np.where(ys_train == 1.0, 1.0, -1.0)

n_samples = len(X_train)
# 線形カーネル
K = X_train.dot(X_train.T)

eta1 = 0.01
eta2 = 0.001
n_iter = 500

H = np.outer(t, t) * K

a = np.ones(n_samples)
for _ in range(n_iter):
    grad = 1 - H.dot(a)
    a += eta1 * grad
    a -= eta2 * a.dot(t) * t
    a = np.where(a > 0, a, 0)
```

予測

0秒完了時間8:48

SVM 実装演習結果キャプチャ

【解答解説】 ステージテスト3

レポート作成方法

機械学習 講義資料

Vimeo機械学習後半

np_ml - Googleドライブ

np_svm.ipynb - Colaboratory

+

← → ↺ 🏠 🔍 📄 📄 70% ☆ 🔄 📄 📄 📄 📄

np_svm.ipynb ☆

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

+ コード + テキスト

[4]

```
a = eta1 * grad
a = -> eta2 * a.dot(t) * t
a = np.where(a > 0, a, 0)
```

▼ 予測

新しいデータ点 \mathbf{x} に対しては、 $y(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x}) + b = \sum_{i=1}^n \alpha_i t_i k(\mathbf{x}, \mathbf{x}_i) + b$ の正負によって分類する。

ここで、最適化の結果得られた $\alpha_i (i = 1, 2, \dots, n)$ の中で $\alpha_i = 0$ に対応するデータ点は予測に影響を与えないので、 $\alpha_i > 0$ に対応するデータ点（サポートベクトル）のみ保持しておく。 b はサポートベクトルのインデックスの集合を S とすると、 $b = \frac{1}{2} \sum_{i \in S} (t_i - \sum_{j=1}^n \alpha_j t_j k(\mathbf{x}_i, \mathbf{x}_j))$ によって求める。

[5]

```
index = a > 1e-6
support_vectors = X_train[index]
support_vector_t = t[index]
support_vector_a = a[index]

term2 = K[index][:, index].dot(support_vector_a * support_vector_t)
b = (support_vector_t - term2).mean()
```

[6]

```
xx0, xx1 = np.meshgrid(np.linspace(-5, 5, 100), np.linspace(-5, 5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

X_test = xx
y_project = np.ones(len(X_test)) * b
for i in range(len(X_test)):
    for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
        y_project[i] += a * sv_t * sv.dot(X_test[i])
y_pred = np.sign(y_project)
```

[7]

```
# 訓練データを可視化
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)

# サポートベクトルを可視化
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
            s=100, facecolor='none', edgecolor='k')

# 傾斜を可視化
plt.contourf(xx0, xx1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))

# マージンと決定境界を可視化
plt.contour(xx0, xx1, y_project.reshape(100, 100), colors='k',
            levels=[-1, 0, 1], alpha=0.5, linestyle=['--', '-', '--'])

# マージンと決定境界を可視化
plt.quiver(0, 0, 0.1, 0.35, width=0.01, scale=1, color='pink')

Outplot: lib.quiver at 0x7fab608d24d0
```

0 秒 完了時間 8.48

🔍 ✕

SVM 実装演習結果キャプチャ

np_svm.pynb

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

+ コード + テキスト

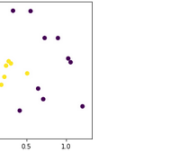
訓練データ生成②（線形分離不可能）

```
[8] factor = .2
     n_samples = 50
     linspace = np.linspace(0, 2 * np.pi, n_samples // 2 + 1)[:-1]
     outer_circ_x = np.cos(linspace)
     outer_circ_y = np.sin(linspace)
     inner_circ_x = outer_circ_x * factor
     inner_circ_y = outer_circ_y * factor

     X = np.vstack((np.append(outer_circ_x, inner_circ_x),
                     np.append(outer_circ_y, inner_circ_y))).T
     y = np.hstack([np.zeros(n_samples // 2, dtype=np.intp),
                    np.ones(n_samples // 2, dtype=np.intp)])
     X += np.random.normal(scale=0.15, size=X.shape)
     x_train = X
     y_train = y

plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
```

Outplotlib.colLECTIONS.PathCollection at 0x7fab57f54600



学習

元のデータ空間では線形分離は出来ないが、特徴空間上で線形分離することを考える。
今回はカーネルとしてRBFカーネル（ガウシアンカーネル）を利用する。

```
[10] def rbf(u, v):
      sigma = 0.8
      return np.exp(-0.5 * ((u - v)**2).sum() / sigma**2)

X_train = x_train
t = np.where(y_train == 1.0, 1.0, -1.0)

n_samples = len(X_train)
# RBFカーネル
K = np.zeros((n_samples, n_samples))
for i in range(n_samples):
    for j in range(n_samples):
        K[i, j] = rbf(X_train[i], X_train[j])

eta1 = 0.01
eta2 = 0.001
n_iter = 5000
```

0秒 完了時間 8:48

SVM 実装演習結果キャプチャ

np_svm.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

+ コード + テキスト

```
[10] a = eta1 * grad
      a -= eta2 * a.dot(t) * t
      a = np.where(a > 0, a, 0)
```

▼ 予測

```
[11] index = a > 1e-6
      support_vectors = X_train[index]
      support_vector_t = t[index]
      support_vector_a = a[index]

      term0 = K[index][:, index].dot(support_vector_a * support_vector_t)
      b = (support_vector_t - term0).mean()

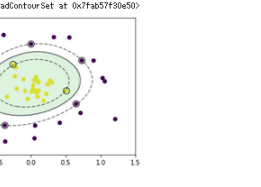
      x0, x01 = np.meshgrid(np.linspace(-1.5, 1.5, 100), np.linspace(-1.5, 1.5, 100))
      xx = np.array([x0, x01]).reshape(2, -1).T

      X_test = xx
      y_project = np.ones(len(X_test)) * b
      for i in range(len(X_test)):
          for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
              y_project[i] += a * sv_t * rbf(X_test[i], sv)
      y_pred = np.sign(y_project)
```

```
[12] # 訓練データを可視化
      plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)
      # サポートベクトルを可視化
      plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
                  s=100, facecolor='none', edgecolor='k')

      # 傾斜を可視化
      plt.contourf(x0, x01, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
      # マージンと決定境界を可視化
      plt.contour(x0, x01, y_project.reshape(100, 100), color='k',
                  levels=[-1, 0, 1], alpha=0.5, linestyle=['--', '-', '--'])
```

<matplotlib.figure.Figure at 0x7fab57f30e0>



▼ ソフトマージンSVM

▼ 訓練データ生成③ (重なりあり)

```
[14] x0 = np.random.normal(size=50).reshape(-1, 2) - 1.
      x1 = np.random.normal(size=50).reshape(-1, 2) + 1.
```


SVM 実装演習結果キャプチャ

np svm.ipynb

ファイル 編集 表示 挿入 ランタイム ツール ヘルプ

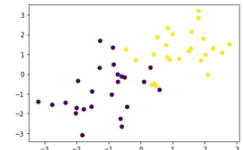
ソフトマージンSVM

訓練データ生成③ (重なりあり)

```
[14]: x0 = np.random.normal(size=50).reshape(-1, 2) - 1.
x1 = np.random.normal(size=50).reshape(-1, 2) + 1.
x_train = np.concatenate([x0, x1])
y_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)

plt.scatter(x_train[:, 0], x_train[:, 1], c=y_train)

import matplotlib.collections
PathCollection at 0x7fab57e37500
```



学習

分類不可能な場合は学習できないが、データ点がマージン内部に入ることや誤分類を許容することでその問題を回避する。

スラック変数 $\xi_i \geq 0$ を導入し、マージン内部に入ったり誤分類された点に対しては、 $\xi_i = |1 - t_i y(\mathbf{x}_i)|$ とし、これらを許容する代わりに対して、ペナルティを与えるように、最適化問題を以下のように修正する。

$$\min_{\mathbf{w}, b, \xi} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^n \xi_i$$

subject to $t_i(\mathbf{w}\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \quad (i = 1, 2, \dots, n)$

ただし、パラメータ C はマージンの大きさと誤差の許容度のトレードオフを決めるパラメータである。この最適化問題をラグランジュ乗数法などを用いると、結局最大化する目的関数はハードマージンSVMと同じとなる。

$$\tilde{L}(\mathbf{a}) = \sum_{i=1}^n a_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n a_i a_j t_i t_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

ただし、制約条件が $a_i \geq 0$ の代わりに $0 \leq a_i \leq C \ (i = 1, 2, \dots, n)$ となる。(ハードマージンSVMと同じ $\sum_{i=1}^n a_i t_i = 0$ の制約条件)

```
[16]: X_train = x_train
t = np.where(y_train == 1.0, 1.0, -1.0)

n_samples = len(X_train)
# 線形カーネル
K = X_train.dot(X_train.T)

C = 1
eta1 = 0.01
eta2 = 0.001
n_iter = 1000

H = np.outer(t, t) * K
```

コメント 共有

ディスク

RAM 100% ディスク 70%

0秒 完了時間: 8:48

SVM 実装演習結果キャプチャ

【解答解説】 ステージテスト3 × レポート作成方法 × 機械学習 講義資料 × Vimeo機械学習後半 × np_ml - Googleドライブ × np_svm.py - Colaboratory × +

https://colab.research.google.com/drive/1YEX14wRtOH8Q-zCZNz-WfnoK2A8W04T#scrollTo=Sr_29EtGHgcS

70%

np_svm.py

ファイル 編集 挿入 ランタイム ツール ヘルプ すべての変更を保存し直し

+ コード + テキスト

```
[16] c = 1
      eta1 = 0.01
      eta2 = 0.001
      n_iter = 1000

      H = np.outer(t, t) * K

      a = np.ones(n_samples)
      for _ in range(n_iter):
          grad = 1 - H.dot(a)
          a += eta1 * grad
          a -= eta2 * a.dot(t) * t
          a = np.clip(a, 0, 1)
```

予測

```
[17] index = a > 1e-8
      support_vectors = X_train[index]
      support_vector_t = t[index]
      support_vector_a = a[index]

      tern2 = K[index][:, index].dot(support_vector_a * support_vector_t)
      b = (support_vector_t - tern2).mean()
```

```
[18] x0, x1 = np.meshgrid(np.linspace(-4, 4, 100), np.linspace(-4, 4, 100))
      xx = np.array([x0, x1]).reshape(2, -1).T

      X_test = xx
      y_project = np.ones((len(X_test))) * b
      for i in range(len(X_test)):
          for a, sv_t, sv in zip(support_vector_a, support_vector_t, support_vectors):
              y_project[i] += a * sv_t * sv.dot(X_test[i])
      y_pred = np.sign(y_project)
```

● # 訓練データを可視化

```
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train)
# サポートベクトルを可視化
plt.scatter(support_vectors[:, 0], support_vectors[:, 1],
            s=100, facecolor='none', edgecolor='k')
# 傾斜を可視化
plt.contourf(x0, x1, y_pred.reshape(100, 100), alpha=0.2, levels=np.linspace(0, 1, 3))
# マージンと決定境界を可視化
plt.contour(x0, x1, y_project.reshape(100, 100), colors='k',
            levels=[-1, 0, 1], alpha=0.5, linestyle=['--', '-', '--'])
```

<matplotlib.figure.Figure at 0x7fab5e82850>



[19]

0秒 完了時間: 8:48