13（2）モデルの適合度と複雑性

花澤楓 学籍番号: 2125242

2023/12/10

1　情報量とモデル適合度の尺度

1.1　情報の量をどうやって測るか

情報の量を測る尺度として、シャノンの情報量（Shannon's information）がある。これは、情報の量を確率の対数で測るものである。例えば、コインを投げて表が出る確率が 0.5 であるとき、表が出たという情報の量は、$-\log_2 0.5 = 1$ となる。これは、表が出る確率が 0.5 であることを知ることで、情報の量が 1 ビット増えたことを意味する。また、コインを投げて表が出る確率が 0.25 であるとき、表が出たという情報の量は、$-\log_2 0.25 = 2$ となる。これは、表が出る確率が 0.25 であることを知ることで、情報の量が 2 ビット増えたことを意味する。このように、情報の量は、確率の対数で測ることができる。シャノン情報量を S とすると、S は以下のように定義される。

$$S = -\ln P(E)$$
$$ES = -E\ln P(E) = -\sum P(E)\ln P(E)$$

例えば、天気 $\in${雨, 晴}だとすると、$(P(\text{天気} = \text{雨}), P(\text{天気} = \text{晴})) = (0.5, 0.5)$ の時に最もシャノン情報量が多くなる。シャノン情報量においては「情報とは不確実性の解決である」としているためである。

log を取る理由：

確率の持つ性質、独立事象の加法性があることから ln を取ると便利なためである。つまり、

$$-\ln P(E_1 \cap E_2) = -\ln P(E_1) - \ln P(E_2)$$

が成り立つため。

1.2　情報量基準（information criteria）

情報量基準とは、以下のように定義される。

$$\underbrace{-E^X \ln f(X)}_{\text{未知のデータへの誤差}} = \underbrace{-\frac{\sum_i^N \ln f(X_i)}{N}}_{\text{既知のデータへの誤差}} + \underbrace{k\frac{C_n}{N}}_{\text{罰則項}}$$

モデルのパラメータが増えるほど、情報量基準としては優れたものではないと判断される。ここで、

- $C_n = 1$：Akaike IC(1973), $AIC = -2\sum \ln f(X_i) + 2k$
- $C_n = \ln N$：Bayesian IC(1978), $BIC = -2\sum \ln f(X_i) + k \ln N$

2 再標本抽出法（resampling method）とモデル適合度の測定

再標本抽出法とは数値計算＋ランダム化を用いて、データの性質を調べる方法である。再標本抽出法には、以下のようなものがある。

2.1 ブートストラップ法（boot strap）Efron(1979)

ブートストラップ法とは、統計量の標本分布を推定するために、N の標本集団から N の標本を復元抽出する方法（i.e., 重複を許して resampling する）。ブートストラップの利点は大きく 2 つある。

1. 漸近的なリファインメント（refinement）
   これは、あるクラスの統計量について、ブートストラップによる近似の方が漸近分布による近似よりも精度の高い近似を与えることである。例えば、標準正規分布よりも、ブートストラップによって得られる近似分布の方が t 統計量の厳密分布の良い近似になる。
2. 分布を理論的に求める必要がない
   漸近分布を導出することが難しい統計量や、既知の漸近分布を持たない統計量についても、コンピュータを用いたシミュレーションによる分布を求めることができる。

ブートストラップとは、ブーツのつまみの部分であり、「自力で成し遂げる（pull oneself up by one's bootstrap）」という意味もある。このブートストラップという名前は、与えられた標本をもとに自らのコピーを作り出す様を表現している。

例えば、手元の N の標本から平均を計算し、それを再標本抽出によって複数回繰り返すことで標本平均の分布を求めることができる。同様に、標本分散と標準誤差を求めることができる。resample の回数としては 1000-100000 回程度が一般的である。

また、再標本抽出によって何 % の標本を使っているのかについて計算することができる。あるサンプルをとってきて、そのサンプルが選ばれない確率は $1 - 1/N$ であり、それが N 回繰り返されるため、N を大きくすると $(1 - 1/N)^N \to 1/e$ となる。よって、何 % の標本を使うのかについては、$1 - 1/e = 0.63$ 程度のサンプルを使用していることがわかる。

2.2 ジャックナイフ法（jack knife）Quenouille(1949)

ジャックナイフ法とは、統計量の標本分布を推定するために、N の標本集団から N-1 の標本を非復元抽出する方法（i.e., 重複を許さずに resampling する）。ジャックナイフ法は、ブートストラップ法と比べて、計算量が少なくて済むという利点がある。ジャックナイフ法は、ブートストラップ法の特別な場合として捉えることができる。

2.3 交差検証法（cross validation; CV）

交差検証法とは、汎化誤差を測定するために、標本をランダムに k 分割し、(k-1) の訓練集合（training set）としてモデルを推定（学習）し、残り 1 つの検証集合（testing set）でそのモデルの妥当性を検証すること。k=1 がスタンダード。データが特に少ない時には、データ数を N とした時に k=N とすることがある。この得 sy な場合を leave-one-out method（1 個抜き法）と呼ぶ。

以下では、多項式回帰に L2 正則化を適用し、CV によって最適なハイパーパラメータの値を求める。L2 生息かとは、誤差関数を $E(x)$ とした時に、パラメータの 2 乗和を誤差関数に加えたもので、

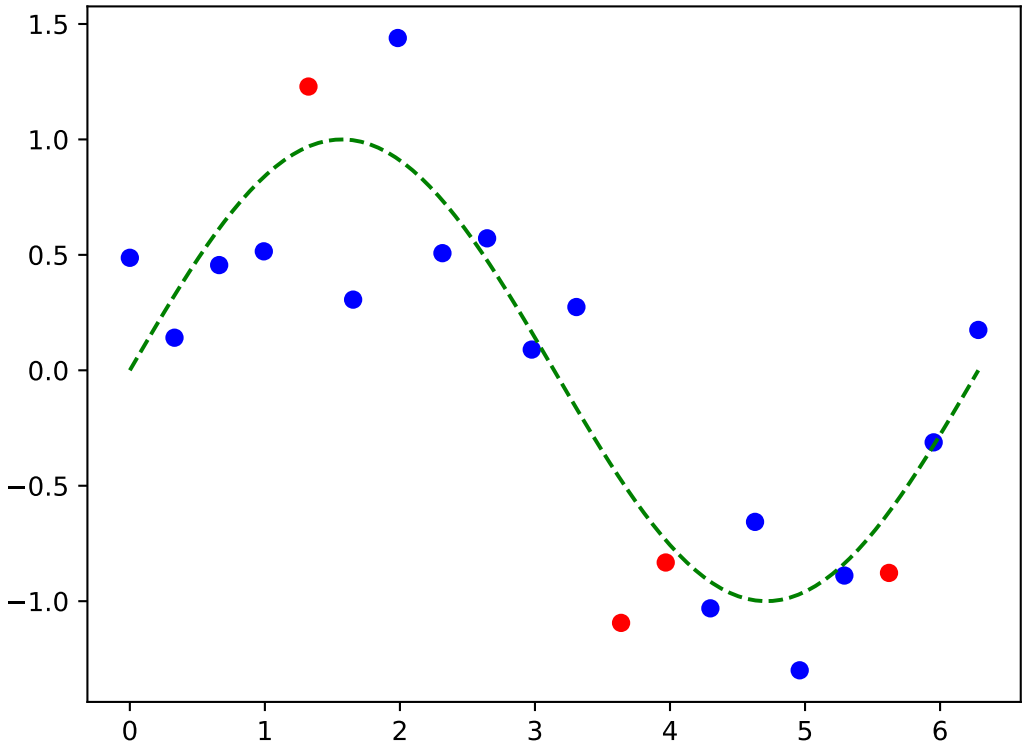
$$\tilde{E}(x) = E(x) + \lambda \sum_{i=1}^{N} x_i^2$$

で定義される。ここで、λ はハイパーパラメータであり、この誤差関数を最小にするように CV によって λ を求める。まず、多項式の次数とハイパーパラメータの値によって当てはまりがどうなるかをプロットする。（R ではうまくいかなかったので Python で実行。参考 url：https://colab.research.google.com/drive/1kiiNeVw3SAc9br5LgE8QPmbuvJ2LvKcj?usp=sharing#scrollTo=keEsvghkaiVb）

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

# sin(x) にホワイトノイズを加えたデータ
def create_dataset(data_num, sigma=0.3):
    X = np.linspace(0, 2*np.pi, num=data_num)
    t = np.sin(X) + sigma * np.random.randn(data_num)
    return X, t

# t: 正解値，y: 予測値
def RMSE(t, y):
    return np.sqrt(np.mean((y - t)**2))

def R2(t, y):
    return 1 - np.sum((y - t)**2) / np.sum((t - np.mean(t))**2)

def MAE(t, y):
    return np.mean(np.abs(y - t))

# リッジ回帰
class LinearRegression(object):
    # モデルの訓練
```

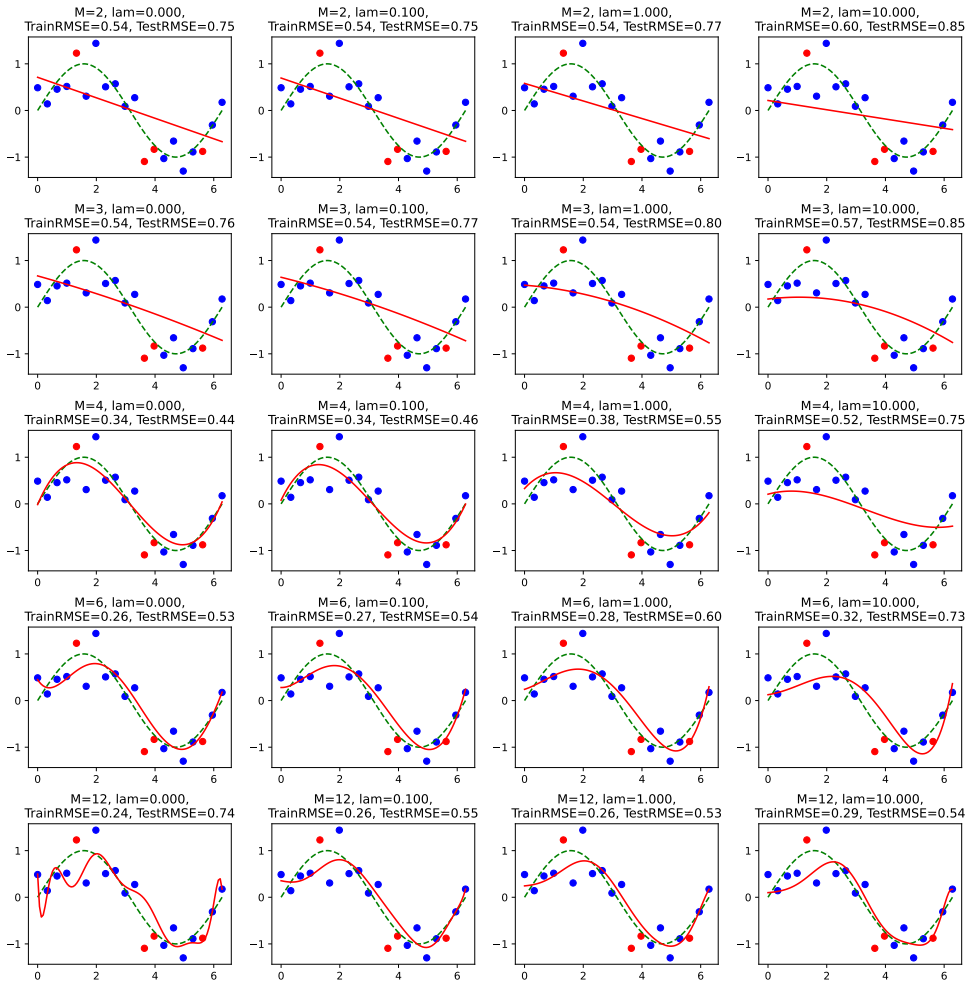
```
    # X: 学習データの入力変数
    # t: 学習データの目的変数
    # lam: 正則化係数
    def fit(self, X, t, lam=0.):
        d = X.shape[1]
        XX = np.dot(X.T, X)
        # numpy.linalg.solve で解く
        self.w = np.linalg.solve(XX + lam * np.identity(d), np.dot(X.T, t))

    # 新しいデータの予測
    # X: 入力変数
    def predict(self, X):
        return np.dot(X, self.w)

# 多項式
# x^0 から x^(M-1) までの基底関数の値を返す
class Polynomial(object):
    def __init__(self, M=1):
        self.M = M

    # X は（データ数 x 1）もしくは（データ数）のサイズの配列
    def __call__(self, X):
        return np.array([X.flatten()**i for i in range(self.M)]).T


# 乱数シードの固定
np.random.seed(1)

# データの作成
N = 20
orgX, org_t = create_dataset(N)

index = np.arange(N)
np.random.shuffle(index)

# 訓練データ（8割を訓練データに）
X_train, t_train = orgX[index[:int(0.8*N)]], org_t[index[:int(0.8*N)]]
# テスト用データ（残りの2割）
X_test, t_test = orgX[index[int(0.8*N):]], org_t[index[int(0.8*N):]]
```

```
# 訓練用データのプロット
plt.scatter(X_train, t_train, marker='o', color='blue', label=None)
# テスト用データのプロット
plt.scatter(X_test, t_test, marker='o', color='red', label=None)
# 真の曲線を表示
XX, tt = create_dataset(100, sigma=0.)
plt.plot(XX, tt, color='green', linestyle='--')
plt.show()
```




```
# モデルの学習と評価
lams = [0.0, 0.1, 1.0, 10.0]  # 正則化係数
Ms = [2, 3, 4, 6, 12]  # 次数

plt.clf()
fig = plt.figure(figsize=(16, 16))
plt.subplots_adjust(hspace=0.4)
```

```
i = 0
for M in Ms:
    for lam in lams:
        # リッジ回帰モデルの学習
        Phi = Polynomial(M=M)
        lr_model = LinearRegression()
        lr_model.fit(Phi(X_train), t_train, lam=lam)

        # 予測曲線の取得
        XX, tt = create_dataset(100, sigma=0.)
        yy = lr_model.predict(Phi(XX))

        # 予測値の計算
        y_train = lr_model.predict(Phi(X_train))
        y_test = lr_model.predict(Phi(X_test))

        # プロット
        subplt = fig.add_subplot(len(Ms), len(lams), i+1)
        subplt.set_title('M=%d, lam=%.3f,\nTrainRMSE=%.2f, TestRMSE=%.2f'%(M, lam, RMSE(t_train, y_t
        subplt.scatter(X_train, t_train, marker='o', color='blue', label=None)
        subplt.scatter(X_test, t_test, marker='o', color='red', label=None)
        subplt.plot(XX, tt, color='green', linestyle='--')
        subplt.plot(XX, yy, color='red')  # 予測曲線を表示

        i += 1

plt.show()
```


M=2, lam=0.000,
TrainRMSE=0.54, TestRMSE=0.75
M=2, lam=0.100,
TrainRMSE=0.54, TestRMSE=0.75
M=2, lam=1.000,
TrainRMSE=0.54, TestRMSE=0.77
M=2, lam=10.000,
TrainRMSE=0.60, TestRMSE=0.85
M=3, lam=0.000,
TrainRMSE=0.54, TestRMSE=0.76
M=3, lam=0.100,
TrainRMSE=0.54, TestRMSE=0.77
M=3, lam=1.000,
TrainRMSE=0.54, TestRMSE=0.80
M=3, lam=10.000,
TrainRMSE=0.57, TestRMSE=0.85
M=4, lam=0.000,
TrainRMSE=0.34, TestRMSE=0.44
M=4, lam=0.100,
TrainRMSE=0.34, TestRMSE=0.46
M=4, lam=1.000,
TrainRMSE=0.38, TestRMSE=0.55
M=4, lam=10.000,
TrainRMSE=0.52, TestRMSE=0.75
M=6, lam=0.000,
TrainRMSE=0.26, TestRMSE=0.53
M=6, lam=0.100,
TrainRMSE=0.27, TestRMSE=0.54
M=6, lam=1.000,
TrainRMSE=0.28, TestRMSE=0.60
M=6, lam=10.000,
TrainRMSE=0.32, TestRMSE=0.73
M=12, lam=0.000,
TrainRMSE=0.24, TestRMSE=0.74
M=12, lam=0.100,
TrainRMSE=0.26, TestRMSE=0.55
M=12, lam=1.000,
TrainRMSE=0.26, TestRMSE=0.53
M=12, lam=10.000,
TrainRMSE=0.29, TestRMSE=0.54

次に、交差検証を行う。

```
from sklearn.model_selection import KFold
# 交差検証（訓練データの中で CV を行う）
kf = KFold(n_splits=5, shuffle=True, random_state=0)

optM, optlam = 0, 0
meanRMSE = np.inf

for M in Ms:
    for lam in lams:
        rmse = []
        for i, (train, valid) in enumerate(kf.split(X_train)):
            # リッジ回帰モデルの学習
            Phi = Polynomial(M=M)
            lr_model = LinearRegression()
            lr_model.fit(Phi(X_train[train]), t_train[train], lam=lam)
```

```
            # Validation データの予測値の計算
            y_valid = lr_model.predict(Phi(X_train[valid]))

            # 評価指標を計算
            rmse.append(RMSE(t_train[valid], y_valid))
        print('M=%d, lam=%.3f, Average RMSE=%.3f (%s)'%(M, lam, np.mean(rmse), rmse))
        if meanRMSE > np.mean(rmse):
            meanRMSE = np.mean(rmse)
            optM, optlam = M, lam
```

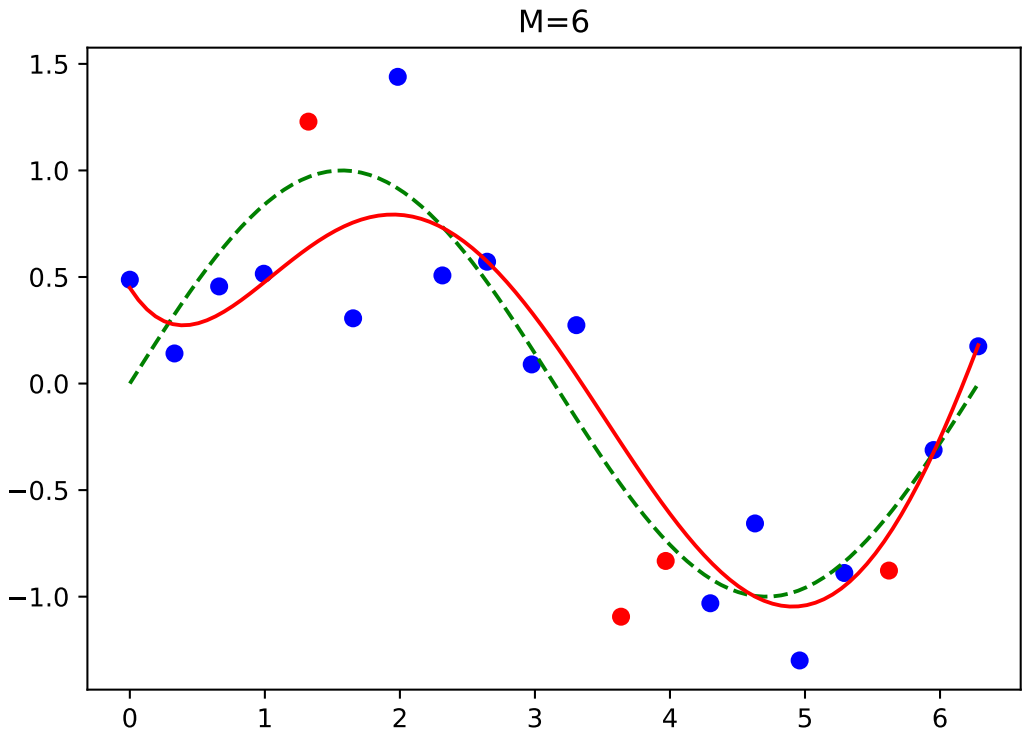
```
## M=2, lam=0.000, Average RMSE=0.616 ([0.8917778035276837, 0.22849345038755897, 0.8652156243292713,
## M=2, lam=0.100, Average RMSE=0.612 ([0.8935907398347092, 0.22379597256476902, 0.8409231634878787,
## M=2, lam=1.000, Average RMSE=0.591 ([0.9070108572446542, 0.2288040052463807, 0.688427210563779, 0
## M=2, lam=10.000, Average RMSE=0.613 ([0.9490664011173215, 0.4160310460489965, 0.45671969392481043
## M=3, lam=0.000, Average RMSE=0.656 ([0.9453491060120585, 0.26031355599000994, 0.9524883477399454,
## M=3, lam=0.100, Average RMSE=0.652 ([0.9486776694412627, 0.2355377865196231, 0.950811776531517, 0
## M=3, lam=1.000, Average RMSE=0.646 ([0.9635132169003908, 0.1817975584849352, 0.9449783261361091,
## M=3, lam=10.000, Average RMSE=0.661 ([0.9430752245194763, 0.321314384815722, 0.8452036254977439,
## M=4, lam=0.000, Average RMSE=0.512 ([0.4715758345006626, 0.91574014601184, 0.3912454283849275, 0.
## M=4, lam=0.100, Average RMSE=0.484 ([0.5260144737861943, 0.640608399679838, 0.4337090601738509, 0
## M=4, lam=1.000, Average RMSE=0.496 ([0.7147342826089119, 0.22177356910004437, 0.5658325877676497,
## M=4, lam=10.000, Average RMSE=0.667 ([0.9543383549523166, 0.3351012073468777, 0.8356250246225639,
## M=6, lam=0.000, Average RMSE=0.347 ([0.25166889110368007, 0.6165430222683065, 0.11863138076546643
## M=6, lam=0.100, Average RMSE=0.390 ([0.3143380712285073, 0.5770281086196116, 0.29703330363437663,
## M=6, lam=1.000, Average RMSE=0.381 ([0.36022437516227956, 0.36404502035066216, 0.3640063329146814
## M=6, lam=10.000, Average RMSE=0.428 ([0.43214422938449726, 0.2428808729946182, 0.501072195300815,
## M=12, lam=0.000, Average RMSE=22.771 ([7.822480294148158, 4.032453875054537, 84.14482234146978, 1
## M=12, lam=0.100, Average RMSE=3.043 ([4.814685355179716, 0.674286576233139, 3.792233784985932, 5.
## M=12, lam=1.000, Average RMSE=2.412 ([3.5002687902306104, 0.5802850262571948, 4.3930526674253025,
## M=12, lam=10.000, Average RMSE=2.002 ([2.2120893399999586, 0.4501499990929838, 4.9613100061297795
```

```
print('opt M=%d, opt lam=%.3f'%(optM, optlam))
```

```
## opt M=6, opt lam=0.000
```

```
M = optM
lam = optlam

# 線形回帰モデルの学習（訓練データ全てを使う）
```

```
Phi = Polynomial(M=M)
lr_model = LinearRegression()
lr_model.fit(Phi(X_train), t_train)

# 予測曲線の取得
XX, tt = create_dataset(100, sigma=0.)
yy = lr_model.predict(Phi(XX))

# プロット
plt.title('M=%d'%(M))
plt.scatter(X_train, t_train, marker='o', color='blue', label=None)
plt.scatter(X_test, t_test, marker='o', color='red', label=None)
plt.plot(XX, tt, color='green', linestyle='--')
plt.plot(XX, yy, color='red')  # 予測曲線を表示
plt.show()
```




```
# 評価指標の表示
y_train = lr_model.predict(Phi(X_train))
```

```
y_test = lr_model.predict(Phi(X_test))
print('（学習データ）RMSE: %.3f, MAE: %.3f, R2: %.3f'%(RMSE(t_train, y_train), MAE(t_train, y_train),
```

```
## （学習データ）RMSE: 0.256, MAE: 0.189, R2: 0.862
```

```
print('（テストデータ）RMSE: %.3f, MAE: %.3f, R2: %.3f'%(RMSE(t_test, y_test), MAE(t_test, y_test), R2
```

```
## （テストデータ）RMSE: 0.525, MAE: 0.456, R2: 0.689
```

CV の結果、次数は 6、ハイパーパラメータの値は 0 が最も当てはまりが良いことがわかった。