ラビットチャレンジstudyai0027863

機械学習

実装演習レポート北村裕斗 hiroto7018@gmail.com

ビデオ視聴学習者提出区分け	科目	章タイトル	1点100文字以上で要点のまとめ	実装演習結果キャプチャー又はサマリーと考察	「確認テスト」など、自身の考察結果	演習問題や参考図書、修了課題など関連記事レポートによる加点
【b】 1つのURLで提出	機械学習 （基準点：12点）	線形回帰モデル	1点	1点	不要	1点
		非線形回帰モデル	1点	1点	不要	1点
		ロジスティック回帰モデル	1点	1点	不要	1点
		主成分分析	1点	1点	不要	1点
		アルゴリズム	1点	1点	不要	1点
		サポートベクターマシーン	1点	1点	不要	1点

- 講義動画視聴およびソース実装演習を確実に実施しているかを審査します。
  - レポートが講義本筋に沿ってまとめてあるか。受講者自身の言葉で課題や気づきが記載されているか。
- 必須要件を満たさない場合、他の受講者のレポートのコピーなど不正が発覚した場合、差し戻しとします。
- 数式やコード演習結果の個々の間違いは審査に影響しません。なお、講師からのフィードバックはありませんのでご了承ください。
- 各科目以下の場合は差し戻し
  - 基準点以下
  - 実装演習を全く行なっていない場合（応用数学以外）

下記の要件の通り、区分ごとに単元レポートを作成、ご提出ください。

1）各章につき100文字以上で要点をまとめ、実装演習結果、確認テストについての自身の考察等を取り入れたレポートとする。

2）各科目の基準点が足りない場合、実装演習が不足する場合は差し戻しとする。

※各章は講義動画および講義資料（PDF）でご確認ください。

- **レポート掲載場所：自身のGitHub、SlideShareなどのBlogやWeb媒体で作成。お持ちでない方は新規アカウント（無料）を作成。**
- **レポート提出方法：レポート掲載ページのURLを【学習システム（LMS）内の指定フォーム】より提出。**

線形回帰モデル

1)

a.アウトライン

線形回帰モデル

○回帰問題を解くための機械学習モデルの一つ

○教師あり学習(教師データから学習)

○入力とm次元パラメータの線形結合を出力するモデル

■慣例として予測値にはハットを付ける(正解データとは異なる)

線形結合(入力とパラメータの内積)

○入力ベクトルと未知のパラメータの各要素を掛け算し足し合わせたもの

○入力ベクトルとの線形結合に加え、切片も足し合わせる

○(入力のベクトルが多次元でも)出力は1次元(スカラー)となる

モデルのパラメータ

○モデルに含まれる推定すべき未知のパラメータ

○特徴量が予測値に対してどのように影響を与えるかを決定する重みの集合

■正の(負の)重みをつける場合、その特徴量の値を増加させると、予測の値が増加(減少)

■重みが大きければ(0であれば)、その特徴量は予測に大きな影響力を持つ(全く影響しない)

○切片

■y軸との交点を表す

b.数学的定式化

i.データ形式の説明

●データの分割

○学習用データ:機械学習モデルの学習に利用するデータ

○検証用データ:学習済みモデルの精度を検証するためのデータ

●なぜ分割するか

○モデルの汎化性能(Generalization)を測定するため

○データへの当てはまりの良さではなく、未知のデータに対してどれくらい精度が高いかを測りたい

ii.線形回帰モデルの説明

学習に使ったデータと別のデータで検証(モデルの未来のデータへの当てはまりの良さ= 汎化性能)

iii.パラメータの推定

線形回帰モデルのパラメータは最小二乗法で推定

●平均二乗誤差(残差平方和)

○データとモデル出力の二乗誤差の和

■小さいほど直線とデータの距離が近い

■動画解説

○パラメータのみに依存する関数

■データは既知の値でパラメータのみ未知

●最小二乗法

○学習データの平均二乗誤差を最小とするパラメータを探索

○学習データの平均二乗誤差の最小化は、その勾配が0になる点を求めれば良い

iv.モデルの評価

[topic] 最尤法による回帰係数の推定○誤差を正規分布に従う確率変数を仮定し尤度関数の最大化を利用した推定も可能○線形回帰の場合には、最尤法による解は最小二乗法の解と一致

1.ハンズオン

a.pandasとnumpyの紹介

b.scikit-learnの紹介

c.住宅価格予測

In [39]:
```
import pandas as pd
import numpy as np
from sklearn.datasets import load_boston
boston = load_boston()
```

In [41]:
```
df = pd.DataFrame(boston.data, columns=boston.feature_names)
df
```

Out[41]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33
...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0.0	0.573	6.593	69.1	2.4786	1.0	273.0	21.0	391.99	9.67
502	0.04527	0.0	11.93	0.0	0.573	6.120	76.7	2.2875	1.0	273.0	21.0	396.90	9.08
503	0.06076	0.0	11.93	0.0	0.573	6.976	91.0	2.1675	1.0	273.0	21.0	396.90	5.64
504	0.10959	0.0	11.93	0.0	0.573	6.794	89.3	2.3889	1.0	273.0	21.0	393.45	6.48
505	0.04741	0.0	11.93	0.0	0.573	6.030	80.8	2.5050	1.0	273.0	21.0	396.90	7.88

506 rows × 13 columns

In [46]:
```
df.describe()
```

Out[46]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRA
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.0
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.45
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.60
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.40
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.05
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.20
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.00

In [43]:
```
import seaborn as sns
```

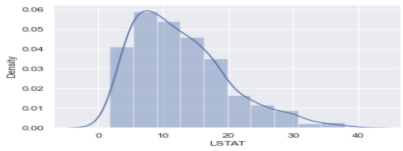
In [49]:
```
df['CRIM']
```

Out[49]:
```
0      0.00632
1      0.02731
2      0.02729
3      0.03237
4      0.06905
        ...
501    0.06263
502    0.04527
503    0.06076
504    0.10959
505    0.04741
Name: CRIM, Length: 506, dtype: float64
```

In [50]:
```
sns.set()
```

In [53]:
```
sns.distplot(df['LSTAT'], bins=10);
```

```
/Users/kitamurahiroto/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



In [ ]:
```
```

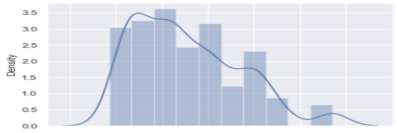
In [54]:
```
df.corr()
```

Out[54]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B
CRIM	1.000000	-0.200469	0.406583	-0.055892	0.420972	-0.219247	0.352734	-0.379670	0.625505	0.582764	0.289946	-0.385
ZN	-0.200469	1.000000	-0.533828	-0.042697	-0.516604	0.311991	-0.569537	0.664408	-0.311948	-0.314563	-0.391679	0.1755
INDUS	0.406583	-0.533828	1.000000	0.062938	0.763651	-0.391676	0.644779	-0.708027	0.595129	0.720760	0.383248	-0.356
CHAS	-0.055892	-0.042697	0.062938	1.000000	0.091203	0.091251	0.086518	-0.099176	-0.007368	-0.035587	-0.121515	0.0487
NOX	0.420972	-0.516604	0.763651	0.091203	1.000000	-0.302188	0.731470	-0.769230	0.611441	0.668023	0.188933	-0.380
RM	-0.219247	0.311991	-0.391676	0.091251	-0.302188	1.000000	-0.240265	0.205246	-0.209847	-0.292048	-0.355501	0.1280
AGE	0.352734	-0.569537	0.644779	0.086518	0.731470	-0.240265	1.000000	-0.747881	0.456022	0.506456	0.261515	-0.273
DIS	-0.379670	0.664408	-0.708027	-0.099176	-0.769230	0.205246	-0.747881	1.000000	-0.494588	-0.534432	-0.232471	0.2915
RAD	0.625505	-0.311948	0.595129	-0.007368	0.611441	-0.209847	0.456022	-0.494588	1.000000	0.910228	0.464741	-0.444
TAX	0.582764	-0.314563	0.720760	-0.035587	0.668023	-0.292048	0.506456	-0.534432	0.910228	1.000000	0.460853	-0.441
PTRATIO	0.289946	-0.391679	0.383248	-0.121515	0.188933	-0.355501	0.261515	-0.232471	0.464741	0.460853	1.000000	-0.177
B	-0.385064	0.175520	-0.356977	0.048788	-0.380051	0.128069	-0.273534	0.291512	-0.444413	-0.441808	-0.177383	1.0000
LSTAT	0.455621	-0.412995	0.603800	-0.053929	0.590879	-0.613808	0.602339	-0.496996	0.488676	0.543993	0.374044	-0.366

In [55]:
```
sns.distplot(df['NOX'], bins=10);
```

```
/Users/kitamurahiroto/opt/anaconda3/lib/python3.8/site-packages/seaborn/distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.09	1.0	296.0	15.3	396.9	4.98

In [93]:
```
sample = x.iloc[0,:]
sample.shape
```

Out[93]: (12,)

In [94]:
```
np.array(sample).reshape(1,-1).shape
```

Out[94]: (1, 12)

In [96]:
```
y_pred = model.predict(np.array(sample).reshape(1,-1))
```

In [97]:
```
y_pred
```

Out[97]: array([8.81476199])

n [100]:
```
import joblib
```

n [101]:
```
joblib.dump(model, 'model.pkl')
```

ut[101]: ['model.pkl']

In [ ]:

n [103]:
```
model_new = joblib.load('model.pkl')
```

n [104]:
```
model_new.predict([sample])
```

ut[104]: array([8.81476199])

In [ ]:

In [ ]:
```
#paramerter
```

n [105]:
```
model.coef_
```

ut[105]:
```
array([ 8.94905500e-02,  2.98243442e-02,  1.43516901e-01, -1.27278630e+00,
        5.90783829e+00, -4.72102037e+00,  1.02516194e-01,  2.50149843e-01,
        7.79684302e-02, -3.60838682e-03,  1.50386432e-01, -6.35746860e-03])
```

n [106]:
```
np.set_printoptions(precision=3)
```

n [108]:
```
model.coef_
```

ut[108]:
```
array([ 8.949e-02,  2.982e-02,  1.435e-01, -1.273e+00,  5.908e+00,
       -4.721e+00,  1.025e-01,  2.501e-01,  7.797e-02, -3.608e-03,
        1.504e-01, -6.357e-03])
```

n [109]:
```
np.set_printoptions(precision=3,suppress=True)
```

n [110]:
```
model.coef_
```

ut[110]:
```
array([ 0.089,  0.03 ,  0.144, -1.273,  5.908, -4.721,  0.103,  0.25 ,
        0.078, -0.004,  0.15 , -0.006])
```

n [111]:
```
df.head(3)
```

ut[111]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03

In [ ]:

前処理が全行程の8割ほどを占める。

欠損値の除去は、欠損値が生じる行、列を削除する。

→次元削除、データの圧縮

非線形回帰モデル

a.アウトライン

複雑な非線形構造を内在する現象に対して、非線形回帰モデリングを実施○データの構造を線形で捉えられる場合は限られる○非線形な構造を捉えられる仕組みが必要

b.数学的定式化

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型結合を使用○未知パラメータは線形回帰モデルと同様に最小2乗法や最尤法により推定

よく使われる基底関数

○多項式関数

○ガウス型基底関数

○スプライン関数/ Bスプライン関数

i.非線形回帰モデルの説明

未学習(underfitting)と過学習(overfitting)

○学習データに対して、十分小さな誤差が得られないモデル→未学習

■(対策)モデルの表現力が低いため、表現力の高いモデルを利用する

○小さな誤差は得られたけど、テスト集合誤差との差が大きいモデル→過学習

■(対策1) 学習データの数を増やす

■(対策2) 不要な基底関数(変数)を削除して表現力を抑止

■(対策3) 正則化法を利用して表現力を抑止

ii.パラメータの推定

不要な基底関数を削除

○基底関数の数、位置やバンド幅によりモデルの複雑さが変化

○解きたい問題に対して多くの基底関数を用意してしまうと過学習の問題がおこるため、適切な基底関数を用意(CVなどで選択)

正則化法(罰則化法)

○「モデルの複雑さに伴って、その値が大きくなる正則化項(罰則項)を課した関数」を最小化

○正則化項(罰則項)

■形状によっていくつもの種類があり、それぞれ推定量の性質が異なる(次スライド)

○正則化(平滑化)パラメータ

■モデルの曲線のなめらかさを調節▸適切に決める必要あり

●正則化項(罰則項)の役割○無い▸最小2乗推定量○L2ノルムを利用▸Ridge推定量○L1ノルムを利用▸Lasso推定量

●正則化パラータの役割○小さく▸制約面が大きく○大きく▸制約面が小さく

iii.バイアスバリアンスのトレードオフ

●汎化性能

○学習に使用した入力だけでなく、これまで見たことのない新たな入力に対する予測性能

■(学習誤差ではなく)汎化誤差(テスト誤差)が小さいものが良い性能を持ったモデル。

■汎化誤差は通常、学習データとは別に収集された検証データでの性能を測ることで推定

○バイアス・バリアンス分解

●手元のモデルがデータに未学習しているか過学習しているか

○訓練誤差もテスト誤差もどちらも小さい▸汎化しているモデルの可能性

○訓練誤差は小さいがテスト誤差が大きい▸過学習

○訓練誤差もテスト誤差もどちらも小さくならない▸未学習

○回帰の場合には陽に解が求まります(学習誤差と訓練誤差の値を比較）

●ホールドアウト法

○有限のデータを学習用とテスト用の2つに分割し、「予測精度」や「誤り率」を推定する為に使用

■学習用を多くすればテスト用が減り学習精度は良くなるが、性能評価の精度は悪くなる

■逆にテスト用を多くすれば学習用が減少するので、学習そのものの精度が悪くなることになる。

■手元にデータが大量にある場合を除いて、良い性能評価を与えないという欠点がある。

○基底展開法に基づく非線形回帰モデルでは、基底関数の数、位置、バンド幅の値とチューニングパラメータをホールドアウト値を小さくするモデルで決定する

iv.クロスバリデーション

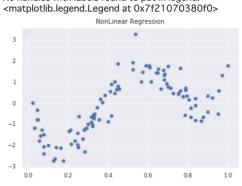
●グリッドサーチ

○全てのチューニングパラメータの組み合わせで評価値を算出

○最も良い評価値を持つチューニングパラメータを持つ組み合わせを、「いいモデルのパラメータ」として採用

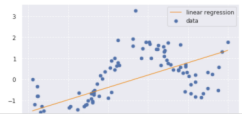
```
# 真の関数からノイズを伴うデータを生成

# 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 0x7f21070380f0>
```



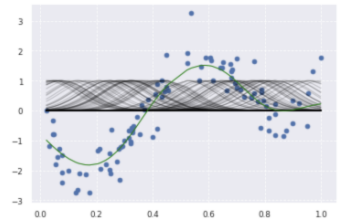

```
from sklearn.linear_model import LinearRegression

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))
```

```
0.3921547168787944
```




```
#Ridge

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

kx = rbf_kernel(X=data, Y=data, gamma=50)
#KX = rbf_kernel(X, x)

#clf = LinearRegression()
clf = Ridge(alpha=30)
clf.fit(kx, target)

p_ridge = clf.predict(kx)

plt.scatter(data, target,label='data')
for i in range(len(kx)):
    plt.plot(data, kx[i], color='black', linestyle='-', linewidth=1, markersize=3, label='rbf', alpha=0.2)

#plt.plot(data, p, color='green', marker='o', linestyle='-', linewidth=0.1, markersize=3)
plt.plot(data, p_ridge, color='green', linestyle='-', linewidth=1, markersize=3,label='ridge regression')
#plt.legend()

print(clf.score(kx, target))
```

```
0.8193413527112867
```

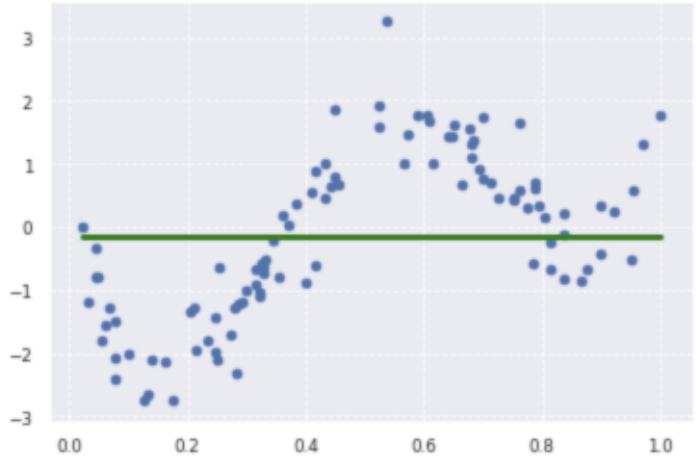


```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import Pipeline
```

```
#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(data, target)
    # make predictions
    p_poly = regr.predict(data)
    # plot regression result
    plt.scatter(data, target, label='data')
    plt.plot(data, p_poly, label='polynomial of degree %d' % (d))
```

```
[ ]  #plt.plot(data, p, color='green', marker='o', linestyle='-', linewidth=0.1, markersize=3)
     plt.plot(data, p_lasso, color='green', linestyle='-', linewidth=3, markersize=3)

     print(lasso_clf.score(kx, target))
```

```
-4.440892098500626e-16
```



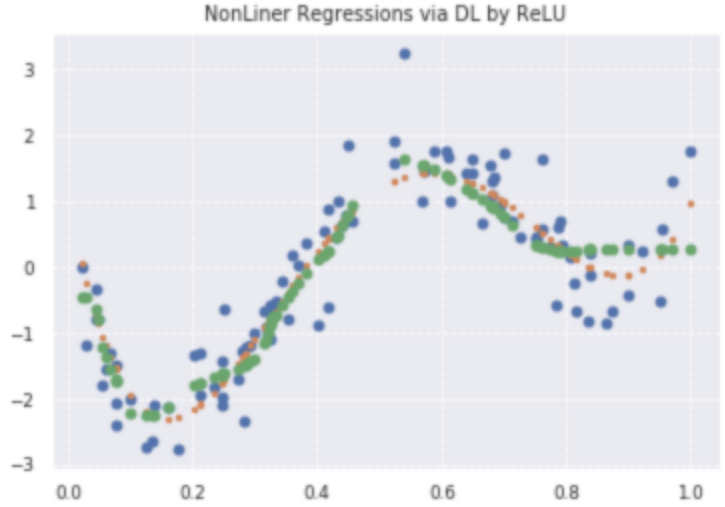

```
[ ]  from sklearn import model_selection, preprocessing, linear_model, svm

     # SVR-rbf
     clf_svr = svm.SVR(kernel='rbf', C=1e3, gamma=0.1, epsilon=0.1)
     clf_svr.fit(data, target)
     y_rbf = clf_svr.fit(data, target).predict(data)

     # plot

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

```
y_pred = estimator.predict(x_train)
```

```
90/90 [==============================] - 0s 726us/step
```

```
plt.title('NonLiner Regressions via DL by ReLU')
plt.plot(data, target, 'o')
plt.plot(data, true_func(data), '.')
plt.plot(x_train, y_pred, "o", label='predicted: deep learning')
#plt.legend(loc=2)
```

```
[<matplotlib.lines.Line2D at 0x7f20b43214e0>]
```




```
print(lasso_clf.coef_)
```

```
[-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.  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.  0.  0.  0.  0.  0.  0.  0.  0.]
```

ロジスティック回帰モデル

a.数学的定式化

i.データ形式の説明

●分類問題(クラス分類)

○ある入力(数値)からクラスに分類する問題

●分類で扱うデータ

○入力(各要素を説明変数または特徴量と呼ぶ)

■m次元のベクトル(m=1の場合はスカラ)○出力(目的変数)

■0 or 1の値

○タイタニックデータ、IRISデータなど

ii.ロジスティック回帰モデルの説明

●ロジスティック線形回帰モデル

○分類問題を解くための教師あり機械学習モデル(教師データから学習)

■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

●シグモイド関数

○入力は実数・出力は必ず0~1の値

○(クラス1に分類される)確率を表現

○単調増加関数

●パラメータが変わるとシグモイド関数の形が変わる

○aを増加させると，x=0付近での曲線の勾配が増加

○aを極めて大きくすると，単位ステップ関数(x<0でf(x)=0，x>0でf(x)=1となるような関数)に近づきます

●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利用すると計算が容易

●シグモイド関数の出力をY=1になる確率に対応させる
○データの線形結合を計算○シグモイド関数に入力▸出力が確率に対応
○[表記] i番目データを与えた時のシグモイド関数の出力をi番目のデータがY=1になる確率とする
○バイアス変化は段差の位置

iii.パラメータの推定
最尤推定

●世の中には様々な確率分布が存在する○正規分布、t分布、ガンマ分布、一様分布、ディリクレ分布・・・○ロジスティック回帰モデルではベルヌーイ分布を利用する●ベルヌーイ分布○数学において、確率pで1、確率1 − p で0をとる、離散確率分布(例:コインを投げ)○「生成されるデータ」は分布のパラメータによって異なる(この場合は確率p)

●ベルヌーイ分布のパラメータの推定○ある分布を考えた時、そのパラメータ(既知)によって、生成されるデータは変化する■0.3と0.8○データからそのデータを生成したであろう尤もらしい分布(パラメータ)を推定したい■最尤推定

●同時確率○あるデータが得られた時、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●尤度関数とは○データは固定し、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

●尤度関数を最大とするパラメータを探す(推定)○対数をとると微分の計算が簡単■同時確率の積が和に変換可能■指数が積の演算に変換可能○対数尤度関数が最大になる点と尤度関数が最大になる点は同じ■対数関数は単調増加(ある尤度の値がx1 < x2の時、必ずlog(x1) < log(x2)となる)○「尤度関数にマイナスをかけたものを最小化」し、「最小2乗法の最小化」と合わせる

●勾配降下法(Gradient descent)○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一つ○ηは学習率と呼ばれるハイパーパラメータでモデルのパラメータの収束しやすさを調整○参考URL: https://qiita.com/masatomix/items/d4e5fb3b52fa4c92366f●なぜ必要か？○[線形回帰モデル(最小2乗法)] ▸MSEのパラメータに関する微分が0になる値を解析に求めることが可能○[ロジスティック回帰モデル(最尤法)] ▸対数尤度関数をパラメータで微分して0になる値を求める必要があるのだが、解析的にこの値を求めることは困難である。

●パラメータが更新されなくなった場合、それは勾配が0になったということ。少なくとも反復学習で探索した範囲では最適な解がもとめられたことになる。●勾配降下法では、パラメータを更新するのにN個全てのデータに対する和を求める必要がある。○nが巨大になった時にデータをオンメモリに載せる容量が足りない、計算時間が莫大になるなどの問題がある○確率的勾配降下法を利用して解決

●確率的勾配降下法(SGD)○データを一つずつランダムに(「確率的」に)選んでパラメータを更新○勾配降下法でパラメータを1回更新するのと同じ計算量でパラメータをn回更新できるので効率よく最適な解を探索可能

iv.モデルの評価

●分類の評価方法○「正解率」がよく使われる○正解した数/ 予測対象となった全データ数■メールのスパム分類●スパム数が80件・普通のメールが20件であった場合●全てをスパムとする分類器の正解率は80%となる。○どのような問題があるか？■分類したいクラスにはそれぞれ偏りがあることが多い■この場合、単純な正解率はあまり意味をなさないことがほとんどです。

●分類の評価方法○表情から怪しい人物を検知する動画分析ソリューション■前スライドの正解率が適当ではない例▸リコールやプレシジョンを使って評価する■異常値検知のタスクぽいが、説明の簡易化のため分類で解くとする

●False Positive○正常な人を間違えて異常としてしまう。○赤い枠が増えるので、確認の工数がかかかる。●False Negative○異常な人を間違えて正常としてしまう。○本当に検知したい脅威が見えなくなる。

●再現率(Recall)○「本当にPositiveなもの」の中からPositiveと予測できる割合(Negativeなものを Positiveとしてしまう事象については考えていない)○「誤り(False Positive)が多少多くても抜け漏れは少ない」予測をしたい際に利用○使用例) 病気の検診で「陽性であるものを陰性と誤診(False Negative)」としてしまうのを避けたい。「陰性を陽性であると誤診(False Positive)」とするものが少し増えたとしても再検査すればい

●適合率(Precision)○モデルが「Positiveと予測」したものの中で本当にPositiveである割合(本当にPositiveなものをNegativeとしてしまう子については考えていない)○見逃し(False Negative)が多くてもより正確な」予測をしたい際に利用○「重要なメールをスパムメールと誤判別」されるより「スパムと予測したものが確実にスパム」である方が便利。スパムメールを検出できなくても(False Negative)、自分でやればいい。

★
1.ハンズオン
a.タイタニックデータ解析
b.特徴量抽出

```
titanic_df.head(5)
```

	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	Cumings, 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

```
titanic_df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

#一部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

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

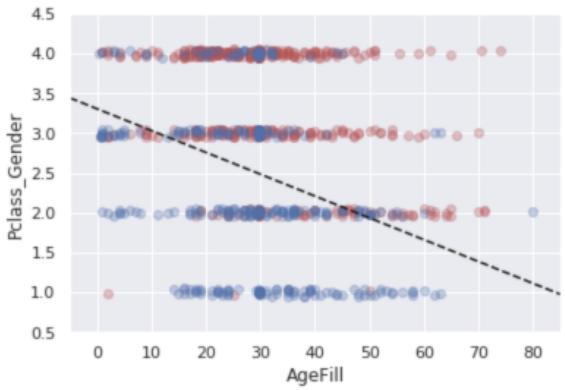
	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
5	0	3	male	NaN	0	0	8.4583	Q
17	1	2	male	NaN	0	0	13.0000	S
19	1	3	female	NaN	0	0	7.2250	C
26	0	3	male	NaN	0	0	7.2250	C
28	1	3	female	NaN	0	0	7.8792	Q
29	0	3	male	NaN	0	0	7.8958	S
31	1	1	female	NaN	1	0	146.5208	C
32	1	3	female	NaN	0	0	7.7500	Q
36	1	3	male	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	Pclass	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
26	0	3	male	NaN	0	0	7.2250	C	29.699118
28	1	3	female	NaN	0	0	7.8792	Q	29.699118
29	0	3	male	NaN	0	0	7.8958	S	29.699118

```
h = 0.02
xmin, xmax = -5, 85
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)

sc = ax.scatter(titanic_df.loc[index_survived, 'AgeFill'],
                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)
sc = ax.scatter(titanic_df.loc[index_notsurvived, 'AgeFill'],
                titanic_df.loc[index_notsurvived, 'Pclass_Gender']+(np.random.rand(len(index_notsurvived))-0.5)*0.1,
                color='b', label='Survived', alpha=0.3)

ax.set_xlabel('AgeFill')
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--')
```

```
[<matplotlib.lines.Line2D at 0x7f9596400f98>]
```



主成分分析

a.アウトライン

●多変量データの持つ構造をより少数個の指標に圧縮

○変量の個数を減らすことに伴う、情報の損失は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(2・3次元の場合)が実現可能

b.数学的定式化

i.データ形式の説明

●係数ベクトルが変われば線形変換後の値が変化

○情報の量を分散の大きさと捉える

○線形変換後の変数の分散が最大となる射影軸を探索

ii.主成分分析の説明

●以下の制約付き最適化問題を解く

○ノルムが1となる制約を入れる(制約を入れないと無限に解がある)

●ラグランジュ関数を微分して最適解を求める

○元のデータの分散共分散行列の固有値と固有ベクトルが、上記の制約付き最適化問題の解となる

●分散共分散行列は正定値対称行列▸固有値は必ず0以上・固有ベクトルは直行

iii.主成分と寄与率

寄与率〇第1~元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時、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

○寄与率：第k主成分の分散の全分散に対する割合(第k主成分が持つ情報量の割合)〇累積寄与率：第1-k主成分まで圧縮した際の情報損失量の割合

1.ハンズオン

a.乳がんデータの分析

```
cancer df shape: (569, 33)
```

```
[ ] 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	peri
842302	M	17.990	10.38	122.80	1001.0	0.11840	0.27760	0.300100	0.147100	0.2419	0.07871	1.0950	0.9053	
842517	M	20.570	17.77	132.90	1326.0	0.08474	0.07864	0.086900	0.070170	0.1812	0.05667	0.5435	0.7339	
84300903	M	19.690	21.25	130.00	1203.0	0.10960	0.15990	0.197400	0.127900	0.2069	0.05999	0.7456	0.7869	
84348301	M	11.420	20.38	77.58	386.1	0.14250	0.28390	0.241400	0.105200	0.2597	0.09744	0.4956	1.1560	
84358402	M	20.290	14.34	135.10	1297.0	0.10030	0.13280	0.198000	0.104300	0.1809	0.05883	0.7572	0.7813	
843786	M	12.450	15.70	82.57	477.1	0.12780	0.17000	0.157800	0.080890	0.2087	0.07613	0.3345	0.8902	
844359	M	18.250	19.98	119.60	1040.0	0.09463	0.10900	0.112700	0.074000	0.1794	0.05742	0.4467	0.7732	
84458202	M	13.710	20.83	90.20	577.9	0.11890	0.16450	0.093660	0.059850	0.2196	0.07451	0.5835	1.3770	
844981	M	13.000	21.82	87.50	519.8	0.12730	0.19320	0.185900	0.093530	0.2350	0.07389	0.3063	1.0020	
84501001	M	12.460	24.04	83.97	475.9	0.11860	0.23960	0.227300	0.085430	0.2030	0.08243	0.2976	1.5990	
845636	M	16.020	23.24	102.70	797.8	0.08206	0.06669	0.032990	0.033230	0.1528	0.05697	0.3795	1.1870	
84610002	M	15.780	17.89	103.60	781.0	0.09710	0.12920	0.099540	0.066060	0.1842	0.06082	0.5058	0.9849	
846226	M	19.170	24.80	132.40	1123.0	0.09740	0.24580	0.206500	0.111800	0.2397	0.07800	0.9555	3.5680	
846381	M	15.850	23.95	103.70	782.7	0.08401	0.10020	0.099380	0.053640	0.1847	0.05338	0.4033	1.0780	
84667401	M	13.730	22.61	93.60	578.3	0.11310	0.22930	0.212800	0.080250	0.2069	0.07682	0.2121	1.1690	
84799002	M	14.540	27.54	96.73	658.8	0.11390	0.15950	0.163900	0.073640	0.2303	0.07077	0.3700	1.0330	
848406	M	14.680	20.13	94.74	684.5	0.09867	0.07200	0.073950	0.052590	0.1586	0.05922	0.4727	1.2400	
84862001	M	16.130	20.68	108.10	798.8	0.11700	0.20220	0.172200	0.102800	0.2164	0.07356	0.5692	1.0730	
849014	M	19.810	22.15	130.00	1260.0	0.09831	0.10270	0.147900	0.094980	0.1582	0.05395	0.7582	1.0170	
8510426	B	13.540	14.36	87.46	566.3	0.09779	0.08129	0.066640	0.047810	0.1885	0.05766	0.2699	0.7886	
8510653	B	13.080	15.71	85.63	520.0	0.10750	0.12700	0.045680	0.031100	0.1967	0.06811	0.1852	0.7477	
8510824	B	9.504	12.44	60.34	273.9	0.10240	0.06492	0.029560	0.020760	0.1815	0.06905	0.2773	0.9768	
8511133	M	15.340	14.26	102.50	704.4	0.10730	0.21350	0.207700	0.097560	0.2521	0.07032	0.4388	0.7096	
851509	M	21.160	23.04	137.20	1404.0	0.09428	0.10220	0.109700	0.086320	0.1769	0.05278	0.6917	1.1270	
852552	M	16.650	21.38	110.00	904.6	0.11210	0.14570	0.152500	0.091700	0.1995	0.06330	0.8068	0.9017	
852631	M	17.140	16.40	116.00	912.7	0.11860	0.22760	0.222900	0.140100	0.3040	0.07413	1.0460	0.9760	
852763	M	14.580	21.53	97.41	644.8	0.10540	0.18680	0.142500	0.087830	0.2252	0.06924	0.2545	0.9832	
852781	M	18.610	20.25	122.10	1094.0	0.09440	0.10660	0.149000	0.077310	0.1697	0.05699	0.8529	1.8490	
852973	M	15.300	25.27	102.40	732.4	0.10820	0.16970	0.168300	0.087510	0.1926	0.06540	0.4390	1.0120	
853201	M	17.570	15.05	115.00	955.1	0.09847	0.11570	0.098750	0.079530	0.1739	0.06149	0.6003	0.8225	

```
[ ] cancer_df.drop('Unnamed: 32', axis=1, inplace=True)
    cancer_df
```

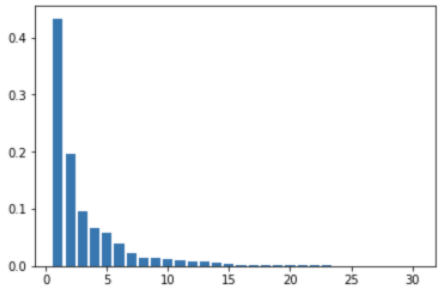
```
X = cancer_df.loc[:, 'radius_mean':]
```

```
# 学習用とテスト用でデータを分離
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(cv=10, 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('Confustion matrix:\n{}'.format(confusion_matrix(y_true=y_test, y_pred=logistic.predict(X_test_scaled))))
```

・検証スコア97%で分類できることを確認

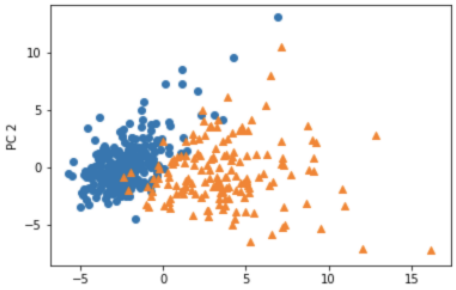
```
[ ] pca = PCA(n_components=30)
    pca.fit(X_train_scaled)
    plt.bar([n for n in range(1, len(pca.explained_variance_ratio_)+1)], pca.explained_variance_ratio_)
```

```
<BarContainer object of 30 artists>
```



```
[ ] # PCA
    # 次元数2まで圧縮
    pca = PCA(n_components=2)
    X_train_pca = pca.fit_transform(X_train_scaled)
    print('X_train_pca shape: {}'.format(X_train_pca.shape))
    # X_train_pca shape: (426, 2)

    # 寄与率
    print('explained variance ratio: {}'.format(pca.explained_variance_ratio_))
    # explained variance ratio: [ 0.43315126  0.19586506]

    # 散布図にプロット
    temp = pd.DataFrame(X_train_pca)
    temp['Outcome'] = y_train.values
    b = temp[temp['Outcome'] == 0]
    m = temp[temp['Outcome'] == 1]
    plt.scatter(x=b[0], y=b[1], marker='o') # 良性は○でマーク
    plt.scatter(x=m[0], y=m[1], marker='^') # 悪性は△でマーク
    plt.xlabel('PC 1') # 第1主成分をx軸
    plt.ylabel('PC 2') # 第2主成分をy軸
```

```
X_train_pca shape: (426, 2)
explained variance ratio: [0.43315126 0.19586506]
Text(0, 0.5, 'PC 2')
```



X軸は-10～20の乱数

Y軸は-10～15の乱数

アルゴリズム

k平均法(k-means)のアルゴリズム

a.アウトライン

k近傍法説明

●分類問題のための機械学習手法

●kを変化させると結果も変わる

●kを大きくすると決定境界は滑らかになる

b.数学的定式化

i.データ形式の説明

ii.k平均法の説明

●教師なし学習●クラスタリング手法●与えられたデータをk個のクラスタに分類する

iii.アルゴリズム

1) 各クラスタ中心の初期値を設定する

2) 各データ点に対して、各クラスタ中心との距離を計算し、最も距離が近いクラスタを割り当てる

3) 各クラスタの平均ベクトル（中心）を計算する

4) 収束するまで2, 3の処理を繰り返す

★

1.ハンズオン

a.syntheticデータ分析

課題〇人口データと分類結果をプロットしてください

k近傍法(kNN)データセット名 人口データレコード数-カラム数-詳細-備考

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

from sklearn.cluster import KMeans
```

```
wine = datasets.load_wine()
```

```
X = wine.data
```

```
X.shape
```

```
(178, 13)
```

```
y=wine.target
```

```
y.shape
```

```
(178,)
```

```
wine.target_names
```

```
array(['class_0', 'class_1', 'class_2'], dtype='<U7')
```

```
model = KMeans(n_clusters=3)
```

```
labels = model.fit_predict(X)
```

```
df = pd.DataFrame({'labels': labels})
type(df)
```

```
pandas.core.frame.DataFrame
```

```
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]
```

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

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

species	class_0	class_1	class_2
labels			
0	13	20	29

サポートベクターマシン

Python 3 エンジニア認定データ分析試験公式教材「Pythonによるあたらしいデータ分析の教科書」
https://www.amazon.co.jp/dp/4798158348/ より

サポートベクターマシンは、分類・回帰だけでなく、外れ値検出にも使えるアルゴリズム。

分類に用いるサポートベクターマシンは直線や平面など運類できないデータを高次元の空間に写して線形分離することにより、分類を行うアルゴリズム。

実際は、高次元の空間に写すのではなく、データ間の近さを定量化するカーネルを導入する。

サポートベクタマシンのアルゴリズムの導入的な説明。

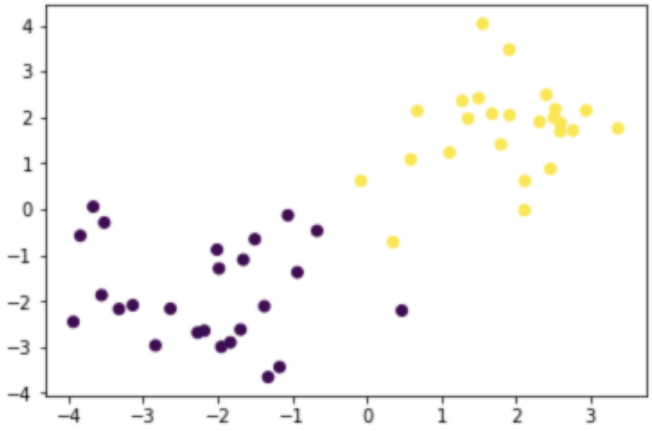
データ間が最も大きくなるような直線（決定境界）を引く。

各クラスのデータをサポートベクタ、そしてクラス間のサポートベクタの距離をマージンと呼ぶ。マージンを最大することにより決定境界を求める。

```
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])
    ys_train = np.concatenate([np.zeros(25), np.ones(25)]).astype(np.int)
    return X_train, ys_train
```

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

```
<matplotlib.collections.PathCollection at 0x7f49d31a42e8>
```



```
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

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)
```

```
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()
```

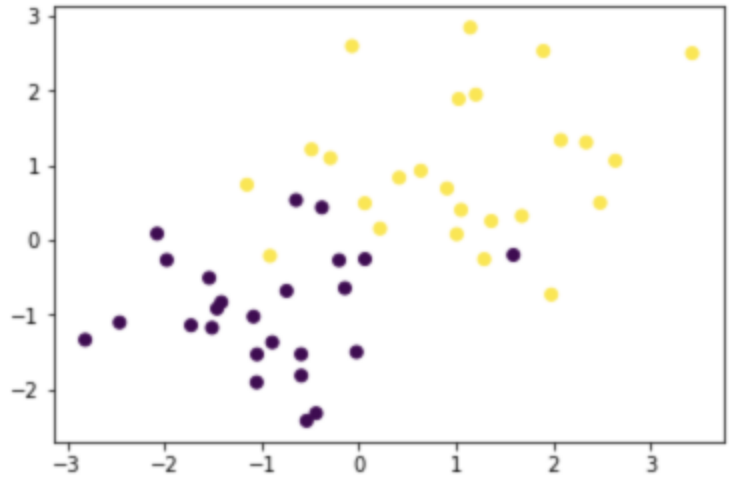
```
xx0, xx1 = np.meshgrid(np.linspace(-1.5, 1.5, 100), np.linspace(-1.5, 1.5, 100))
xx = np.array([xx0, xx1]).reshape(2, -1).T

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 * rbf(X_test[i], sv)
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, facecolors='none', edgecolors='k')
# 領域を可視化
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, linestyles=['--', '-', '--'])
```

```
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)
```

```
<matplotlib.collections.PathCollection at 0x7f49d2fb38d0>
```



オッカムの髭剃り。

シンプルさを保つのは本当に難しいことです。一種のエントロピーの法則とも言えるかもしれません。

理論はできるだけ簡潔であるべきだ