



python で データ解析

4. データ解析のカタログ

メニュー

- データの読み込み
- データフレームの要約
- 種々の統計手法
- 種々の回帰分析、ダミー変数、データの標準化、他
- Scikit-learn: 機械学習の各種手法
- モデルの評価
- その他

※ 本資料では、普通の python を python、Google Colaboratory を Colab と略記します 2

使用するデータ①: *iris*

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa
4.6	3.4	1.4	0.3	setosa
...

- フィッシャーが判別分析法を紹介するために利用したアヤメの品種分類 (Species: setosa, versicolor, virginica)に関するデータ
⇒ 以下の 4 変数を説明変数としてアヤメの種類を判別しようとした

- アヤメのがくの長さ (Sepal.Length)
- アヤメのがくの幅 (Sepal.Width)
- アヤメの花弁の長さ (Petal.Length)
- アヤメの花弁の幅 (Petal.Width)

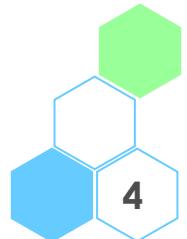


Graphic by (c)Tomo.Yun (<http://www.yunphoto.net>)

使用するデータ②: *ToothGrowth*

- モルモットに VC(ビタミンC) 又は OJ(オレンジジュース) を与えた時の歯の長さを調べる
 - `len`: 長さ(mm)
 - `supp`: サプリの種類(VC 又は OJ)
 - `dose`: 用量(0.5mg, 1.0mg, 2.0mg)

len	supp	dose
4.2	VC	0.5
11.5	VC	0.5
7.3	VC	0.5
:	:	:
27.3	OJ	2
29.4	OJ	2
23	OJ	2



使用するデータ③: DEP

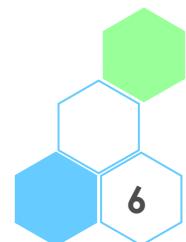
- うつ病を患っている患者さんに薬剤治療を行った後、QOLの点数を測定
- QOL(Quality of Life; 生活の質)の点数: 以下のアンケート票を使って患者さんに回答してもらい、各質問項目で回答した番号を合計したものを当該患者さんの点数とする

No	質問	当てはまらない (1点)	あまり当てはまらない (2点)	やや当てはまる (3点)	当てはまる (4点)
1	起床時に気分が良い		○		
2	朝食は美味しい				○
3	学校/会社に行きたい	○			
:	:	:	:	:	:



使用するデータ③: *DEP* の変数

- **GROUP**: 薬剤の種類(A, B, C)
- **QOL**: QOL の点数(数値) ⇒ 点数が大きい方が良い
- **EVENT**: 改善の有無(1: 改善あり, 2: 改善なし)
⇒ QOLの点数が 5 点以上である場合を「改善あり」とする
- **DAY**: 観察期間(数値, 単位は日)
- **PREDRUG**: 前治療薬の有無(YES: 他の治療薬の投与経験あり、
NO: 投与したことなし)
- **DURATION**: 罹病期間(数値, 単位は年)



データフレーム (DataFrame) の作成

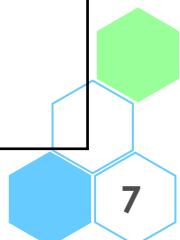
- ライブラリの呼び出し

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

- CSV ファイルから読み込み

```
tg    = pd.read_csv('./tg.csv')
dep   = pd.read_csv('./dep.csv')
iris  = pd.read_csv('./iris.csv', header=0,
                    names=['SL', 'SW', 'PL', 'PW', 'SP'])

iris.loc[iris['SP']=='setosa',      'Y'] = int(1)
iris.loc[iris['SP']=='versicolor', 'Y'] = int(2)
iris.loc[iris['SP']=='virginica',  'Y'] = int(0)
```



データフレーム (DataFrame) の作成

- Google Drive + Colab にて CSV ファイルを読み込む場合、



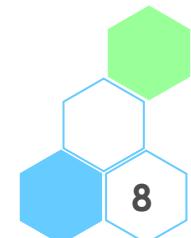
- ドライブのマウントを行う(下記を実行後、access code を入力して認証を行う手順が発生する)

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

- 前頁の方法で読み込み

```
tg    = pd.read_csv('./tg.csv')  
dep   = pd.read_csv('./dep.csv')  
iris = pd.read_csv('./iris.csv', header=0,  
                  names=['SL', 'SW', 'PL', 'PW', 'SP'])
```

[以下、前頁と同じ処理]



参考: Colab の接続に不具合があるとき

- 緊急手段としてベタ打ちデータからデータフレームを作成

```
import io

tmp = """len,supp,dose
4.2,VC,0.5
11.5,VC,0.5
.....
23,OJ,2
"""
tg = pd.read_csv(io.StringIO(tmp))

tmp = """SepalLength,SepalWidth,PetalLength,PetalWidth,Species
5.1,3.5,1.4,0.2,setosa
4.9,3,1.4,0.2,setosa
.....
5.9,3,5.1,1.8,virginica
"""
iris = pd.read_csv(io.StringIO(tmp), header=0,
                    names=['SL', 'SW', 'PL', 'PW', 'SP'])
iris.loc[iris['SP']=='setosa', 'Y'] = int(1)
iris.loc[iris['SP']=='versicolor', 'Y'] = int(2)
iris.loc[iris['SP']=='virginica', 'Y'] = int(0)
```

メニュー

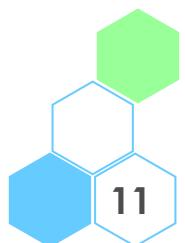
- データの読み込み
- データフレームの要約
- 種々の統計手法
- 種々の回帰分析、ダミー変数、データの標準化、他
- Scikit-learn: 機械学習の各種手法
- モデルの評価
- その他

※ 本資料では、普通の python を python、Google Colaboratory を Colab と略記します 10

データフレームの要約

関数	算出するもの
DataFrame.abs()	絶対値
DataFrame.corr(method={'pearson','kendall','spearman'})	相関係数
DataFrame.count()	非欠測の値の数
DataFrame.cov()	共分散
DataFrame.describe(percentiles=[.25, .5, .75])	要約統計量
DataFrame.max()	最大値
DataFrame.mean()	平均値
DataFrame.median()	中央値
DataFrame.min()	最小値
DataFrame.mode()	最頻値
DataFrame.quantile(q=0.5)	分位点
DataFrame.round(decimals=0)	丸め
DataFrame.sum()	合計値
DataFrame.std()	標準偏差
DataFrame.var()	不偏分散
DataFrame.nunique()	重複を除いた値の種類

※ 引数 axis=0 で「各列(各変数)について」要約を行う、ほとんどの関数で指定不要



データフレームの要約

```
pd.set_option('precision', 2) # 小数点2桁までの表示に

iris.iloc[:,0:4].mean()
iris.iloc[:,0:4].describe(percentiles=[.25, .5, .75])
iris.iloc[:,0:4].quantile(q=[0.05, 0.95])
```

各変数の平均値

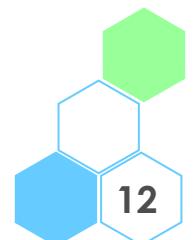
SL	5.84
SW	3.06
PL	3.76
PW	1.20
dtype: float64	

各変数の要約統計量(四分位値付き)

	SL	SW	PL	PW
count	150.00	150.00	150.00	150.00
mean	5.84	3.06	3.76	1.20
std	0.83	0.44	1.77	0.76
min	4.30	2.00	1.00	0.10
25%	5.10	2.80	1.60	0.30
50%	5.80	3.00	4.35	1.30
75%	6.40	3.30	5.10	1.80
max	7.90	4.40	6.90	2.50

各変数の 5%点、95%点

	SL	SW	PL	PW
0.05	4.60	2.34	1.3	0.2
0.95	7.25	3.80	6.1	2.3



データフレームの要約

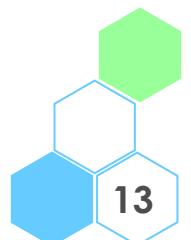
- groupby() で、ある変数のカテゴリ(supp)ごとに要約を行う

```
tg.groupby('supp').agg({'dose': 'nunique', 'len': ['mean', 'std']})
.reset_index()
```

	supp	dose	len	nunique	mean	std
0	OJ	3	20.66	6.61		
1	VC	3	16.96	8.27		

```
tg.iloc[:,0:2].groupby('supp').describe(percentiles=[.25, .5, .75])
```

	len							
	count	mean	std	min	25%	50%	75%	max
supp								
OJ	30.0	20.66	6.61	8.2	15.52	22.7	25.73	30.9
VC	30.0	16.96	8.27	4.2	11.20	16.5	23.10	33.9



データフレームの要約

- クロス集計を行う

```
tb = pd.crosstab(tg['supp'], tg['dose']) # 「行変数, 列変数」の順番
```

dose 0.5 1.0 2.0

supp

 OJ 10 10 10

 VC 10 10 10

```
tb = pd.crosstab(tg['supp'], tg['dose'],
                  margins=True, margins_name='Total') # 合計行・合計列を追加
```

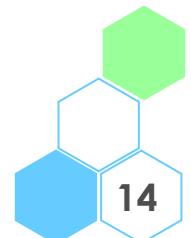
dose 0.5 1.0 2.0 Total

supp

 OJ 10 10 10 30

 VC 10 10 10 30

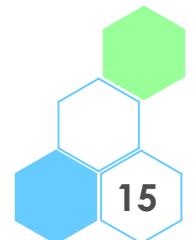
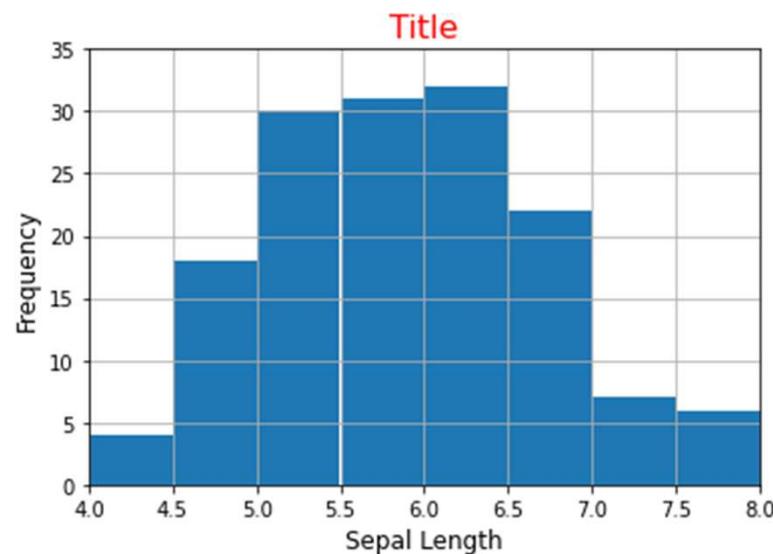
 Total 20 20 20 60



1つの連続データ: ヒストグラム

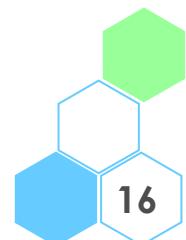
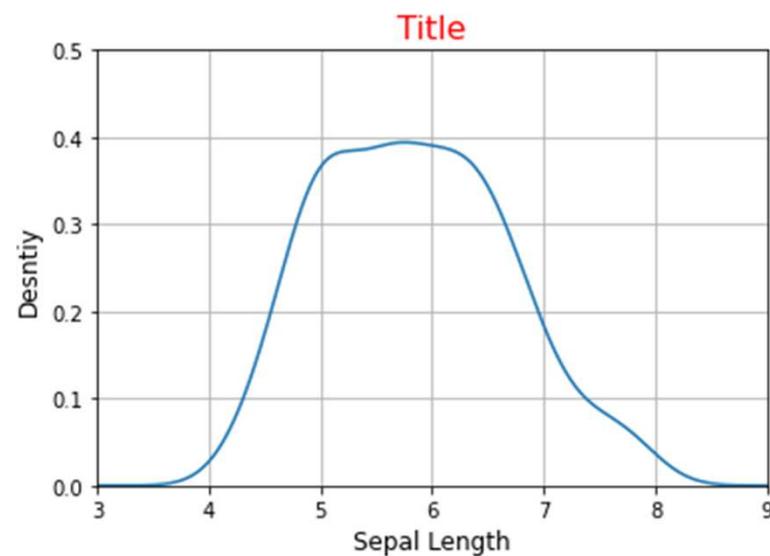
- 各区分 [4, 4.5) [4.5, 5) ... は左端を含み右端を含まず

```
plt.hist(iris.SL, bins=8, range=(4, 8))  
plt.title('Title', size=16, color='red') # タイトル  
plt.xlabel('Sepal Length', size=12) # x軸ラベル  
plt.ylabel('Frequency', size=12) # y軸ラベル  
plt.xlim(4, 8) # x軸の範囲  
plt.ylim(0, 35) # y軸の範囲  
plt.grid() # グリッド線  
plt.show()
```



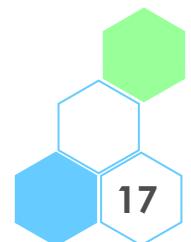
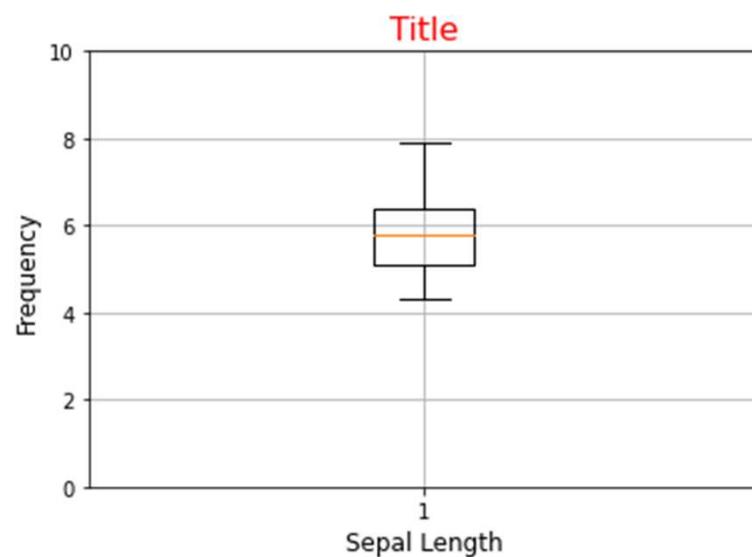
1つの連続データ: 密度推定

```
iris.SL.plot(kind='kde')
plt.title('Title', size=16, color='red') # タイトル
plt.xlabel('Sepal Length', size=12)       # x軸ラベル
plt.ylabel('Desntiy',      size=12)        # y軸ラベル
plt.xlim(3, 9)                          # x軸の範囲
plt.ylim(0, 0.5)                        # y軸の範囲
plt.grid()                             # グリッド線
plt.show()
```



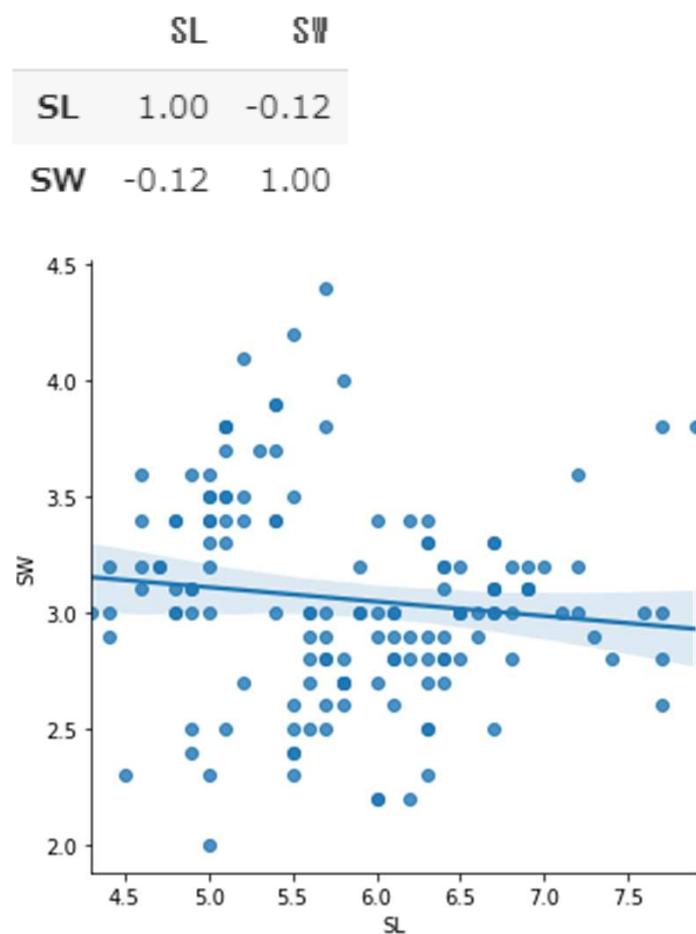
1つの連続データ: 箱ひげ図

```
plt.boxplot(iris.SL)
plt.title('Title', size=16, color='red') # タイトル
plt.xlabel('Sepal Length', size=12)      # x軸ラベル
plt.ylabel('Frequency',    size=12)       # y軸ラベル
plt.ylim(0, 10)                         # y軸の範囲
plt.grid()                             # グリッド線
plt.show()
```



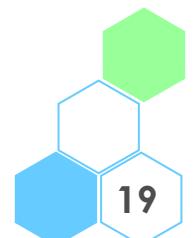
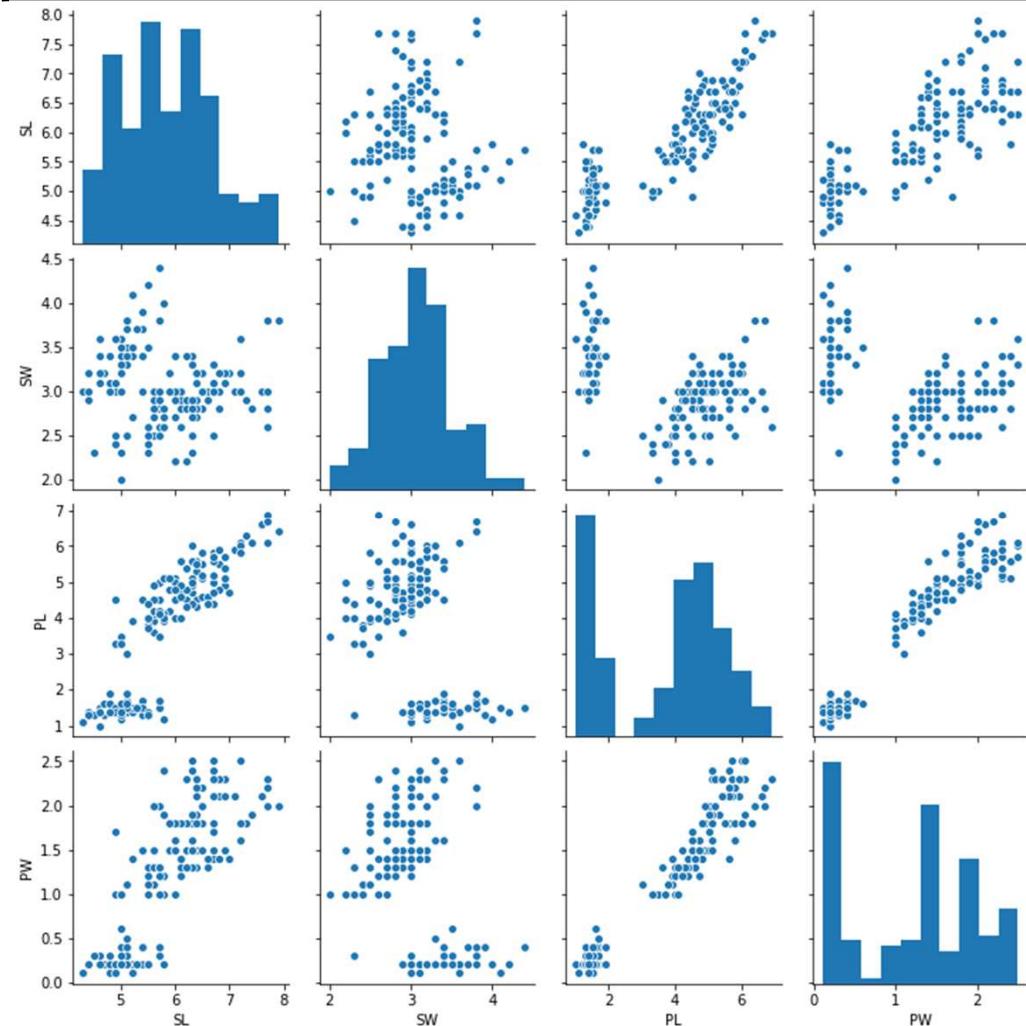
2つの連続データ: 散布図と相関係数

```
iris.iloc[:,0:2].corr()  
g = sns.lmplot(x="SL", y="SW", data=iris)  
plt.show(g)
```



2つの連続データ: 散布図と相関係数

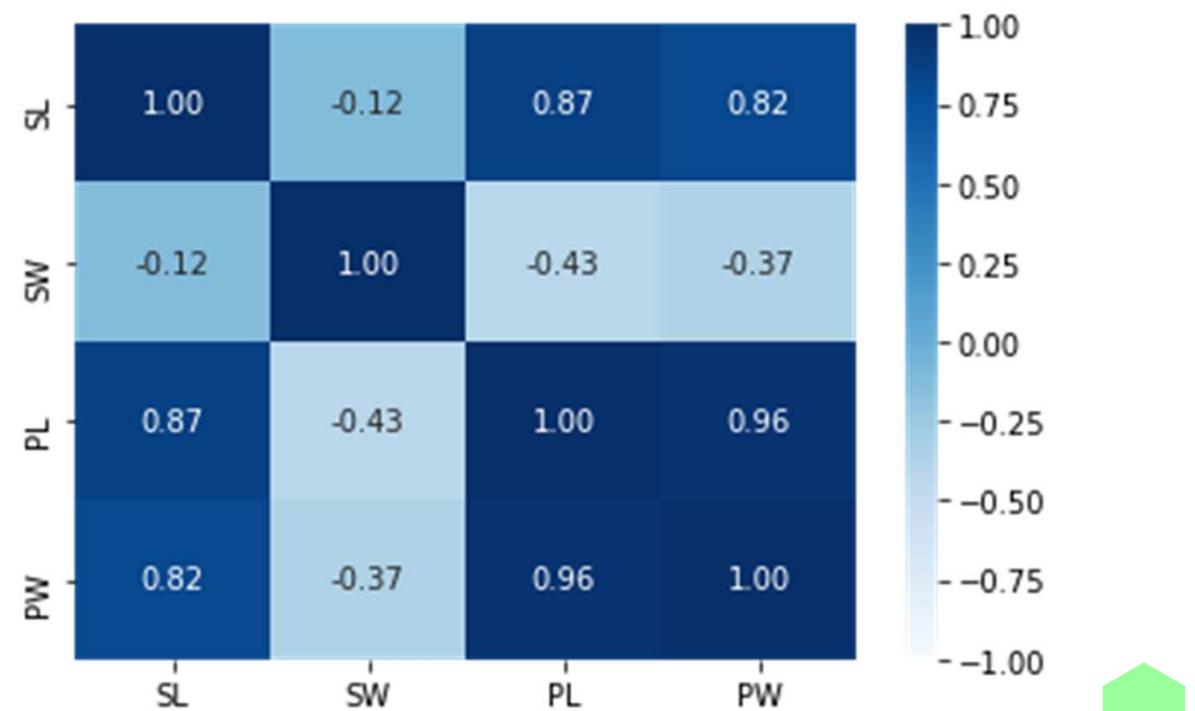
```
g = sns.pairplot(iris.iloc[:, 0:4])  
plt.show(g)
```



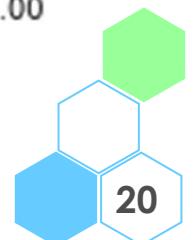
2つの連続データ: 散布図と相関係数

```
r = iris.iloc[:, 0:4].corr()          # cov()にすると共分散
print(r)                            # ピアソンの相関係数
sns.heatmap(r, annot=True, fmt='.2f', cmap='Blues',
             vmin=-1, vmax=1)
```

	SL	SW	PL	PW
SL	1.00	-0.12	0.87	0.82
SW	-0.12	1.00	-0.43	-0.37
PL	0.87	-0.43	1.00	0.96
PW	0.82	-0.37	0.96	1.00



※ cmap(colormap)について → <https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html>



メニュー

- データの読み込み
- データフレームの要約
- 種々の統計手法
- 種々の回帰分析、ダミー変数、データの標準化、他
- Scikit-learn: 機械学習の各種手法
- モデルの評価
- その他

※ 本資料では、普通の python を python、Google Colaboratory を Colab と略記します 21

確率分布(抜粋)

- 正規分布の各種計算
(デフォルトの引数は loc=0, scale=1)

```
from scipy import stats

dnorm = stats.norm.pdf(0)      # 密度
pnorm = stats.norm.cdf(1.96)   # 確率
qnorm = stats.norm.ppf(0.975)  # 分位点
```

- 正規分布に従う乱数

```
np.random.seed(seed=777) # シード設定
r = stats.norm.rvs(loc=0, scale=1,
                   size=5, random_state=777)
```

- リファレンス: Statistical functions (scipy.stats)

<https://docs.scipy.org/doc/scipy/reference/stats.html>

関数	分布
beta()	Beta 分布
chi2()	カイ二乗分布
expon()	指数分布
f()	F 分布
norm()	正規分布
t()	t 分布
uniform()	一様分布(連續)
bernoulli()	Bernoulli 分布
binom()	二項分布
geom()	幾何分布
hypergeom()	超幾何分布
nbinom()	負の二項分布
poisson()	Poisson 分布
randint()	一様分布(離散)

確率分布のメソッド: 正規分布 *norm()* の場合

関数	機能
rvs(loc=0, scale=1, size=1, random_state=None)	Random variates.
pdf(x, loc=0, scale=1)	Probability density function.
logpdf(x, loc=0, scale=1)	Log of the probability density function.
cdf(x, loc=0, scale=1)	Cumulative distribution function.
logcdf(x, loc=0, scale=1)	Log of the cumulative distribution function.
sf(x, loc=0, scale=1)	Survival function.
logsf(x, loc=0, scale=1)	Log of the survival function.
ppf(q, loc=0, scale=1)	Percent point function (inverse of cdf – percentiles).
isf(q, loc=0, scale=1)	Inverse survival function (inverse of sf).
moment(n, loc=0, scale=1)	Non-central moment of order n
stats(loc=0, scale=1, moments='mv')	Mean('m'), variance('v'), skew('s'), and/or kurtosis('k').
entropy(loc=0, scale=1)	(Differential) entropy of the RV.
fit(data)	Parameter estimates for generic data.
expect(func, args=(), loc=0, scale=1, lb=None, ub=None, ...)	Expected value of a function (of one argument).
median(loc=0, scale=1)	Median of the distribution.
mean(loc=0, scale=1)	Mean of the distribution.
var(loc=0, scale=1)	Variance of the distribution.
std(loc=0, scale=1)	Standard deviation of the distribution.
interval(alpha, loc=0, scale=1)	Range that contains alpha percent of the distribution



検定手法(抜粋)

関数	手法
f_oneway(*args[, axis])	One-way ANOVA.
pearsonr(x, y)	Pearson's r and p-value for testing non-correlation.
ttest_1samp(a, popmean[, axis, nan_policy])	T-test for the mean of ONE group of scores.
ttest_ind(a, b[, axis, equal_var, nan_policy])	T-test for the means of two independent samples of scores.
ttest_ind_from_stats(mean1, std1, nobs1, ...)	T-test for means of two independent samples from descriptive statistics.
ttest_rel(a, b[, axis, nan_policy])	T-test on TWO RELATED samples of scores, a and b.
chisquare(f_obs[, f_exp, ddof, axis])	One-way chi-square test.
mannwhitneyu(x, y[, use_continuity, alternative])	Mann-Whitney rank test on samples x and y.
ranksums(x, y)	Wilcoxon rank-sum statistic for two samples.
wilcoxon(x[, y, zero_method, correction, ...])	Wilcoxon signed-rank test.
kruskal(*args, **kwargs)	Kruskal-Wallis H-test for independent samples.
friedmanchisquare(*args)	Friedman test for repeated measurements.
bartlett(*args)	Bartlett's test for equal variances.
binom_test(x[, n, p, alternative])	Binomial test that the probability of success is p.
median_test(*args, **kwds)	Mood's median test.
chi2_contingency(observed[, correction, lambda_])	Chi-square test of independence of variables in a contingency table.
fisher_exact(table[, alternative])	Fisher exact test on a 2x2 contingency table.

検定例

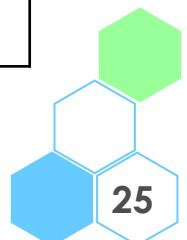
- 2 標本 t 検定(等分散を仮定)

```
tg1 = tg.query('supp == "OJ"').len  
tg2 = tg.query('supp == "VC"').len  
stats.ttest_ind(tg1, tg2, equal_var=True)  
  
(statistic=1.91526826869527, pvalue=0.06039337122412849)
```

- χ^2 検定(連續修正なし)
→ 返り値は順に「統計量, p 値, 自由度, 分割表」

```
tb = pd.crosstab(tg['supp'], tg['dose'])  
stats.chi2_contingency(tb, correction=False)  
  
(0.0, 1.0, 2, array([[10., 10., 10.], [10., 10., 10.]]))
```

- リファレンス: Statistical functions (scipy.stats)
<https://docs.scipy.org/doc/scipy/reference/stats.html>



メニュー

- データの読み込み
- データフレームの要約
- 種々の統計手法
- 種々の回帰分析、ダミー変数、データの標準化、他
- Scikit-learn: 機械学習の各種手法
- モデルの評価
- その他

※ 本資料では、普通の python を python、Google Colaboratory を Colab と略記します 26

単回帰分析: $SW = INT + SL + error$

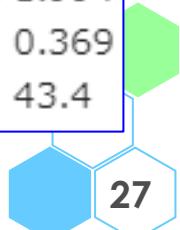
- statsmodels の OLS にて単回帰分析を行う
⇒ 事前に切片項（変数 INT : 1 のみの列）を追加しておく

```
import statsmodels.api as sm

iris['INT'] = 1          # np.ones(iris.shape[0])
ols      = sm.OLS(iris.SW, iris[['INT', 'SL']])
res      = ols.fit()
summary = res.summary()
summary.tables[0]; summary.tables[1]; summary.tables[2]
```

OLS Regression Results	
Dep. Variable:	SW
Model:	OLS
Method:	Least Squares
Date:	Fri, 24 Jul 2020
Time:	08:09:28
No. Observations:	150
Df Residuals:	148
Df Model:	1
Covariance Type:	nonrobust
R-squared:	0.014
Adj. R-squared:	0.007
F-statistic:	2.074
Prob (F-statistic):	0.152
Log-Likelihood:	-86.732
AIC:	177.5
BIC:	183.5

	coef	std err	t	P> t	[0.025 0.975]
INT	3.4189	0.254	13.484	0.000	2.918 3.920
SL	-0.0619	0.043	-1.440	0.152	-0.147 0.023
Omnibus: 2.474 Durbin-Watson: 1.263					
Prob(Omnibus):	0.290	Jarque-Bera (JB):	1.994	Skew:	0.243 Prob(JB): 0.369
Kurtosis:	3.288	Cond. No.	43.4		



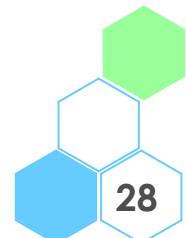
重回帰分析: $SW = INT + SL + PL + PW + error$

- statsmodels の OLS にて重回帰分析を行う
 ⇒ 事前に切片項（変数 INT : 1 のみの列）を追加しておく
- 以下では回帰係数の推定結果 summary.tables[1] のみ表示

```
import statsmodels.api as sm

iris['INT'] = 1
ols      = sm.OLS(iris.SW, iris[['INT', 'SL', 'PL', 'PW']])
res      = ols.fit()
summary = res.summary()
summary.tables[1]
```

	coef	std err	t	P> t [0.025 0.975]
INT	1.0431	0.271	3.855	0.000 0.508 1.578
SL	0.6071	0.062	9.765	0.000 0.484 0.730
PL	-0.5860	0.062	-9.431	0.000 -0.709 -0.463
PW	0.5580	0.123	4.553	0.000 0.316 0.800



種々の回帰分析①

- sklearn.linear_model の関数は事前に切片項を作成しておく必要はない
- 手法(普通の回帰分析、リッジ回帰、ラツソ回帰、Elastic Net)の詳細:
https://scikit-learn.org/stable/modules/linear_model.html

```

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet

x_list = ["SL", "PL", "PW"]          # 説明変数名
y_list = ["SW"]                      # 目的変数名
data_x = iris[x_list]                # 説明変数
data_y = iris[y_list]                # 目的変数
lr = LinearRegression()              # 普通の回帰分析
# lr = Ridge(alpha=0.01)             # リッジ回帰
# lr = Lasso(alpha=0.01)             # ラツソ回帰
# lr = ElasticNet(alpha=0.01) # Elastic Net

lr.fit(data_x, data_y)
print(lr.coef_)                     # 各説明変数の係数
print(lr.intercept_)                # 切片項の係数

```

リッジ回帰: L2正則化項、重みが完全に0にならず説明変数が複雑になる傾向

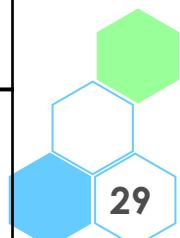
ラツソ回帰:L1正則化項、影響が小さい変数の重みは0(モデルから除外)

Elastic Net: L1とL2の両方の項含む「l1_ratio=0.5」等で配分を決める

```

[[ 0.60706601 -0.58603225  0.55803034]]
[1.04308908]

```



種々の回帰分析②

- 手法の詳細:

SGD Reg. → <https://scikit-learn.org/stable/modules/sgd.html#regression>

SVR → https://scikit-learn.org/stable/auto_examples/svm/plot_svm_regression.html

```
from sklearn.linear_model import SGDRegressor
from sklearn.svm import SVR
from sklearn.ensemble import BaggingRegressor

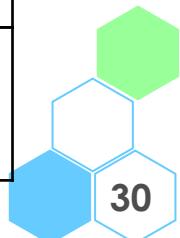
x_list = ["SL", "PL", "PW"]          # 説明変数名
y_list = ["SW"]                      # 目的変数名
data_x = iris[x_list]                # 説明変数
data_y = iris[y_list]                # 目的変数

lr = SGDRegressor(max_iter=1000, tol=1e-3)    # SGD Reg.
# lr = SVR(kernel='linear', C=100, gamma='auto') # SVR
```

```
lr.fit(data_x, data_y.values.ravel())
print(lr.coef_)
print(lr.intercept_)
```

[0.72899196 -0.34191137 -0.10420014]
[0.22757961]

※ .values.ravel() : データを 1 列だけ抽出すると多次元リストになることがあるので、1 次元のリストに変換する



種々の回帰分析③

- 手法の詳細:

SVR → https://scikit-learn.org/stable/auto_examples/svm/plot_svm_regression.html

Bagging Reg. → <http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingRegressor.html>

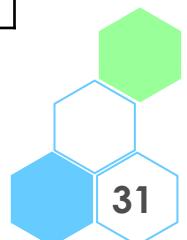
```
from sklearn.svm import SVR
from sklearn.ensemble import BaggingRegressor

x_list = ["SL", "PL", "PW"]          # 説明変数名
y_list = ["SW"]                      # 目的変数名
data_x = iris[x_list]                # 説明変数
data_y = iris[y_list]                # 目的変数

lr = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1) # SVR
# lr = BaggingRegressor(base_estimator=SVR(), n_estimators=10,
# random_state=777)           # Bagging Reg.

# 非線形モデルなので予測値を算出
pred_y = lr.fit(data_x, data_y.values.ravel()).predict(data_x)
```

略



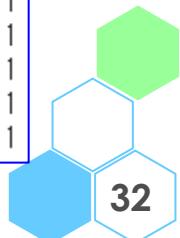
ダミー変数の作成

- pandas の関数 `get_dummies()` でカテゴリ変数に関するダミー変数が作成出来る
 - `drop_first=True` で最初のカテゴリに関する列を削除（通常は指定）

```
x_list = ["SL", "SW", "PL", "PW"]
d_list = ["SP"]
iris_d = pd.concat([iris[x_list], pd.get_dummies(iris[d_list]),
                    drop_first=True)], axis=1)
iris_d = pd.concat([iris[x_list], pd.get_dummies(iris[d_list])],
                    axis=1)
print(iris_d)
```

	SL	SW	PL	PW	SP_versicolor	SP_virginica
0	5.1	3.5	1.4	0.2	0	0
1	4.9	3.0	1.4	0.2	0	0
2	4.7	3.2	1.3	0.2	0	0
3	4.6	3.1	1.5	0.2	0	0
4	5.0	3.6	1.4	0.2	0	0
..
145	6.7	3.0	5.2	2.3	0	1
146	6.3	2.5	5.0	1.9	0	1
147	6.5	3.0	5.2	2.0	0	1
148	6.2	3.4	5.4	2.3	0	1
149	5.9	3.0	5.1	1.8	0	1

	SL	SW	PL	PW	SP_setosa	SP_versicolor	SP_virginica
0	5.1	3.5	1.4	0.2	1	0	0
1	4.9	3.0	1.4	0.2	1	0	0
2	4.7	3.2	1.3	0.2	1	0	0
3	4.6	3.1	1.5	0.2	1	0	0
4	5.0	3.6	1.4	0.2	1	0	0
..
145	6.7	3.0	5.2	2.3	0	0	1
146	6.3	2.5	5.0	1.9	0	0	1
147	6.5	3.0	5.2	2.0	0	0	1
148	6.2	3.4	5.4	2.3	0	0	1
149	5.9	3.0	5.1	1.8	0	0	1



ダミー変数の使用例: ロジスティック回帰分析

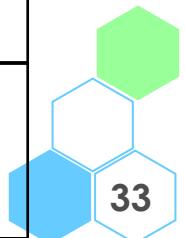
- $\text{logit}(\text{setosaの割合}) = \text{INT} + \text{SL} + \text{SW} + \text{PL} + \text{PW} + \text{error}$

```
from sklearn.linear_model import LogisticRegression

x_list = ["SL", "SW", "PL", "PW"]
d_list = ["SP"]
iris_d = pd.concat([iris[x_list],
                    pd.get_dummies(iris[d_list])], axis=1)

y_list = ["SP_setosa"]      # setosa:1、それ以外:0
data_x = iris_d[x_list]
data_y = iris_d[y_list]
lr = LogisticRegression()
lr.fit(data_x, data_y.values.ravel()) # 赤字：型変換
print(lr.coef_)
print(lr.intercept_)
```

```
[[ -0.44501376   0.89999242  -2.32353827  -0.97345836]]
[ 6.69040651]
```



データの標準化

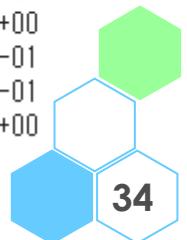
- sklearn.preprocessing の StandardScaler() と fit_transform()
にて、データを「平均0、分散1」のデータに標準化する

```
x_list      = ["SL", "SW", "PL", "PW"]
data_x      = iris[x_list]

from sklearn.preprocessing import StandardScaler
scaler      = StandardScaler()      # データを標準化
data_std    = scaler.fit_transform(iris[x_list])

print(data_x.describe())
print(pd.DataFrame(data_std).describe())
```

	SL	SW	PL	PW		0	1	2	3
count	150.000000	150.000000	150.000000	150.000000	count	1.500000e+02	1.500000e+02	1.500000e+02	1.500000e+02
mean	5.843333	3.057333	3.758000	1.199333	mean	-2.775558e-16	-9.695948e-16	-8.652338e-16	-4.662937e-16
std	0.828066	0.435866	1.765298	0.762238	std	1.003350e+00	1.003350e+00	1.003350e+00	1.003350e+00
min	4.300000	2.000000	1.000000	0.100000	min	-1.870024e+00	-2.433947e+00	-1.567576e+00	-1.447076e+00
25%	5.100000	2.800000	1.600000	0.300000	25%	-9.006812e-01	-5.923730e-01	-1.226552e+00	-1.183812e+00
50%	5.800000	3.000000	4.350000	1.300000	50%	-5.250608e-02	-1.319795e-01	3.364776e-01	1.325097e-01
75%	6.400000	3.300000	5.100000	1.800000	75%	6.745011e-01	5.586108e-01	7.627583e-01	7.906707e-01
max	7.900000	4.400000	6.900000	2.500000	max	2.492019e+00	3.090775e+00	1.785832e+00	1.712096e+00



例: PCA(主成分分析) + データ標準化なし

- 手法の詳細:

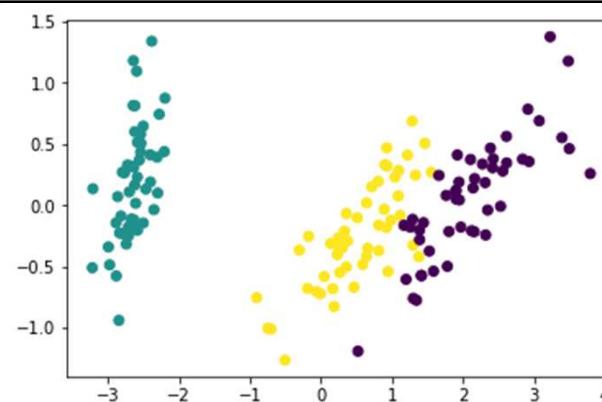
<https://scikit-learn.org/stable/modules/decomposition.html#principal-component-analysis-pca>
https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_lda.html

```
from sklearn.decomposition import PCA

x_list      = ["SL", "SW", "PL", "PW"]

data_x      = iris[x_list]
pca         = PCA(n_components=2)          # 2つの主成分に縮約
data_pca    = pca.fit_transform(data_x)
print(pca.explained_variance_ratio_)      # 寄与率
print(sum(pca.explained_variance_ratio_))
plt.scatter(data_pca[:, 0], data_pca[:, 1], c=iris["Y"])
plt.show()
```

[0.92461872 0.05306648]
0.977685206318795



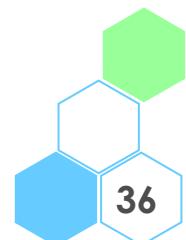
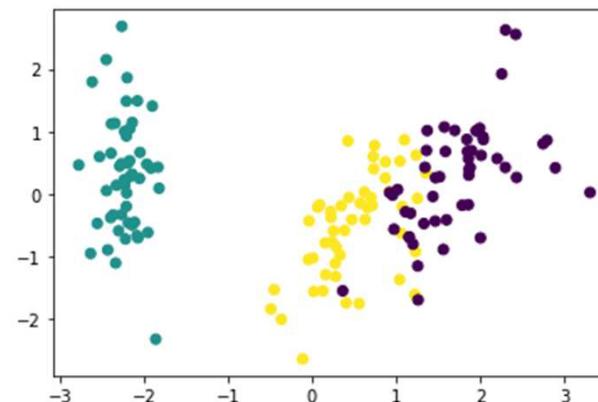
例: PCA(主成分分析) + データ標準化あり

- 手法の詳細:

<https://scikit-learn.org/stable/modules/decomposition.html#principal-component-analysis-pca>
https://scikit-learn.org/stable/auto_examples/decomposition/plot_pca_vs_lda.html

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
x_list = ["SL", "SW", "PL", "PW"]
scaler = StandardScaler() # データを標準化
data_std = scaler.fit_transform(iris[x_list])
pca = PCA(n_components=2) # 2つの主成分に縮約
data_pca = pca.fit_transform(data_std)
print(pca.explained_variance_ratio_) # 寄与率
print(sum(pca.explained_variance_ratio_))
plt.scatter(data_pca[:, 0], data_pca[:, 1], c=iris["Y"])
plt.show()
```

[0.72962445 0.22850762]
0.9581320720000164



参考: 次元圧縮の手法

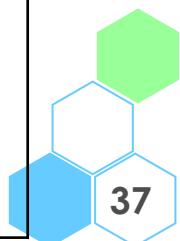
- 手法の詳細:

PCA → <https://scikit-learn.org/stable/modules/decomposition.html#principal-component-analysis-pca>
 Kernel PCA → <https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.KernelPCA.html>
 Isomap → <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.Isomap.html>
 Spectral Embedding → <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.SpectralEmbedding.html>
 LLE → <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.LocallyLinearEmbedding.html>

```
from sklearn.decomposition import PCA
from sklearn.decomposition import KernelPCA
from sklearn.manifold import Isomap, SpectralEmbedding,
LocallyLinearEmbedding

x_list      = ["SL", "SW", "PL", "PW"]
data_x      = iris[x_list]
dr          = PCA(n_components=2)                      # PCA
data_dr     = dr.fit_transform(data_x)
dr          = KernelPCA(n_components=2, kernel='rbf')   # Kernel PCA
data_dr    = dr.fit_transform(data_x)
dr          = Isomap(n_components=2)                    # Isomap
data_dr   = dr.fit_transform(data_x)
dr          = SpectralEmbedding(n_components=2)        # Spectral Embedding
data_dr  = dr.fit_transform(data_x)
dr          = LocallyLinearEmbedding(n_components=2)   # LLE
data_dr = dr.fit_transform(data_x)

plt.scatter(data_dr[:, 0], data_dr[:, 1], c=iris["Y"])
plt.show()
```



因果推論：傾向スコア、IPW の算出

- 手法の詳細: Hernan MA, Robins JM (2020) "Causal Inference: What If. Boca Raton", Chapman & Hall/CRC.
<https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/>

```

import statsmodels.api as sm

dep = dep.query('GROUP != "C"')
dep['ID'] = list(dep.index)
dep['GROUP_N'] = dep['GROUP'].apply(lambda x : 1 if x=='A' else 0)
dep['PREDR_N'] = dep['PREDRUG'].apply(lambda x : 1 if x=='YES' else 0)
dep['INT'] = 1

def ps_ipw(x):
    if x.GROUP_N==0:
        return 1/(1-x.ps)
    else:
        return 1/x.ps

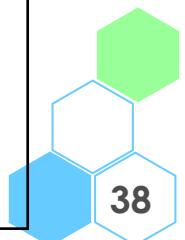
def ps_sipw(x, fa):
    if x.GROUP_N==0:
        return (1-fa)/(1-x.ps)
    else:
        return fa/x.ps

# 傾向スコア
dep['ps'] = sm.Logit(dep.GROUP_N, dep[['INT', 'PREDR_N',
                                         'DURATION']]).fit().predict()

# IPW(Inverse Probability Weighting)
dep['ipw'] = dep.apply(lambda x: ps_ipw(x), axis=1)

# Stabilized IPW
fa = dep.query('GROUP_N==1').GROUP_N.count() /
dep.GROUP_N.count()
dep['sipw'] = dep.apply(lambda x: ps_sipw(x, fa), axis=1)

```



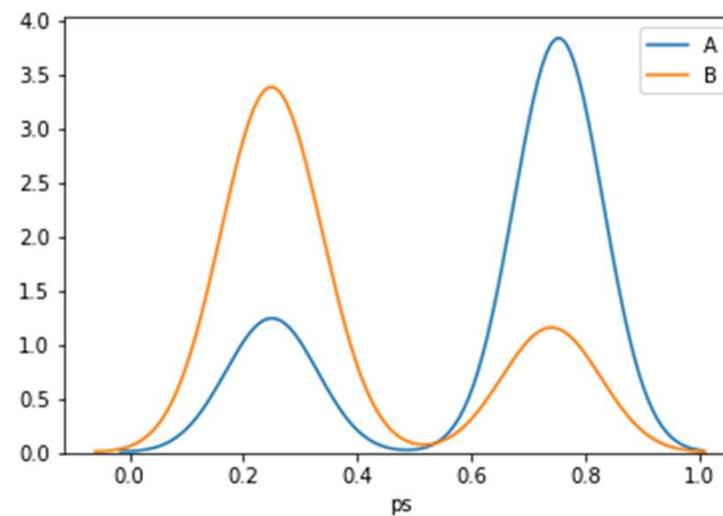
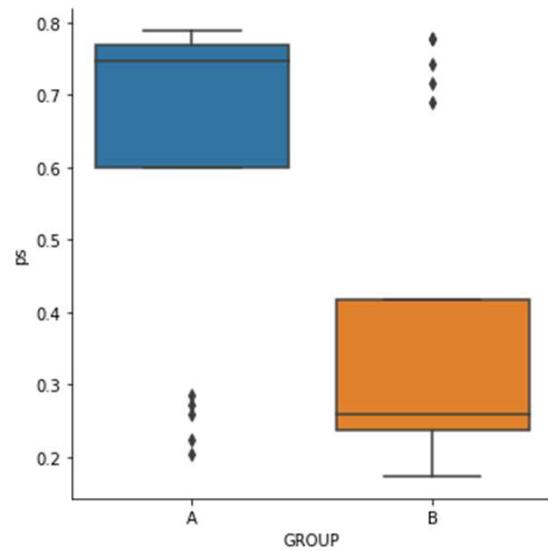
因果推論：傾向スコアの分布確認

- 手法の詳細: Hernan MA, Robins JM (2020) "Causal Inference: What If. Boca Raton", Chapman & Hall/CRC.
<https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/>

```

g = sns.catplot(x="GROUP", y="ps", kind="box", data=dep)
plt.show(g)      # 箱ひげ図

ps_A = dep.query('GROUP == "A"').ps
ps_B = dep.query('GROUP == "B"').ps
g = sns.distplot(ps_A, hist=False, label='A')
g = sns.distplot(ps_B, hist=False, label='B')
plt.show(g)      # 密度曲線
    
```



因果推論: IPW で調整解析

- 手法の詳細: Hernan MA, Robins JM (2020) "Causal Inference: What If. Boca Raton", Chapman & Hall/CRC.
<https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/>

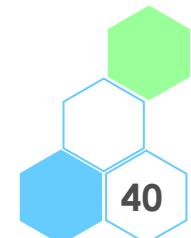
```
res_ols = sm.OLS(dep.QOL, dep[['INT', 'GROUP_N']]).fit()
print( res_ols.summary().tables[1] )      # 無調整の回帰分析
res_ipw = sm.GEE(dep.QOL, dep[['INT', 'GROUP_N']], groups=dep.ID,
weights=dep.ipw).fit()
print( res_ipw.summary().tables[1] )      # IPWで調整した重みづけ回帰
```

無調整の回帰分析

	coef	std err	t	P> t	[0.025 0.975]
INT	4.0000	0.862	4.639	0.000	2.255 5.745
GROUP_N	2.5000	1.219	2.050	0.047	0.032 4.968

IPWで調整した重みづけ回帰

	coef	std err	z	P> z	[0.025 0.975]
INT	5.4929	1.188	4.622	0.000	3.164 7.822
GROUP_N	-0.0576	1.437	-0.040	0.968	-2.875 2.760



因果推論: 標準化(Standardization)

- 手法の詳細: Hernan MA, Robins JM (2020) "Causal Inference: What If. Boca Raton", Chapman & Hall/CRC.
<https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/>

```

import statsmodels.api as sm

df          = dep[['QOL', 'GROUP_N', 'PREDR_N']] # 交絡因子:PREDR_N
df['GxP']   = df.GROUP_N * df.PREDR_N
df['zero']  = 0
df['one']   = 1

# 本では3ブロックに分けていたが、ここでは最初のブロックのみでモデル構築、予測時に残りのブロックを考慮
ols = sm.OLS(df.QOL, df[['one', 'GROUP_N', 'PREDR_N', 'GxP']])
res = ols.fit()
print(res.summary().tables[1])

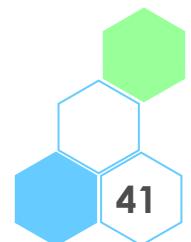
A_pred = res.predict(df[['one', 'one', 'PREDR_N', 'PREDR_N']])
print('standardized mean: {:.2f}'.format(A_pred.mean())) # 薬剤 A の標準化平均

B_pred = res.predict(df[['one', 'zero', 'PREDR_N', 'zero']])
print('standardized mean: {:.2f}'.format(B_pred.mean())) # 薬剤 B の標準化平均

```

	coef	std err	t	P> t	[0.025	0.975]
<hr/>						
one	8.8000	1.420	6.198	0.000	5.921	11.679
GROUP_N	-1.4000	1.639	-0.854	0.399	-4.725	1.925
PREDR_N	-6.4000	1.639	-3.904	0.000	-9.725	-3.075
GxP	2.8000	2.318	1.208	0.235	-1.902	7.502
<hr/>						

standardized mean: 5.60 ← 薬剤 A の標準化平均
 standardized mean: 5.60 ← 薬剤 B の標準化平均



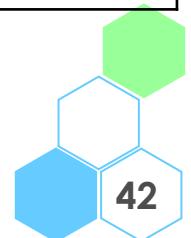
因果推論：標準化された平均値の差の95% CI

- 手法の詳細: Hernan MA, Robins JM (2020) "Causal Inference: What If. Boca Raton", Chapman & Hall/CRC.
<https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/>

```
# Bootstrap for 95% CI
est_diff      = A_pred.mean() - B_pred.mean()
boot_samples = []
for _ in range(1000):
    sample = df.sample(n=df.shape[0], replace=True)
    y      = sample.QOL
    X      = sample[['one', 'GROUP_N', 'PREDR_N', 'GxP']]
    A      = sample[['one', 'one',      'PREDR_N', 'PREDR_N']]
    B      = sample[['one', 'zero',     'PREDR_N', 'zero']]
    result = sm.OLS(y, X).fit()
    A_pred = result.predict(A)
    B_pred = result.predict(B)
    boot_samples.append(A_pred.mean() - B_pred.mean())

std = np.std(boot_samples)
lo  = est_diff - 1.96 * std
hi  = est_diff + 1.96 * std
print('                  estimate  95% C.I.')
print('causal effect  {:>6.1f}  ({:>0.1f}, {:>0.1f})'.format(est_diff, lo, hi))
```

	estimate	95% C.I.
causal effect	0.0	(-2.4, 2.4)



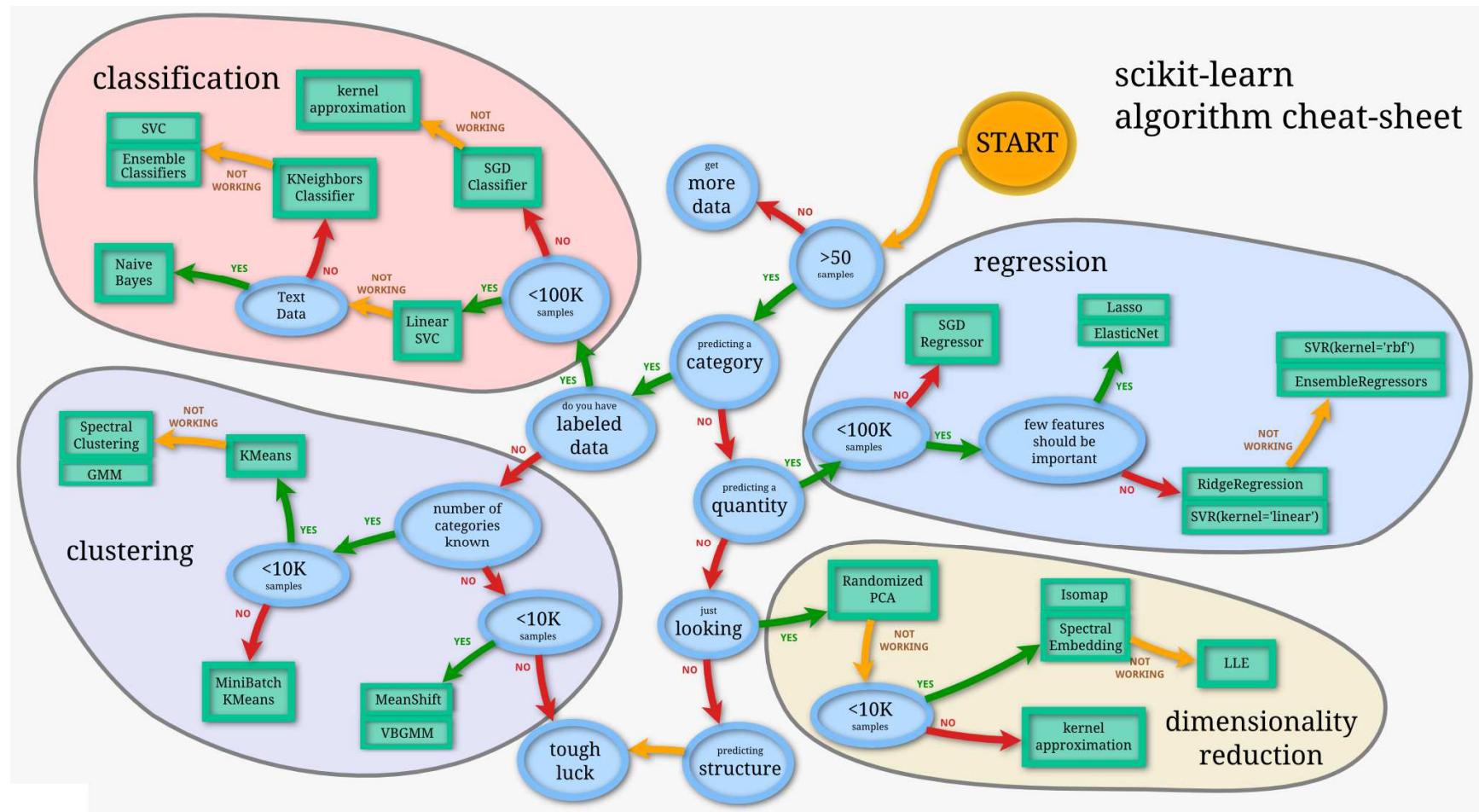
メニュー

- データの読み込み
- データフレームの要約
- 種々の統計手法
- 種々の回帰分析、ダミー変数、データの標準化、他
- **Scikit-learn: 機械学習の各種手法**
- モデルの評価
- その他

※ 本資料では、普通の python を python、Google Colaboratory を Colab と略記します 43

引用: Scikit-learn algorithm cheat-sheet

- Choosing the right estimator for solving a machine learning problem
https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
- 上記リンク先に下記チャートがあり、手法名をクリックすると説明へ飛ぶ



分類: SGD(*Stochastic Gradient Descent*)

- 手法の詳細:

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

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

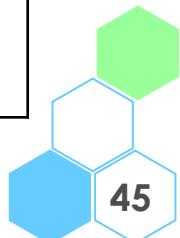
https://scikit-learn.org/stable/auto_examples/linear_model/plot_sgd_iris.html

```
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score

x_list = ["SL", "SW", "PL", "PW"]
y_list = ["SP"]
data_x = iris[x_list]
data_y = iris[y_list]

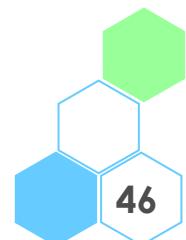
clf = SGDClassifier(loss="hinge", penalty="l2", max_iter=100,
                     random_state=777)
clf.fit(data_x, data_y.values.ravel()) # 実行
y_pred = clf.predict(data_x)          # 予測してみる
print(accuracy_score(data_y, y_pred)) # 予測的中率
```

0.94



SGD(*Stochastic Gradient Descent*)の補足

- SGD: SGDClassifier() の引数 loss について
 - `loss="hinge"`: (soft-margin) linear Support Vector Machine に相当
 - `loss="log"`: logistic regression に相当
 - `loss="perceptron", eta0=1, learning_rate="constant", penalty=None`: perceptron(ちなみに Perceptron() は SGDClassifier() を継承) に相当
 - `loss='squared_loss', penalty='l2'`: Ridge 回帰を適用する際のペナルティ関数を使用
- SGDの利点:
 - 効率が良い
 - コードの修正が容易
- SGDの欠点:
 - くり返し数や Regularization parameter 等の hyperparameter を多数調整する必要あり
 - Scaling の方法に敏感.



分類: Kernel Approximation + SGD

- 手法の詳細:

https://scikit-learn.org/stable/modules/kernel_approximation.html

https://scikit-learn.org/stable/modules/generated/sklearn.kernel_approximation.RBFSampler.html

```
from sklearn.kernel_approximation import RBFSampler
from sklearn.linear_model import SGDClassifier
from sklearn.metrics import accuracy_score

x_list = ["SL", "SW", "PL", "PW"]
y_list = ["SP"]
data_x = iris[x_list]
data_y = iris[y_list]

rbf_feature = RBFSampler(gamma=1, random_state=777)
X_features = rbf_feature.fit_transform(data_x)
clf = SGDClassifier(loss="hinge", penalty="l2", max_iter=100,
                     random_state=777)
clf.fit(X_features, data_y.values.ravel()) # 実行
y_pred = clf.predict(X_features)          # 予測してみる
print(accuracy_score(data_y, y_pred))      # 予測的中率
```

0.9666666666666667

分類: SVM(*Support Vector Machine*)

- 手法の詳細:

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
https://scikit-learn.org/stable/auto_examples/svm/plot_iris_svc.html

```
from sklearn import svm
from sklearn.metrics import accuracy_score

x_list = ["SL", "SW", "PL", "PW"]
y_list = ["SP"]
data_x = iris[x_list]
data_y = iris[y_list]

C = 1.0 # SVM regularization parameter
clf = svm.SVC(kernel='linear', C=C, random_state=777)
# clf = svm.LinearSVC(C=C, max_iter=10000, random_state=777)
# clf = svm.SVC(kernel='rbf', gamma=0.7, C=C, random_state=777)
# clf = svm.SVC(kernel='poly', degree=3, gamma='auto', C=C,
#                 random_state=777)
clf.fit(data_x, data_y.values.ravel()) # 実行
y_pred = clf.predict(data_x)          # 予測してみる
print(accuracy_score(data_y, y_pred)) # 予測的中率
```

0.933333333333

分類: K-Nearest Neighbors Classification

- 手法の詳細:

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

https://scikit-learn.org/stable/auto_examples/neighbors/plot_classification.html

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score

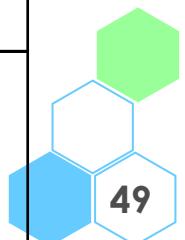
x_list = ["SL", "SW", "PL", "PW"]
y_list = ["SP"]
data_x = iris[x_list]
data_y = iris[y_list]
# n_neighborsは可変、weights='uniform'も可
clf = KNeighborsClassifier(n_neighbors=5, weights='distance')
clf.fit(data_x, data_y.values.ravel()) # 実行
y_pred = clf.predict(data_x)          # 予測してみる
print(accuracy_score(data_y, y_pred)) # 予測的中率

cv = cross_val_score(clf, data_x, data_y.values.ravel(), cv=10)
print(cv.mean())        # 平均的中率と
print(cv.std())         # その標準偏差
```

1.0

0.9666666666666668

0.04472135954999579



分類: CART

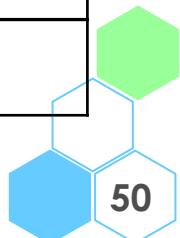
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score

x_list = ["SL", "SW", "PL", "PW"]
y_list = ["SP"]
data_x = iris[x_list]
data_y = iris[y_list]

# 回帰木の場合は DecisionTreeRegressor(max_depth=2) 等とする
clf = DecisionTreeClassifier(max_depth=2)
clf.fit(data_x, data_y)          # 実行
y_pred = clf.predict(data_x)     # 予測してみる
print(accuracy_score(data_y, y_pred)) # 予測的中率

plt.figure(figsize=(18, 12))
plot_tree(clf, feature_names=x_list,
          class_names=["setosa", "versicolor", "virginica"],
          filled=True, rounded=True, proportion=True, fontsize=10)
```

0.96

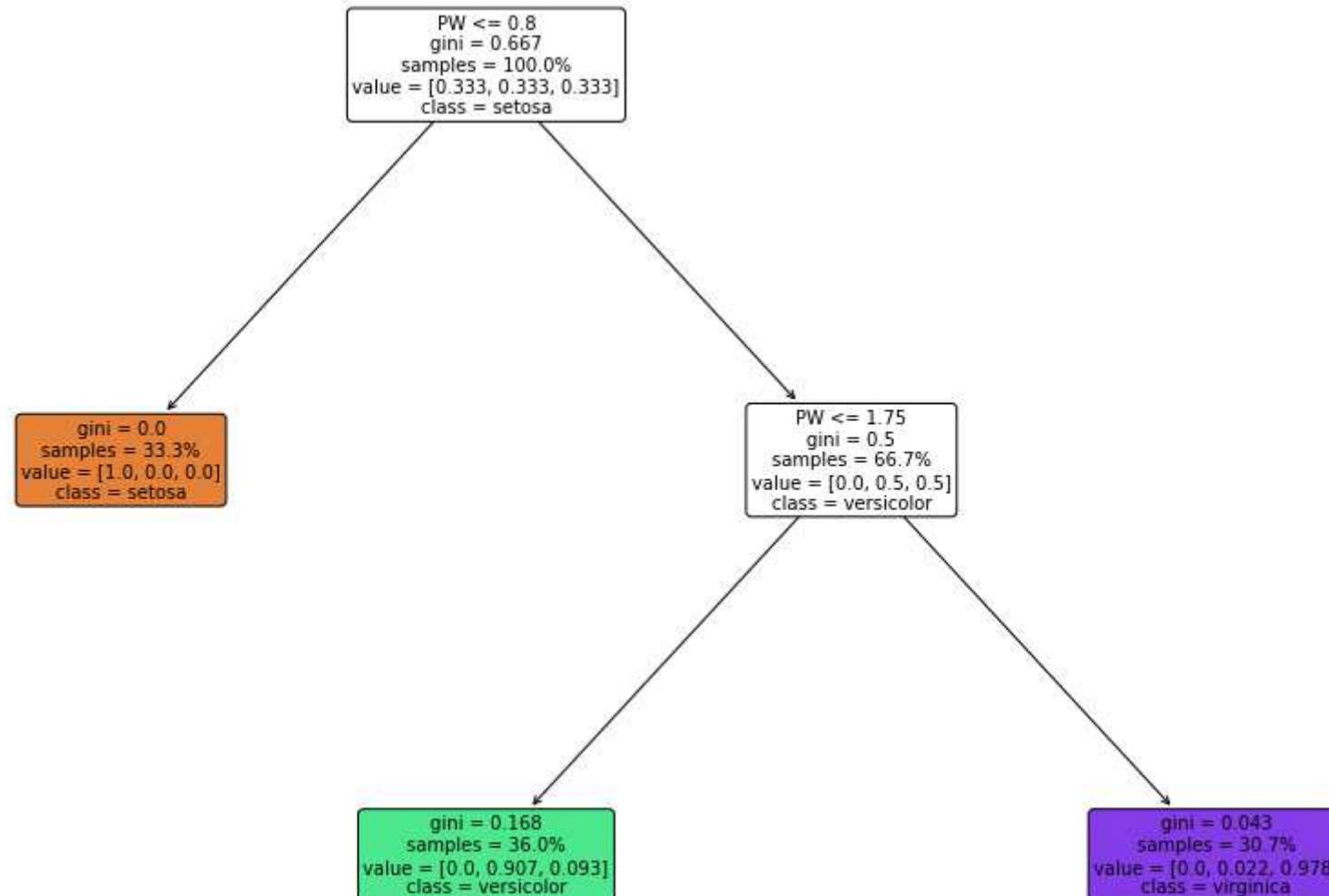


分類: CART

- 手法の詳細:

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

<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>



分類: Random Forest

- 手法の詳細:

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

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

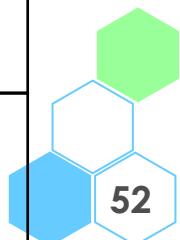
x_list = ["SL", "SW", "PL", "PW"]
y_list = ["SP"]
data_x = iris[x_list]
data_y = iris[y_list]

clf = RandomForestClassifier(n_estimators=10, max_depth=3)
clf.fit(data_x, data_y.values.ravel()) # 実行
y_pred = clf.predict(data_x)          # 予測してみる
print(accuracy_score(data_y, y_pred)) # 予測的中率

importances = clf.feature_importances_ # 重要度
("SL", "SW", "PL", "PW")
print(importances)
```

0.9733333333333334

[0.09688497 0.00595655 0.35066177 0.54649671]



Clustering: K-means

- 手法の詳細:

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

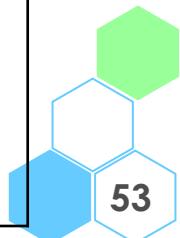
<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

```
from sklearn.cluster import KMeans, MiniBatchKMeans

x_list = ["SL", "SW", "PL", "PW"]
data_x = iris[x_list]
clust = KMeans(n_clusters=3)
#clust = MiniBatchKMeans(init='k-means++', n_clusters=3,
#                        batch_size=10, n_init=10,
#                        max_no_improvement=10, verbose=0) # データ数>1万
clust.fit(data_x)
clust.labels_ # clust.labels_.tolist()でリスト化
clust.fit_predict(data_x) # 上記と同じ機能

array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 2, 1, 2, 2, 2, 2, 1, 2, 1, 2, 1, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2,
       2, 1, 2, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1, 2, 2, 2, 1],  

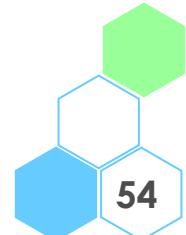
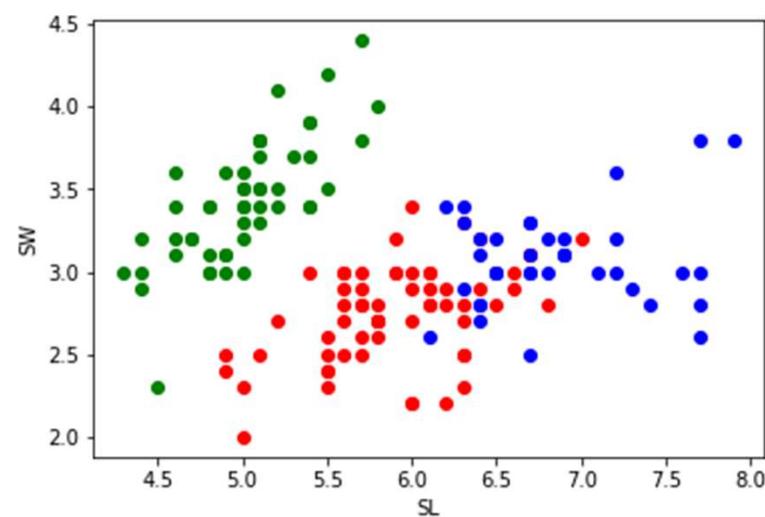
      dtype=int32)
```



Clustering: K-means

- クラスタリング結果を図示

```
labels = clust.labels_
plt.figure(1)
tmp = data_x[labels == 0]
plt.scatter(tmp.SL, tmp.SW, color='red')
tmp = data_x[labels == 1]
plt.scatter(tmp.SL, tmp.SW, color='green')
tmp = data_x[labels == 2]
plt.scatter(tmp.SL, tmp.SW, color='blue')
plt.xlabel('SL')
plt.ylabel('SW')
plt.show()
```



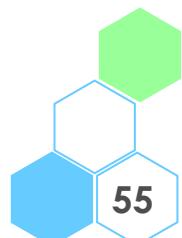
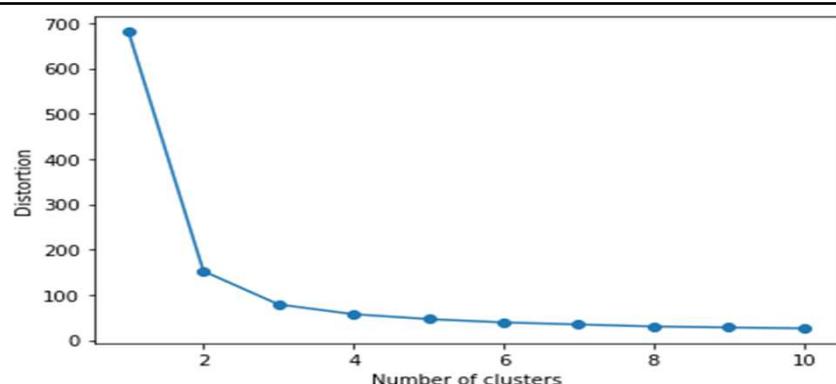
Clustering: K-means

- 手法の詳細:

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

<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html>

```
# Elbow method to find the optimal number of clusters
distortions = []
for i in range(1, 11):
    km = KMeans(n_clusters=i, init='k-means++', n_init=10,
                 max_iter=300, random_state=777)
    km.fit(data_x)
    distortions.append(km.inertia_)
plt.plot(range(1, 11), distortions, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Distortion')
plt.tight_layout()
plt.show() # クラスターの数が3のときがelbow(最適)
```



Clustering: その他の手法

- Clustering の詳細: <https://scikit-learn.org/stable/modules/clustering.html>
Mean Shift → <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html>
GMM & VBGMM → → <https://scikit-learn.org/stable/modules/mixture.html>
<http://scikit-learn.org/stable/modules/generated/sklearn.mixture.BayesianGaussianMixture.html>
Spectral Clustering → <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html>

```
from sklearn.cluster import MeanShift, SpectralClustering
from sklearn.mixture import GaussianMixture

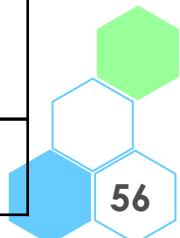
x_list = ["SL", "SW", "PL", "PW"]
data_x = iris[x_list]

ms = MeanShift()
ms.fit(data_x)
ms.labels_

# cov_type: 'spherical', 'diag', 'tied', 'full'
gmm = GaussianMixture(n_components=3, covariance_type='tied',
                      max_iter=20, random_state=777)
gmm.fit(data_x).predict(data_x)

sc = SpectralClustering(n_clusters=3,
                        assign_labels="discretize", random_state=777).fit(data_x)
sc.labels_
```

略



メニュー

- データの読み込み
- データフレームの要約
- 種々の統計手法
- 種々の回帰分析、ダミー変数、データの標準化、他
- Scikit-learn: 機械学習の各種手法
- **モデルの評価**
- その他

※ 本資料では、普通の python を python、Google Colaboratory を Colab と略記します 57

Holdout 法

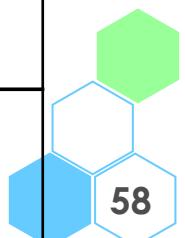
- モデルの評価を行うため、`sklearn.model_selection` の `train_test_split()` により元データを「学習用データ」と「テスト用データ」に分ける
 - `x_train`、`x_test`: 説明変数の「学習用データ」と「テスト用データ」
 - `y_train`、`y_test`: 目的変数の「学習用データ」と「テスト用データ」
 - `test_size=0.3`: 「テスト用データ」の割合（全体の 30%）
 - `stratify` にカテゴリ変数を指定することで層化抽出も可

参考: `Kfold()`、`StratifiedKFold()`、`GroupKFold()`、`ShuffleSplit()`、`GroupShuffleSplit()`、`StratifiedShuffleSplit()` 等のデータ分割関数がある

```
from sklearn.model_selection import train_test_split
x_list = ["SL", "PL", "PW"]          # 説明変数名
y_list = ["SW"]                      # 目的変数名
data_x = iris[x_list]                # 説明変数
data_y = iris[y_list]                # 目的変数

x_train, x_test, y_train, y_test = \
    train_test_split(data_x, data_y, test_size=0.3)
print(x_train.shape); print(x_test.shape)
print(y_train.shape); print(y_test.shape)

(105, 3)   (45, 3)
(105, 1)   (45, 1)
```



Holdout 法: Root MSE(RMSE)

1. 「学習用データ」で回帰モデル（21頁の普通の回帰分析モデル）を構築
2. 「テスト用データ」で RMSE を計算、RMSE が小さいモデルが良い

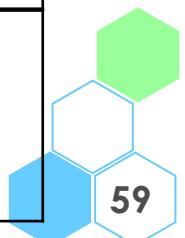
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train, y_train)                      # 普通の回帰分析

def myRMSE(x, y):
    n = len(x)
    z = pow( pow(x-y,2).sum() /n, 1/2)
    return(z)

y_pred = lr.predict(x_test)                    # テスト用データで予測
print( myRMSE(y_pred, y_test) )               # 実際の値との乖離をRMSEにて
```

```
from statsmodels.tools.eval_measures import rmse
rmse = rmse(y_pred, y_test)                  # RMSE
print( rmse )
```

```
SW 0.308755
dtype: float64
[0.30875474]
```



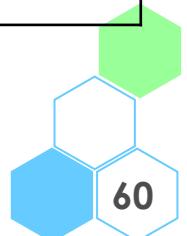
Cross Validation (CV)

- 手法の詳細:
https://scikit-learn.org/stable/modules/cross_validation.html
- scoring で指定できる指標一覧:
https://scikit-learn.org/stable/modules/model_evaluation.html
- 以下では、データを 10分割(cv=10)し、「9 個でモデル構築、残りの 1 つで
—RMSE の計算」を 10 回くり返し、「—RMSE」の平均とその標準偏差を算出

```
from sklearn.model_selection import cross_val_score

cv = cross_val_score(lr, data_x, data_y.values.ravel(), cv=10,
                      scoring='neg_root_mean_squared_error')
print(cv.mean())      # —RMSE の平均と
print(cv.std())       # その標準偏差
```

```
-0.30566231059078913
0.06359518506847732
```



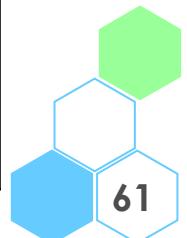
Cross Validation (CV)

- 手法の詳細:
https://scikit-learn.org/stable/modules/cross_validation.html
- scoring で指定できる指標一覧:
https://scikit-learn.org/stable/modules/model_evaluation.html
- cross_validate() を使用すると複数の指標に対して一度に計算できる

```
from sklearn.model_selection import cross_validate

methods = { 'R2': 'r2', 'nRMSE': 'neg_root_mean_squared_error',
            'nMSE': 'neg_mean_squared_error' }
cv      = cross_validate(lr, data_x, data_y.values.ravel(),
                        cv=10, scoring=methods)
print(cv['test_R2'].mean())          # R2の平均
print(cv['test_nRMSE'].mean())       # -RMSEの平均
print(cv['test_nMSE'].mean())        # -MSEの平均
```

0.008091101548069402
-0.30566231059078913
-0.09747379567959395



Grid Search CV で最良のモデル探索

- 手法の詳細:

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

scoring で指定できる指標一覧:

https://scikit-learn.org/stable/modules/model_evaluation.html

```
from sklearn import svm
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

x_list = ["SL", "SW", "PL", "PW"] ; y_list = ["SP"]
data_x = iris[x_list]           ; data_y = iris[y_list]

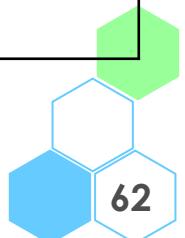
svc      = svm.SVC()           # 34頁のSVM
params   = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
clf      = GridSearchCV(svc, params, cv=10, scoring='accuracy') # 探索
clf.fit(data_x, data_y.values.ravel())
# print(sorted(clf.cv_results_.keys()))
print(clf.best_estimator_)      # 最良のモデル
print(clf.best_params_)         # 最良のパラメータ

clf = clf.best_estimator_       # 最良のモデルで
clf.fit(data_x, data_y.values.ravel()) # 実行
y_pred = clf.predict(data_x)     # 予測してみる
print(accuracy_score(data_y, y_pred)) # 予測的中率
```

SVC(C=10, break_ties=False, cache_size=200, ...)

{'C': 10, 'kernel': 'linear'}

0.98



メニュー

- データの読み込み
- データフレームの要約
- 種々の統計手法
- 種々の回帰分析、ダミー変数、データの標準化、他
- Scikit-learn: 機械学習の各種手法
- モデルの評価
- **その他**

※ 本資料では、普通の python を python、Google Colaboratory を Colab と略記します 63

種々のクロス集計

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'BLOOD': ['A', 'B', 'O', 'A', 'B', 'O'],
                   'GENDER': ['M', 'F', 'M', 'F', 'M', 'F'],
                   'SCORE': [10, 10, 20, 20, 10, 10]})

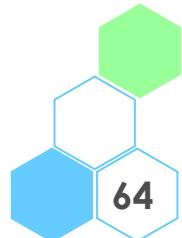
pd.crosstab(index=df['BLOOD'], columns=df['GENDER'])
pd.crosstab(index=[df['BLOOD'], df['GENDER']],
             columns=df['SCORE'])
pd.crosstab(index=df['BLOOD'], columns=df['GENDER'],
             margins=True)
pd.crosstab(index=df['BLOOD'], columns=df['GENDER'],
             values=df['SCORE'], aggfunc=np.mean)
```

GENDER	F	M
BLOOD		
A	1	1
B	1	1
O	1	1

SCORE	10	20
BLOOD	GENDER	
A	F	0 1
	M	1 0
B	F	1 0
	M	1 0
O	F	1 0
	M	0 1

GENDER	F	M	All
BLOOD			
A	1	1	2
B	1	1	2
O	1	1	2
All	3	3	6

GENDER	F	M
BLOOD		
A	20	10
B	10	10
O	10	20



標準化 → PCAで次元削減 → ロジスティック回帰で学習

```

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression

x_list = ["SL", "SW", "PL", "PW"]
d_list = ["SP"]
iris_d = pd.concat([iris[x_list], pd.get_dummies(iris[d_list])], axis=1)
y_list = ["SP_virginica"] # virginica:1、それ以外:0
data_x = iris_d[x_list]
data_y = iris_d[y_list]

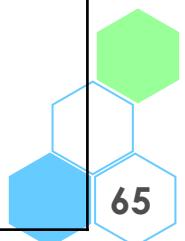
X_train, X_test, y_train, y_test = \
    train_test_split(data_x, data_y, test_size=0.3,
                     stratify=data_y, random_state=777)

sc = StandardScaler() # データを標準化
X_train_std = sc.fit_transform(X_train) # データを正規化
X_test_std = sc.transform(X_test) # X_trainを正規化した変換式で変換

pca = PCA(n_components=2)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)

lr = LogisticRegression(multi_class='ovr', random_state=777,
solver='lbfgs')
lr = lr.fit(X_train_pca, y_train)
y_pred = lr.predict(X_test_pca)
print('Test Accuracy: %.3f' % lr.score(X_test_pca, y_test))

```



パイプライン

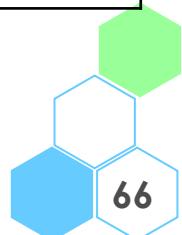
- `make_pipeline()` で一連の処理の wrapper を実現
 - 前頁の後半部分をパイプラインで記載簡略

```
from sklearn.pipeline import make_pipeline
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression

pipe_lr = make_pipeline(StandardScaler(),
    PCA(n_components=2),
    LogisticRegression(random_state=777, solver='lbfgs'))

pipe_lr.fit(X_train, y_train)
y_pred = pipe_lr.predict(X_test)
print('Test Accuracy: %.3f' % pipe_lr.score(X_test, y_test))
```

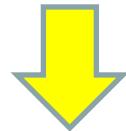
Test Accuracy: 0.911



CVに関する補足

```
from sklearn.model_selection import StratifiedKFold

kfold = StratifiedKFold(n_splits=10).split(X_train, y_train)
scores = []
for k, (train, test) in enumerate(kfold):
    pipe_lr.fit(X_train.iloc[train], y_train.iloc[train])
    score = pipe_lr.score(X_train.iloc[test], y_train.iloc[test])
    scores.append(score)
    print('Fold: %2d, Class dist.: %s, Acc: %.3f' % (k+1,
        np.bincount(y_train.iloc[train].values.ravel()), score))
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))
```



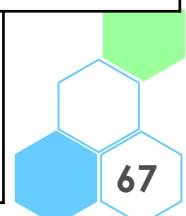
明示的にデータを分割して CV を行うことを簡略化

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(estimator=pipe_lr, X=X_train, y=y_train, cv=10, n_jobs=1)
print('CV accuracy scores: %s' % scores)
print('CV accuracy: %.3f +/- %.3f' % (np.mean(scores), np.std(scores)))

from sklearn.model_selection import learning_curve
```

CV accuracy scores: [0.90909091 0.90909091 0.90909091 0.90909091
1. 1. 1. 0.9 0.9 0.7]
CV accuracy: 0.914 +/- 0.083

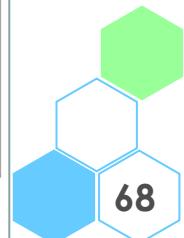
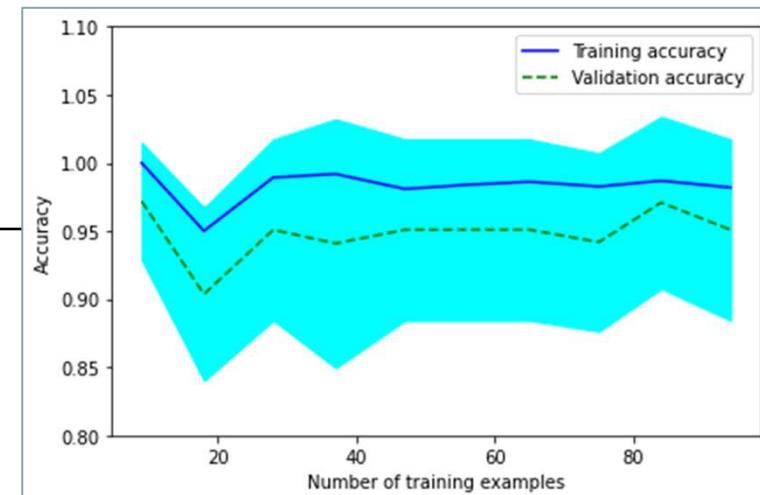


CVに関する補足

- ロジスティック回帰モデルに L2正則項を入れ、学習用データの割合を 0.1~1(10%~100%)に変動させ accuracy を計算

```
from sklearn.model_selection import learning_curve
pipe_lr = make_pipeline(StandardScaler(),
    LogisticRegression(penalty='l2', solver='lbfgs', random_state=777, max_iter=10000))
train_sizes, train_scores, test_scores = learning_curve(estimator=pipe_lr,
    X=X_train, y=y_train, train_sizes=np.linspace(0.1, 1.0, 10), cv=10)
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.fill_between(train_sizes, train_mean + train_std, train_mean - train_std, color='blue')
plt.fill_between(train_sizes, test_mean + test_std, test_mean - test_std, color='cyan')
plt.plot(train_sizes, train_mean, color='blue', markersize=5, label='Training accuracy')
plt.plot(train_sizes, test_mean, color='green', linestyle='--', markersize=5,
    label='Validation accuracy')
plt.xlabel('Number of training examples')
plt.ylabel('Accuracy')
plt.legend(loc='upper right')
plt.ylim([0.8, 1.1])
plt.tight_layout()
plt.show()
```

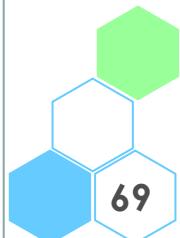
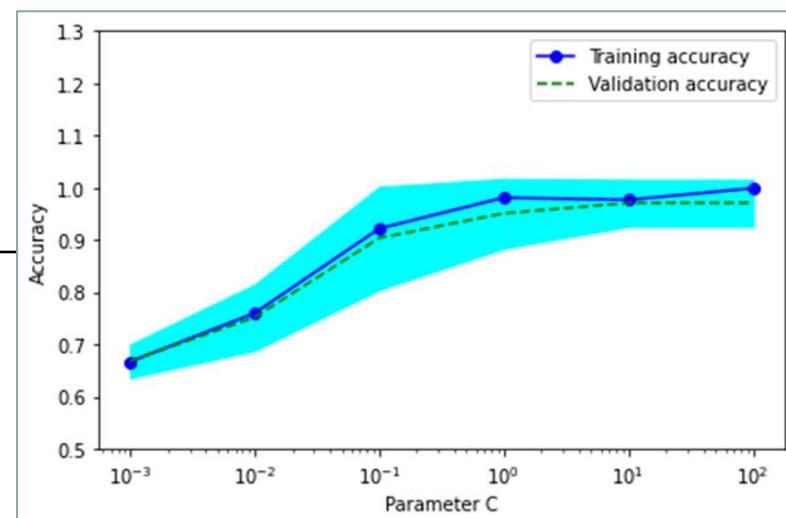


CVに関する補足

- ロジスティック回帰モデルに L2正則項を入れ、正則項の重みの逆数 C(大きい方が罰則小)の値を変動させ accuracy を計算

```
from sklearn.model_selection import validation_curve
range = [0.001, 0.01, 0.1, 1.0, 10.0, 100.0]
train_scores, test_scores = validation_curve(estimator=pipe_lr,
X=X_train, y=y_train, param_name='logisticregression__C', param_range=range, cv=10)
train_mean = np.mean(train_scores, axis=1)
train_std = np.std(train_scores, axis=1)
test_mean = np.mean(test_scores, axis=1)
test_std = np.std(test_scores, axis=1)

plt.fill_between(range, train_mean + train_std, train_mean - train_std, color='blue')
plt.fill_between(range, test_mean + test_std, test_mean - test_std, color='cyan')
plt.plot(range, train_mean, color='blue', marker='o', label='Training accuracy')
plt.plot(range, test_mean, color='green', linestyle='--', markersize=5,
label='Validation accuracy')
plt.xlabel('Parameter C')
plt.ylabel('Accuracy')
plt.legend(loc='upper right')
plt.xscale('log')
plt.ylim([0.5, 1.3])
plt.tight_layout()
plt.show()
```



Bagging

- 学習用データからブートストラップ抽出することを n_estimators 回行う、それぞれに対してモデルを作成して予測を実行
- 結果は、分類(BaggingClassifier)であれば多数決で集約
 - ちなみに、回帰を行う場合(BaggingRegressor)は平均値等で集約

```
from sklearn.tree      import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.metrics   import accuracy_score

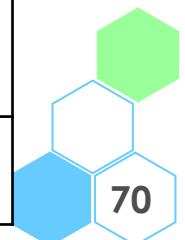
X_train, X_test, y_train, y_test = train_test_split(data_x, data_y, test_size=0.5,
                                                    stratify=data_y, random_state=777)

tree = DecisionTreeClassifier(criterion='entropy', max_depth=None, random_state=777)
bag  = BaggingClassifier(base_estimator=tree, n_estimators=500, max_samples=1.0,
                        max_features=1.0, bootstrap=True, bootstrap_features=False, random_state=777)

tree      = tree.fit(X_train, y_train)
y_train_pred = tree.predict(X_train)
y_test_pred = tree.predict(X_test)
tree_train  = accuracy_score(y_train, y_train_pred)
tree_test   = accuracy_score(y_test, y_test_pred)
print('Decision tree train/test accuracies %.3f/%.3f' % (tree_train, tree_test))

bag      = bag.fit(X_train, y_train)
y_train_pred = bag.predict(X_train)
y_test_pred = bag.predict(X_test)
bag_train  = accuracy_score(y_train, y_train_pred)
bag_test   = accuracy_score(y_test, y_test_pred)
print('Bagging train/test accuracies %.3f/%.3f' % (bag_train, bag_test))
```

Decision tree train/test accuracies 1.000/0.947
 Bagging train/test accuracies 1.000/0.960



Boosting(AdaBooost)

1. 学習用データから非復元抽出、モデルを作成して予測を実行
2. 学習用データから非復元抽出、モデル作成時に「#1で誤分類したデータの重みを相対的に増やす」処理をお加えた上でモデルを作成して予測
3. #2 を n_estimators 回実行(ちなみに、回帰の場合は BaggingRegressor())

```
from sklearn.tree      import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics   import accuracy_score

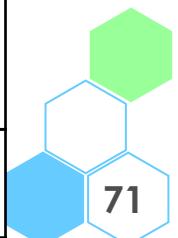
tree = DecisionTreeClassifier(criterion='entropy', max_depth=1, random_state=777)
ada  = AdaBoostClassifier(base_estimator=tree, n_estimators=500, learning_rate=0.1,
random_state=777)

tree      = tree.fit(X_train, y_train)
y_train_pred = tree.predict(X_train)
y_test_pred  = tree.predict(X_test)
tree_train   = accuracy_score(y_train, y_train_pred)
tree_test    = accuracy_score(y_test, y_test_pred)
print('Decision tree train/test accuracies %.3f/%.3f' % (tree_train, tree_test))

ada      = ada.fit(X_train, y_train)
y_train_pred = ada.predict(X_train)
y_test_pred  = ada.predict(X_test)

ada_train = accuracy_score(y_train, y_train_pred)
ada_test  = accuracy_score(y_test, y_test_pred)
print('AdaBoost train/test accuracies %.3f/%.3f' % (ada_train, ada_test))

Decision tree train/test accuracies 0.973/0.947
AdaBoost train/test accuracies 1.000/0.920
```



Decision Tree Regression

- 手法の詳細:https://scikit-learn.org/stable/auto_examples/tree/plot_tree_regression.html
<https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html>

```

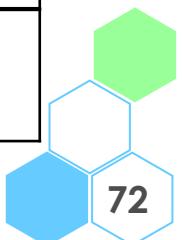
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score

x_list = ["SL", "PL", "PW"]          # 説明変数名
y_list = ["SW"]                      # 目的変数名
data_x = iris[x_list]                # 説明変数
data_y = iris[y_list]                # 目的変数

tree = DecisionTreeRegressor(max_depth=3)
tree.fit(data_x, data_y)
cv = cross_val_score(tree, data_x, data_y, cv=10,
scoring='neg_root_mean_squared_error')
print(cv.mean())      # -RMSEの平均
print(cv.std())       # その標準偏差

```

-0.2764791276356073
0.07166560044254297



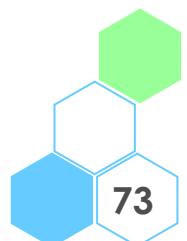
Random Forest Regression

- 手法の詳細:<https://scikit-learn.org/stable/modules/ensemble.html#forest>
<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score

forest = RandomForestRegressor(n_estimators=1000,
criterion='mse', random_state=777)
forest.fit(data_x, data_y)
cv = cross_val_score(forest, data_x, data_y, cv=10,
scoring='neg_mean_squared_error')
print(cv.mean())      # -MSEの平均
print(cv.std())       # その標準偏差
```

```
-0.08521257467062979
0.0408052842521939
```



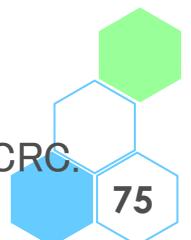
メニュー

- データの読み込み
- データフレームの要約
- 種々の統計手法
- 種々の回帰分析、ダミー変数、データの標準化、他
- Scikit-learn: 機械学習の各種手法
- モデルの評価
- その他

※ 本資料では、普通の python を python、Google Colaboratory を Colab と略記します 74

参考文献

- Python 3.8.3 documentation
<https://docs.python.org/3/>
<https://docs.python.org/ja/3/>
- Pandas User's Guide
https://pandas.pydata.org/pandas-docs/stable/user_guide/index.html
- matplotlib documentation
<https://matplotlib.org/index.html>
https://matplotlib.org/1.5.1/faq/usage_faq.html
- seaborn documentation
<https://seaborn.pydata.org/>
- scipy.org
<https://docs.scipy.org/doc/scipy/reference/index.html>
- Scikit-learn documentation
<https://scikit-learn.org/stable/>
https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html
- Wes McKinney (2018) "Python for Data Analysis 2nd Edition", O'Reilly Media, Inc.
<https://evanli.github.io/programming-book-3/Python/Python%20for%20Data%20Analysis%20-%20Wes%20McKinney.pdf>
<https://github.com/wesm/pydata-book>
- Sebastian Raschka & Vahid Mirjalili (2019) "Python Machine Learning, 3rd Edition", Packt Publishing
<https://github.com/rasbt/python-machine-learning-book-3rd-edition>
- note.nkmk.me
<https://note.nkmk.me/python/>
- Hernan MA, Robins JM (2020) "Causal Inference: What If. Boca Raton", Chapman & Hall/CRC.
<https://www.hsph.harvard.edu/miguel-hernan/causal-inference-book/>





– End of File –

