正則化

縮小推定

- 前章までのパラメータ推定法
  - 最小二乗法、最尤推定、etc.
  - モデルがデータにできるだけフィットするようにパラメータを最適化
- 縮小推定（shrinkage method）
  - 各パラメータの絶対値が大きくならないようにする
  - 正則化（regularization）とも呼ばれる
  - 代表的な縮小推定手法
    - リッジ回帰（ridge regression）
    - Lasso (the Least absolute shrinkage and selection operator)

リッジ回帰

- 最小二乗法による線形回帰のパラメータ推定では

$$\mathrm{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2$$

を最小化するようにパラメータ $\beta_0, \beta_1, \ldots, \beta_p$ を決定

- 問題点
  - 学習データへのフィッティングしか考慮していないので過学習が起きやすい
  - p が n より大きい場合、解が一意に決まらない

リッジ回帰

- リッジ回帰（ridge regression）の目的関数

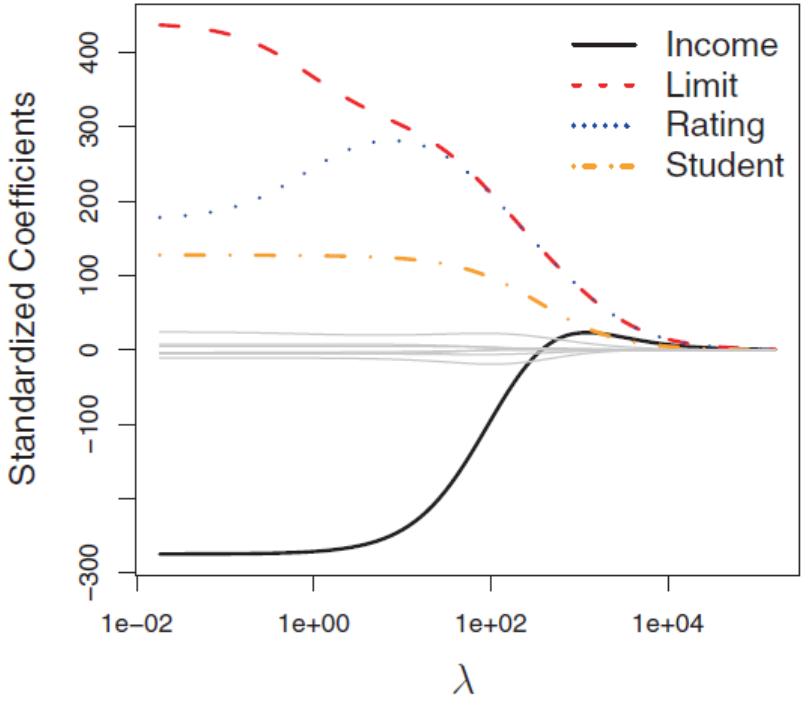
$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\beta_j^2$$

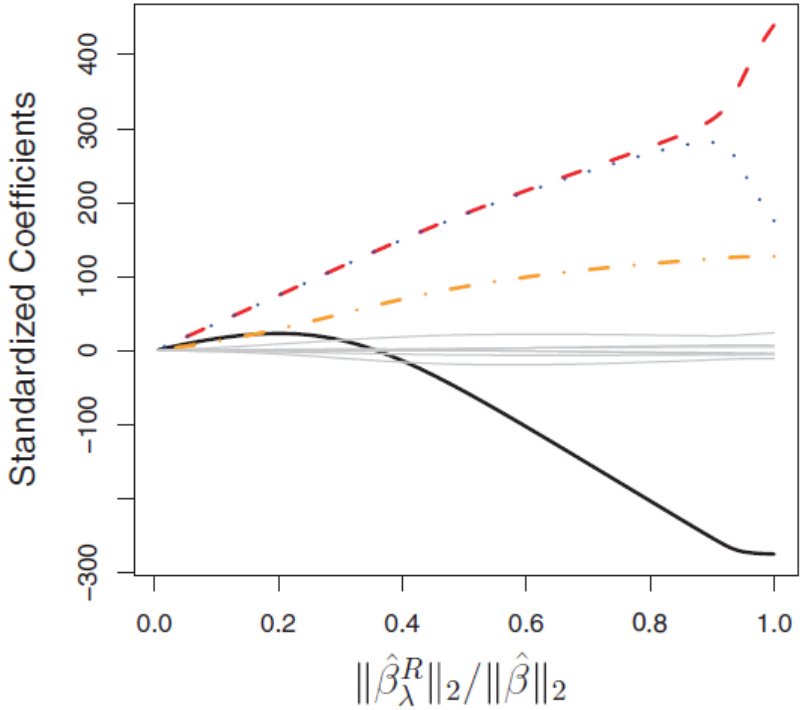
パラメータの絶対値が大きくなることに対するペナルティ（L2正則化項）

- λ: チューニングパラメータ（ハイパーパラメータ）
  - λ がゼロであれば最小二乗法と同じ
  - λ を大きくすると $\beta_1, \ldots, \beta_p$ はゼロに近づく
    - β_0 にはペナルティがつかないことに注意
  - 交差検証などによって適切な値に決める

リッジ回帰

- Credit データセット
  - 残高を予測





λ を大きくするとパラメータの値が小さくなる

$\hat{\beta}_\lambda^R$ ：リッジ回帰のパラメータベクトル

$\|\beta\|_2 = \sqrt{\sum_{j=1}^{p} \beta_j^2}$ ・・・β の $\ell 2$ノルム

リッジ回帰

- 予測変数のスケール
  - 普通の最小二乗法の場合は、予測変数のスケールを変えてもモデルの性能は変わらない
    - 予測変数 x_j の値を c 倍すると β_j が $1/c$ になるだけ
  - リッジ回帰の場合は性能に大きな影響
    - 値の大きな予測変数が大きな影響力を持つ

- 前処理として予測変数の標準化を行う

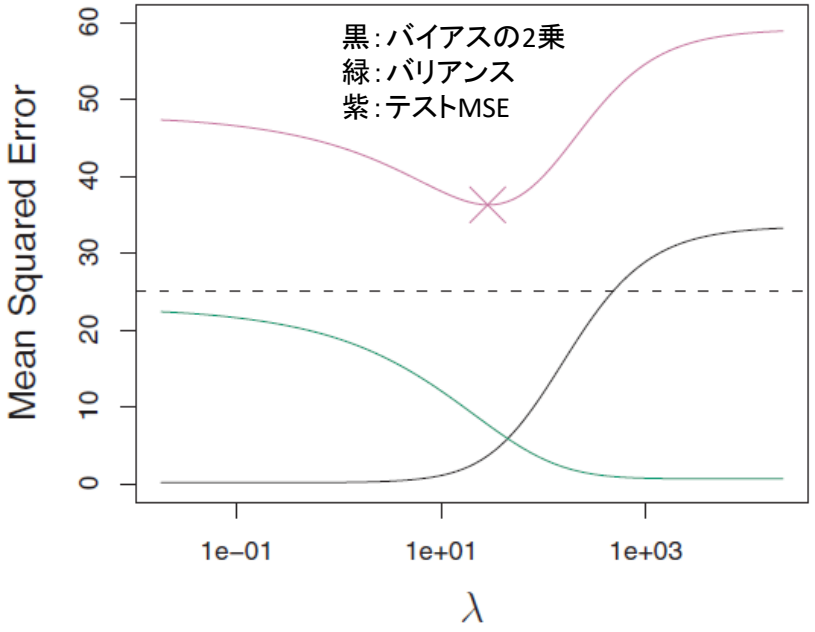
$$\tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_j}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(x_{ij} - \bar{x}_j\right)^2}}$$

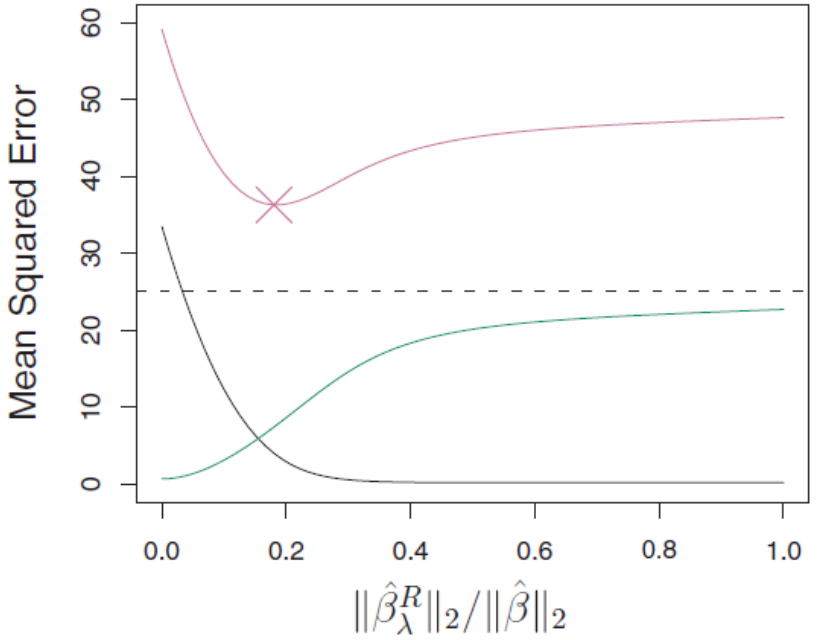
リッジ回帰

- 人工データ（$p = 45$, $n = 50$）

Bias-variance trade-off（再掲）

$$E\left[\left(y_0 - \hat{f}(x_0)\right)^2\right] = \mathrm{Var}\left(\hat{f}(x_0)\right) + \left[\mathrm{Bias}\left(\hat{f}(x_0)\right)\right]^2 + \mathrm{Var}(\varepsilon)$$





Lasso

- Lasso (the Least absolute shrinkage and selection operator) の目的関数

$$\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2+\lambda\sum_{j=1}^{p}\left|\beta_j\right|$$

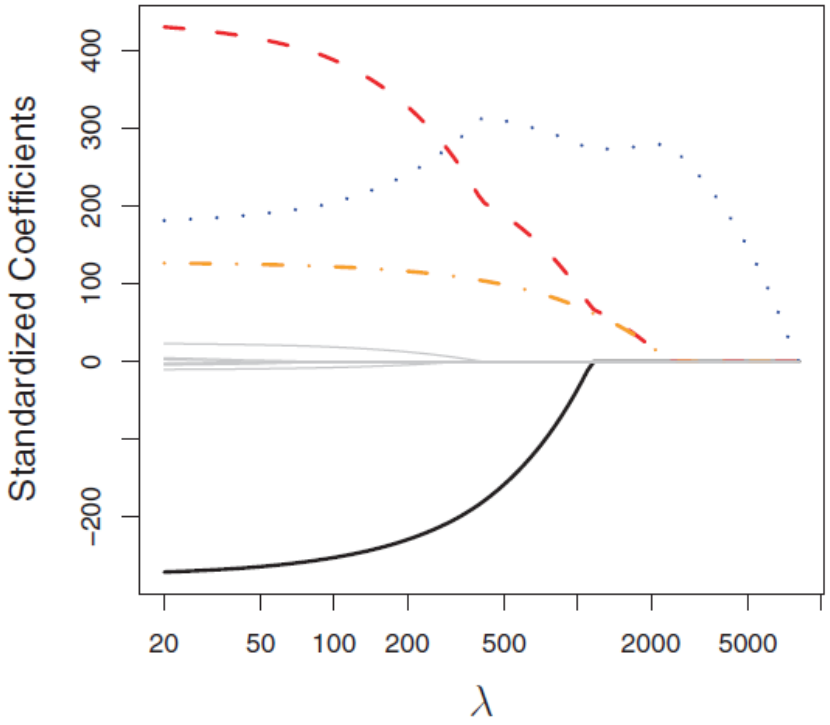
L1正則化項

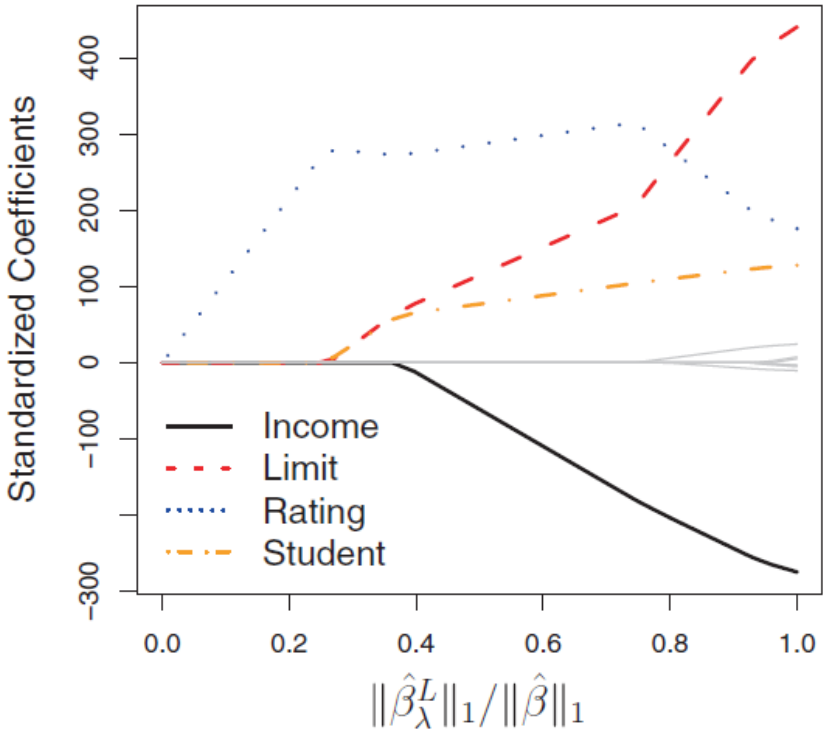
- 性質
  - パラメータのL1ノルムに比例するペナルティ
  - λを大きくすると多くのパラメータがゼロになる

    → 変数選択（variable selection）によってスパース（sparse）なモデルを生成

Lasso

- Credit データセット





λ が大きくなるにしたがって
使われる予測変数が減る

$\hat{\beta}_\lambda^L$ ：Lasso のパラメータベクトル

$\|\beta\|_1 = \sum_{j=1}^{p} |\beta_j|$ ・・・ β の $\ell 1$ノルム

縮小推定

- Lasso とリッジ回帰の別な定式化
  - Lasso

$$\underset{\beta}{\text{minimize}}\left\{\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2\right\} \quad \text{subject to} \quad \sum_{j=1}^{p}\left|\beta_j\right| \le s$$

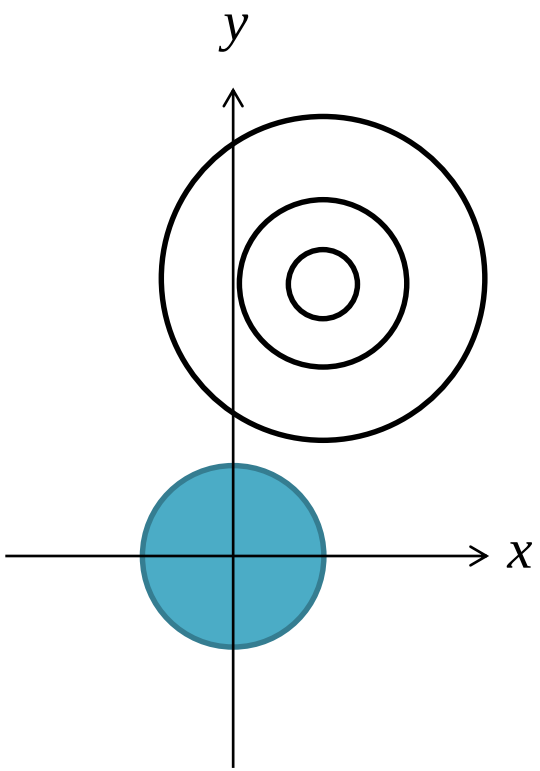
  - リッジ回帰

$$\underset{\beta}{\text{minimize}}\left\{\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2\right\} \quad \text{subject to} \quad \sum_{j=1}^{p}\beta_j^2 \le s$$

参考

不等式制約付きの最適化

- 例



$$f(x,y)=(x-1)^2+(y-4)^2$$

$$g(x,y)=x^2+y^2-1$$

$$\underset{x,y}{\text{minimize}}\ f(x,y) \quad \text{subject to} \quad g(x,y)\le 0$$

制約なしで解いた場合の解 $(x, y) = (1, 4)$ が制約を満たさないので等式制約を考えると

$$\nabla f(x,y)=-\lambda\nabla g(x,y)$$

$$g(x,y)=0$$

ラグランジュの未定乗数法

を解くことになるが、これは上式を満たす $\lambda > 0$ のもとで、以下の制約なし最小化問題を解くのと同じ

$$L=f(x,y)+\lambda g(x,y)$$

$$\nabla L=\nabla f(x,y)+\lambda\nabla g(x,y)=0$$

縮小推定

- 最良変数選択
  - ゼロでない変数が最大 *s* 個という制約のもとで RSS を最小化

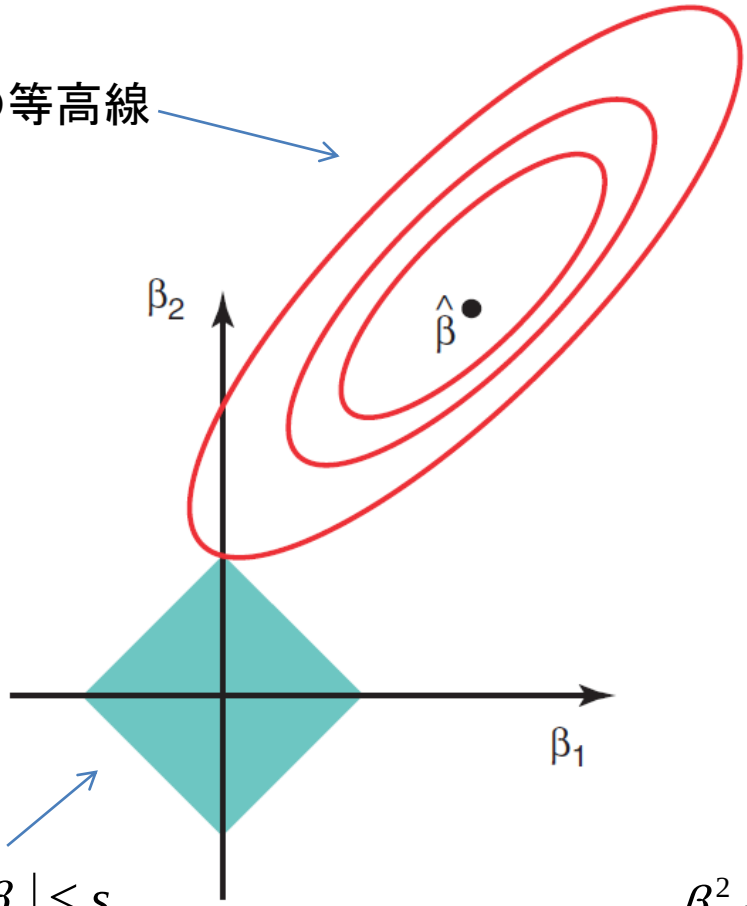
$$\underset{\beta}{\text{minimize}}\left\{\sum_{i=1}^{n}\left(y_i-\beta_0-\sum_{j=1}^{p}\beta_j x_{ij}\right)^2\right\} \quad \text{subject to} \quad I\left(\beta_j \neq 0\right) \leq s$$

  - 実際にこの基準でパラメータを求めるのは困難
    - リッジ回帰や Lasso と異なり勾配法が使えない

縮小推定

Lasso

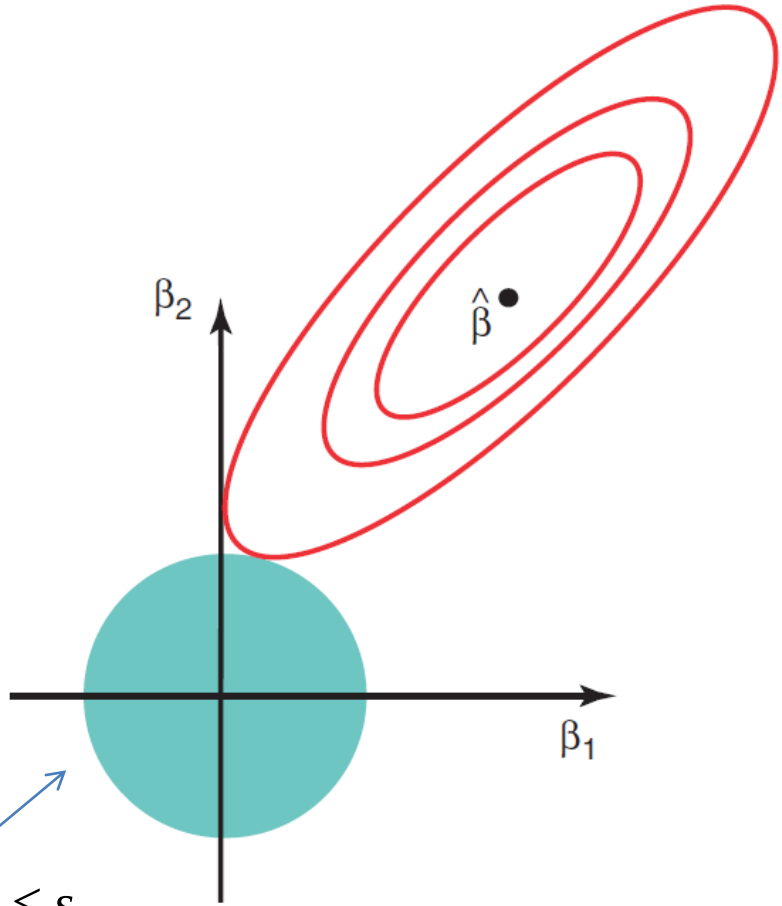
RSSの等高線



$|\beta_1| + |\beta_2| \leq s$

→ β_1がゼロになる

リッジ回帰



$\beta_1^2 + \beta_2^2 \leq s$

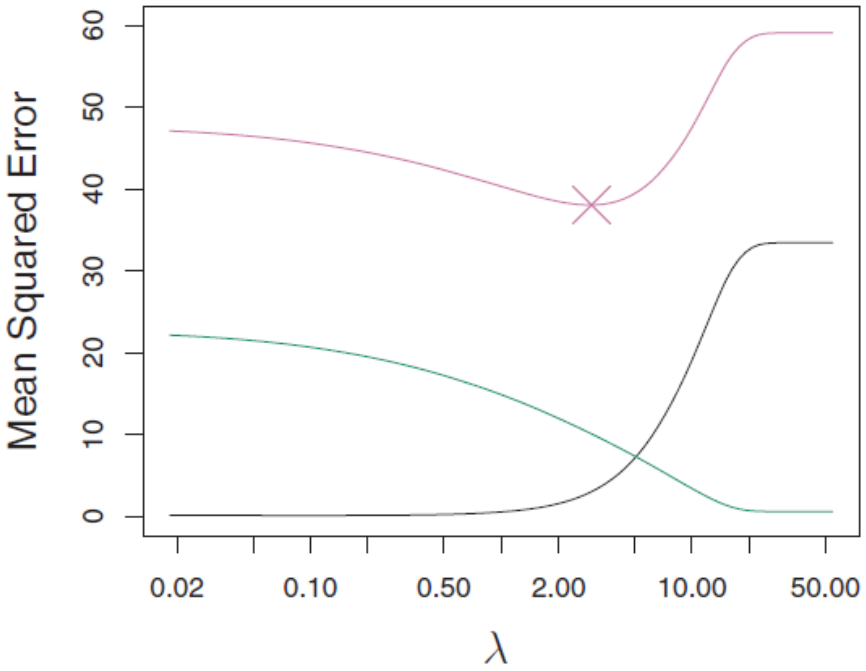
→ β_1もβ_2もゼロにならない

Lasso vs リッジ回帰

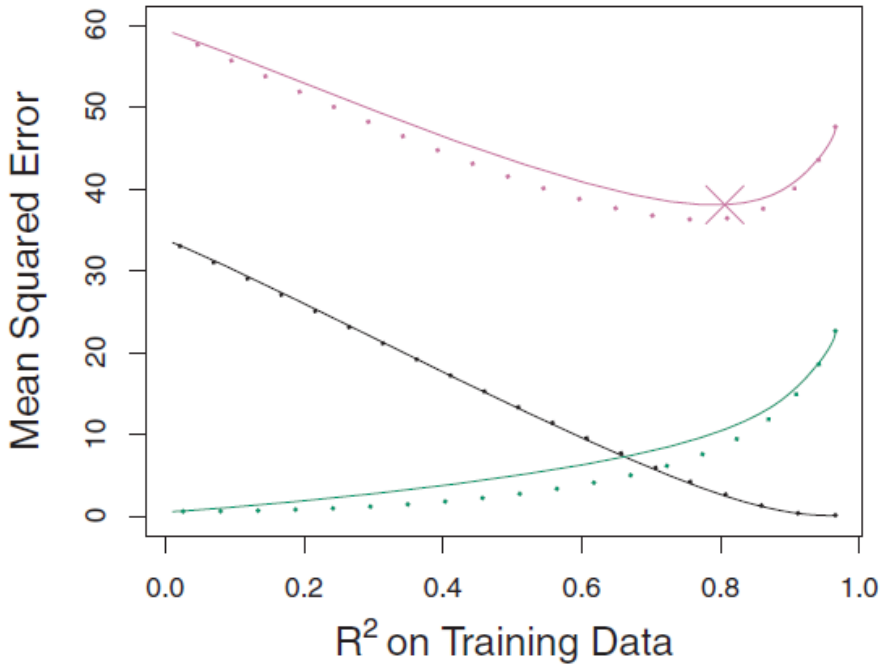
- 人工データ（$p = 45$, $n = 50$）

Lasso

黒：バイアスの2乗
緑：バリアンス
紫：テストMSE



実線：Lasso
点線：リッジ回帰

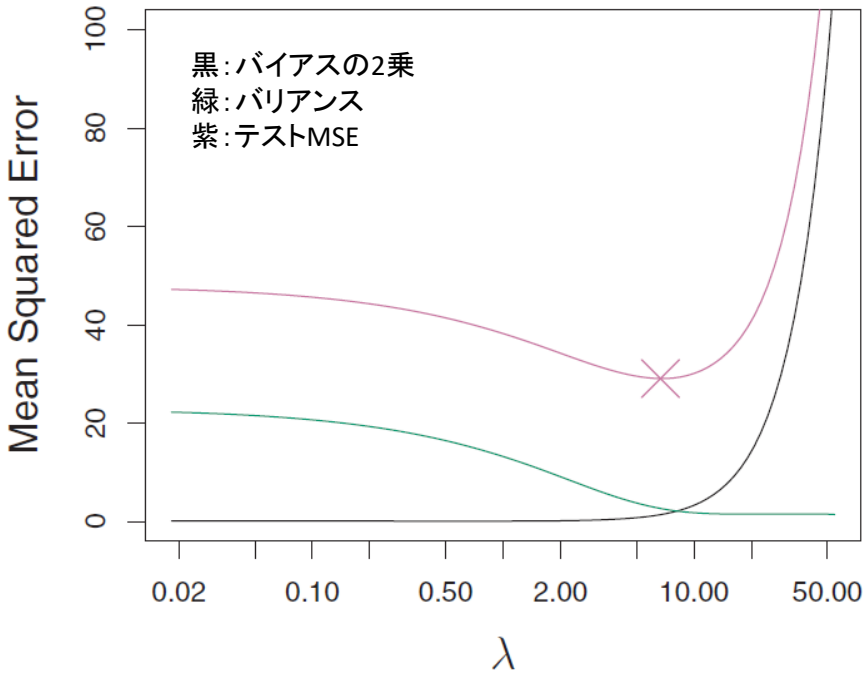


リッジ回帰の方が少し精度が良い

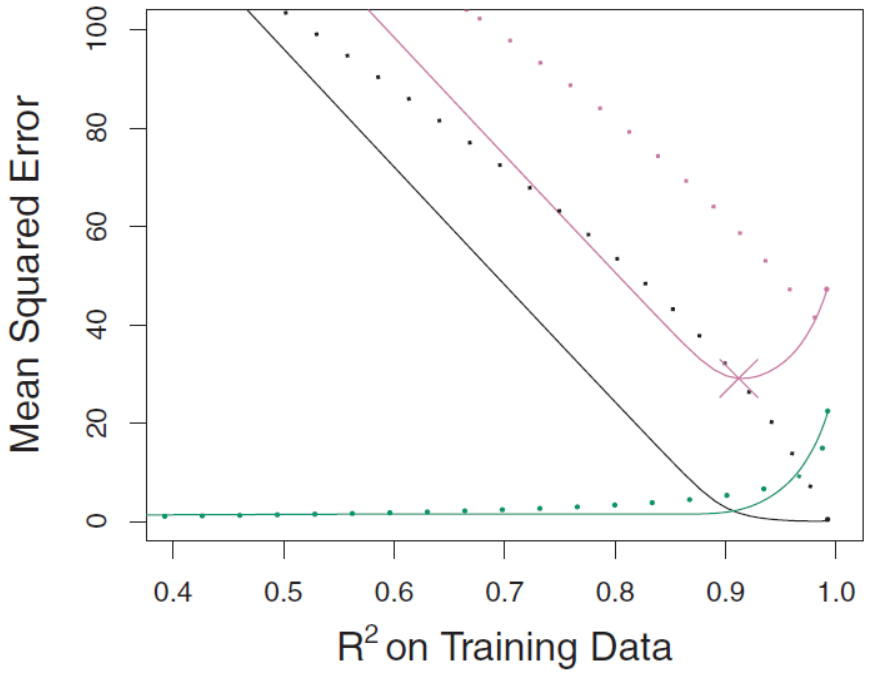
Lasso vs リッジ回帰

- 別な人工データ（$p = 45, n = 50$）
  - 応答が45個の予測変数のうち2個の予測変数で決まる場合

Lasso



実線：Lasso
点線：リッジ回帰



Lasso の方が少し精度が良い

Python実習

- Hitters データセットの読み込み
  - ダウンロード
    - サンプルファイル　Hitters.csv
      - https://www.logos.t.u-tokyo.ac.jp/~tsuruoka/lecture/sml/Hitters.csv
    - 上記ファイルをホームディレクトリに保存（Windowsの場合）
  - ホームディレクトリに移動

```
>>> import os
>>> os.chdir(os.path.expanduser("~"))
```

  - `pandas.read_csv()` 関数でデータフレームとして読み込み

```
>>> import pandas as pd
>>> Hitters = pd.read_csv('Hitters.csv', header=0, na_values='NA')
```

Python実習

- Salary 情報がないデータを削除

```
>>> Hitters.head()
...
>>> Hitters.shape
(322, 20)
>>> Hitters = Hitters.dropna().reset_index(drop=True)
>>> Hitters.shape
(263, 20)
```

Python実習

- 入力値の行列、出力値のベクトルを作成
  - 質的変数をダミー変数に変換

```
>>> y = Hitters.Salary
>>> dummies = pd.get_dummies(Hitters[['League', 'Division',
'NewLeague']])
>>> dummies.head()
>>> Xd = Hitters.drop(['Salary', 'League', 'Division', 'NewLeague'],
axis = 1)
>>> X = pd.concat([Xd, dummies[['League_A', 'Division_E',
'NewLeague_A']]], axis=1)
```

Python実習

- リッジ回帰
  - Ridge()
    - 引数 alpha: 式中のλに相当

```
>>> from sklearn.linear_model import Ridge
>>> ridge = Ridge(alpha = 10, normalize = True)
>>> ridge.fit(X, y)
>>> pd.Series(ridge.coef_, index=X.columns)
AtBat            0.069853
Hits             0.273515
HmRun            0.952606
...
Division_E      14.283610
NewLeague_A     -1.484429
dtype: float64
>>> ridge.intercept_
278.60251474651693
```

Python実習

- リッジ回帰
  - 様々なλで学習してパラメータを保存

```
>>> import numpy as np
>>> alphas = 10**np.linspace(10, -2, 100)
>>> alphas
>>> coeffs = []
>>> intercepts = []
>>> for a in alphas:
...     ridge.set_params(alpha=a)
...     ridge.fit(X, y)
...     coeffs.append(ridge.coef_)
...     intercepts.append(ridge.intercept_)
```

Python実習

- リッジ回帰
  - 学習したモデルのパラメータ
    - $\lambda \approx 11{,}498$ のモデル

```
>>> alphas[49]
11497.569953977356
>>> pd.Series(coeffs[49], index=X.columns)
...
>>> intercepts[49]
535.5453647543033
>>> import math
>>> math.sqrt(sum(coeffs[49]**2))
0.015604015364701398                    ← L2ノルム
```

Python実習

- リッジ回帰
  - 学習したモデルのパラメータ
    - $\lambda \approx 705$ のモデル

```
>>> alphas[59]
705.4802310718645
>>> pd.Series(coeffs[59], index=X.columns)
...
>>> intercepts[59]
529.7820602618896
>>> math.sqrt(sum(coeffs[59]**2))
0.2532564951096926                    ← L2ノルム
```

Python実習

- リッジ回帰
  - λ とパラメータの関係をプロット

```
>>> import matplotlib.pyplot as plt
>>> ax = plt.gca()
>>> ax.plot(alphas, coeffs)
>>> ax.set_xscale('log')
>>> plt.axis('tight')
>>> plt.xlabel('alpha')
>>> plt.ylabel('weights')
>>> plt.show()
```

Python実習

- データの分割
  - train_test_split()

```
>>> from sklearn import model_selection
>>> X_train, X_test , y_train, y_test =
model_selection.train_test_split(X, y, test_size=0.5)
```

- 学習、評価（テストMSE）

```
>>> ridge = Ridge(normalize=True, alpha=4)
>>> ridge.fit(X_train, y_train)
>>> pd.Series(ridge.coef_, index=X.columns)
...
>>> pred = ridge.predict(X_test)
>>> from sklearn.metrics import mean_squared_error
>>> mean_squared_error(y_test, pred)
125865.9554098947
```

→ alpha を変えたらどうなるか？

Python実習

- 交差検証による λ のチューニング
  - RidgeCV()

```
>>> from sklearn.linear_model import RidgeCV
>>> ridgecv =  RidgeCV(alphas, normalize = True)
>>> ridgecv.fit(X_train, y_train)
...
>>> ridgecv.alpha_
0.869749002617834
```

Python実習

- ベストな λ での精度

```
>>> ridge_best = Ridge(alpha=ridgecv.alpha_, normalize=True)
>>> ridge_best.fit(X_train, y_train)
...
>>> mean_squared_error(y_test, ridge_best.predict(X_test))
106112.14133627179
>>> pd.Series(ridge_best.coef_, index=X.columns)
```

Python実習

- Lasso

```
>>> from sklearn.linear_model import Lasso, LassoCV
>>> lasso = Lasso(normalize=True)
```

- 交差検証による λ のチューニング

```
>>> lassocv = LassoCV(alphas=None, max_iter=1e5, normalize=True)
>>> lassocv.fit(X_train, y_train)
>>> lasso.set_params(alpha=lassocv.alpha_)
>>> lasso.fit(X_train, y_train)
>>> mean_squared_error(y_test, lasso.predict(X_test))
...
>>> pd.Series(lasso.coef_, index=X.columns)
...
```

いくつかのパラメータの値がゼロになっている

縮小推定（パラメータ計算の例）

- 簡単なパラメータ推定の例

$$n = p$$

データ行列

$$\mathbf{X} = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & & 0 \\ \vdots & & \ddots & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\beta_0 = 0$$

最小二乗法

$$L = \sum_{j=1}^{p} \left(y_j - \beta_j \right)^2$$

$$\frac{\partial L}{\partial \beta_i} = \frac{\partial}{\partial \beta_i} \sum_{j=1}^{p} \left(y_j - \beta_j \right)^2$$

$$= -2\left(y_i - \beta_i \right)$$

$$\frac{\partial L}{\partial \beta_i} = 0 \quad \text{より} \quad \hat{\beta}_i = y_i$$

縮小推定（パラメータ計算の例）

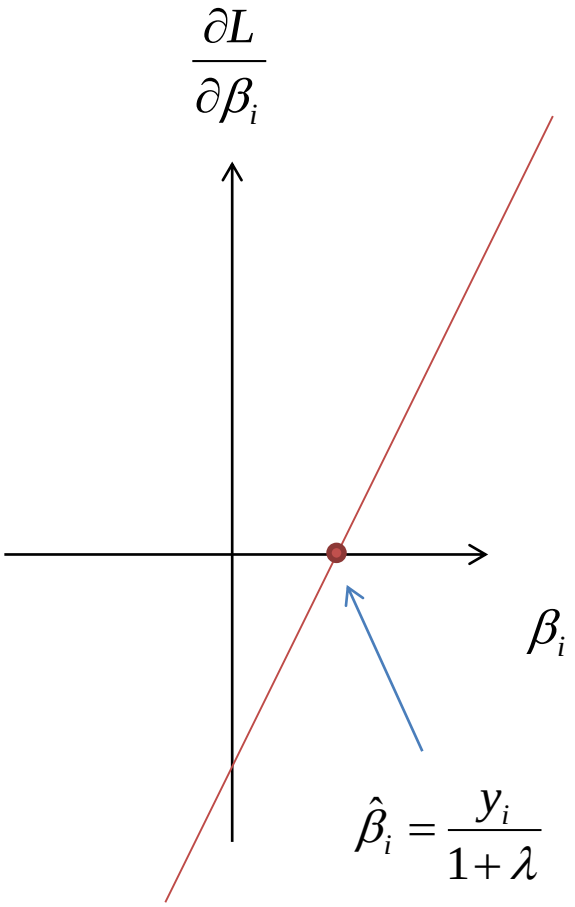
- リッジ回帰

$$L = \sum_{j=1}^{p} \left(y_j - \beta_j \right)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

$$\frac{\partial L}{\partial \beta_i} = -2\left(y_i - \beta_i \right) + 2\lambda\beta_i$$

$$= \left(2 + 2\lambda \right)\beta_i - 2y_i$$

$$\frac{\partial L}{\partial \beta_i} = 0 \quad \text{より} \quad \hat{\beta}_i^R = \frac{y_i}{1+\lambda}$$



縮小推定（パラメータ計算の例）

- Lasso

$$L = \sum_{j=1}^{p} \left(y_j - \beta_j \right)^2 + \lambda \sum_{j=1}^{p} \left| \beta_j \right|$$

$$\hat{\beta}_j^L = \begin{cases} y_j - \lambda/2 & \text{if } y_j > \lambda/2 \\ y_j + \lambda/2 & \text{if } y_j < -\lambda/2 \\ 0 & \text{if } \left| y_j \right| \leq \lambda/2 \end{cases}$$

参考

Lasso のパラメータ計算の例

- Lasso

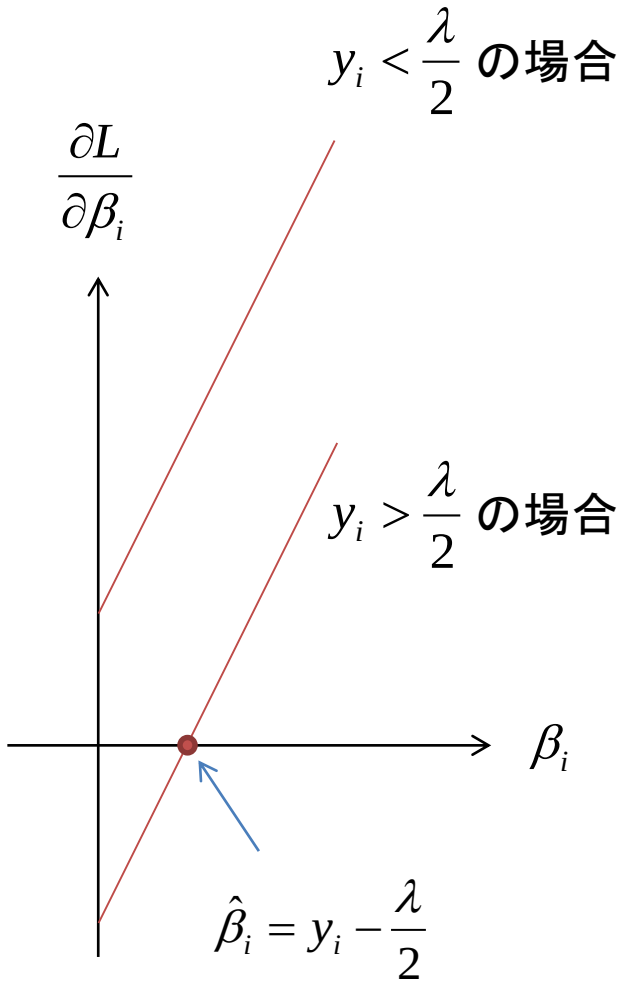
$$L = \sum_{j=1}^{p} \left(y_j - \beta_j \right)^2 + \lambda \sum_{j=1}^{p} \left| \beta_j \right|$$

$\beta_i > 0$ の領域

$$\frac{\partial L}{\partial \beta_i} = -2\left(y_i - \beta_i \right) + \lambda = 2\beta_i + \lambda - 2y_i$$

$y_i > \frac{\lambda}{2}$ の場合 $\frac{\partial L}{\partial \beta_i} = 0$ となるのは $\hat{\beta}_i = y_i - \frac{\lambda}{2}$

$y_i \leq \frac{\lambda}{2}$ の場合は $\frac{\partial L}{\partial \beta_i} > 0$



参考

Lasso のパラメータ計算の例

- Lasso

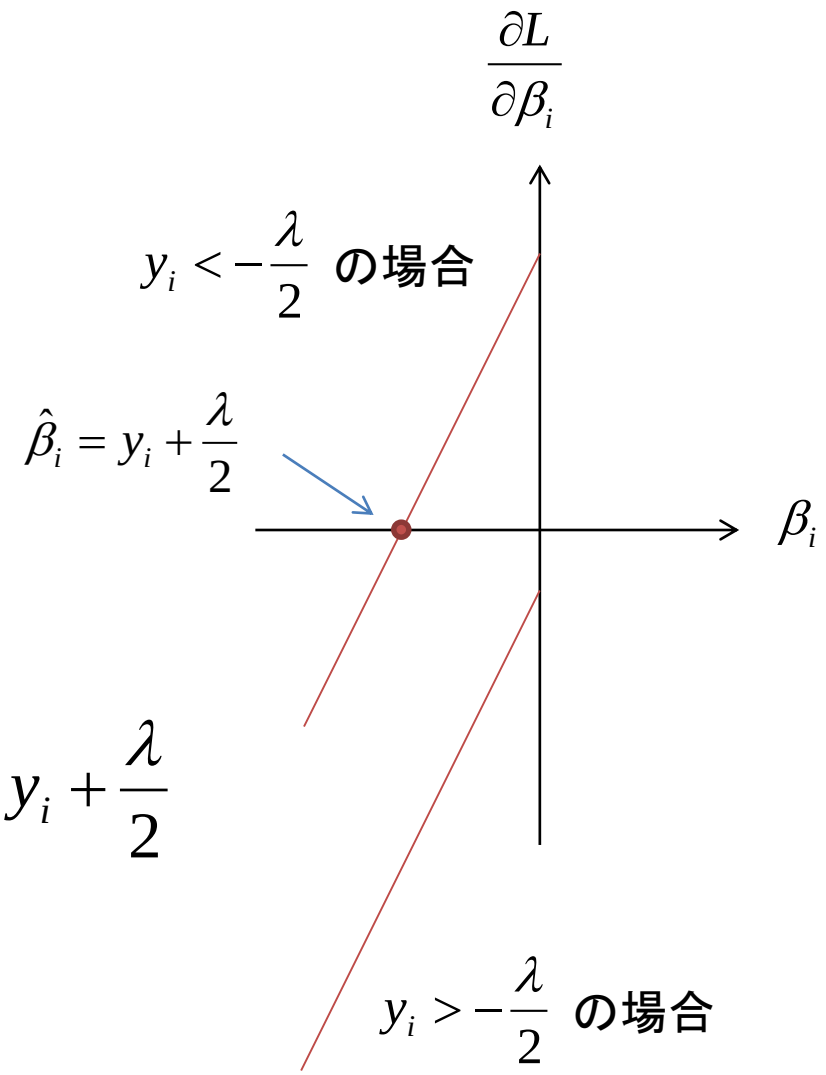
$$L=\sum_{j=1}^{p}\left(y_j-\beta_j\right)^2+\lambda\sum_{j=1}^{p}\left|\beta_j\right|$$

$\beta_i<0$ の領域

$$\frac{\partial L}{\partial \beta_i}=-2\left(y_i-\beta_i\right)-\lambda=2\beta_i-\lambda-2y_i$$

$y_i<-\frac{\lambda}{2}$ の場合 $\frac{\partial L}{\partial \beta_i}=0$ となるのは $\hat{\beta}_i=y_i+\frac{\lambda}{2}$

$y_i\geq-\frac{\lambda}{2}$ の場合は $\frac{\partial L}{\partial \beta_i}<0$



参考

Lasso のパラメータ計算の例

- Lasso

$$L = \sum_{j=1}^{p} \left(y_j - \beta_j \right)^2 + \lambda \sum_{j=1}^{p} \left| \beta_j \right|$$

$\left| y_i \right| \le \frac{\lambda}{2}$ の場合は $\frac{\partial L}{\partial \beta_i} < 0 \ \left(\beta_i < 0 \right), \frac{\partial L}{\partial \beta_i} > 0 \ \left(\beta_i > 0 \right)$ より

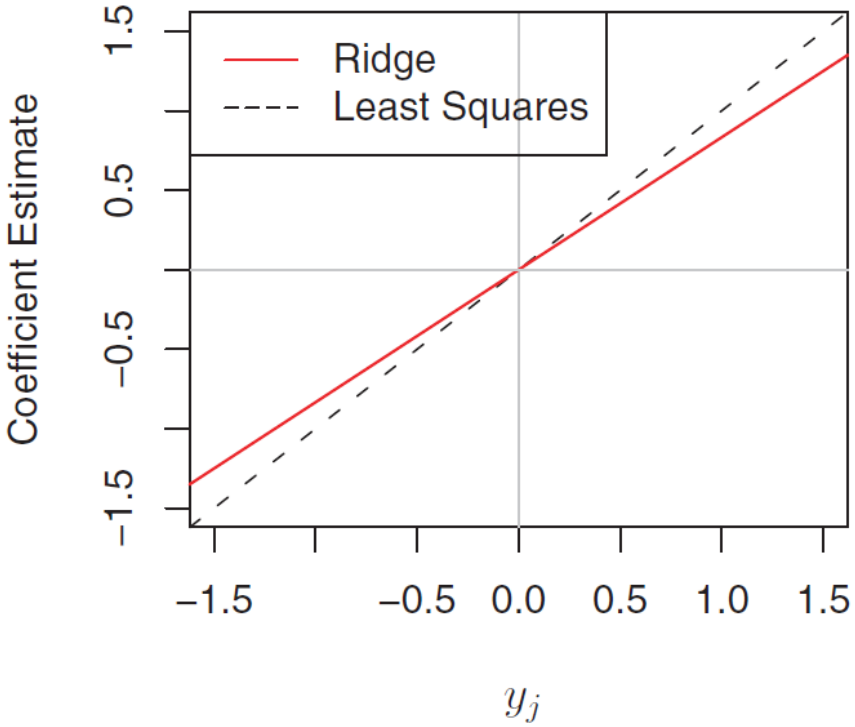
$\beta_i = 0$ で最小

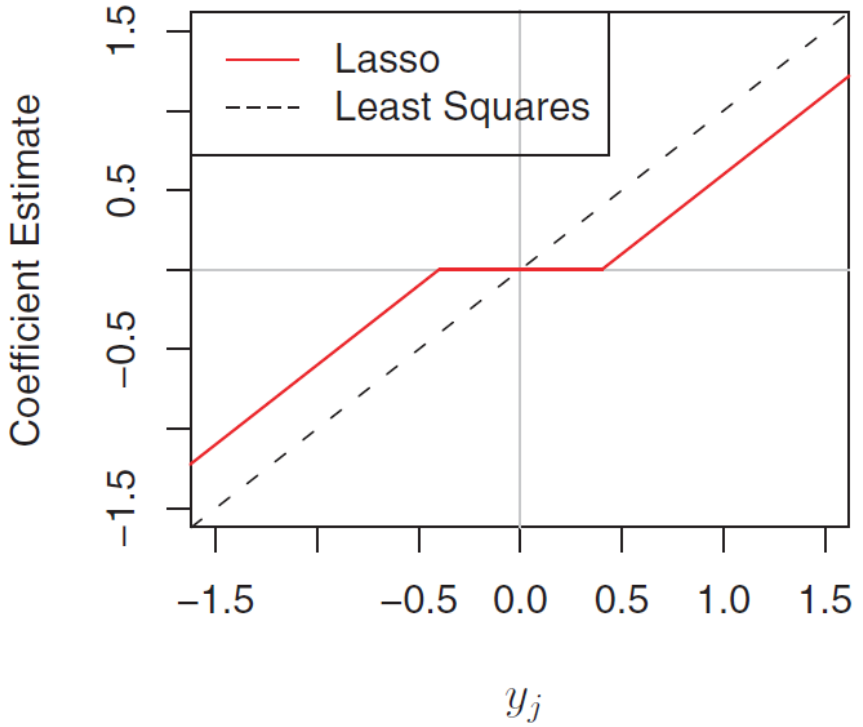
結局

$$\hat{\beta}_j^L = \begin{cases} y_j - \lambda/2 & \text{if } y_j > \lambda/2 \\ y_j + \lambda/2 & \text{if } y_j < -\lambda/2 \\ 0 & \text{if } \left| y_j \right| \le \lambda/2 \end{cases}$$

縮小推定（パラメータ計算の例）

- パラメータ推定の結果





$$\hat{\beta}_i^R = \frac{y_i}{1+\lambda}$$

$$\hat{\beta}_j^L = \begin{cases} y_j - \lambda/2 & \text{if } y_j > \lambda/2 \\ y_j + \lambda/2 & \text{if } y_j < -\lambda/2 \\ 0 & \text{if } |y_j| \le \lambda/2 \end{cases}$$

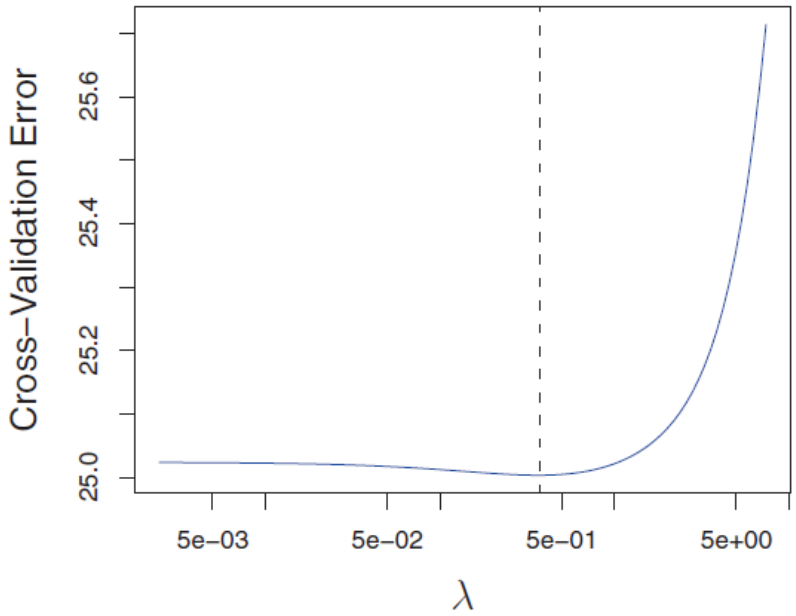
リッジ回帰は割り算で縮む

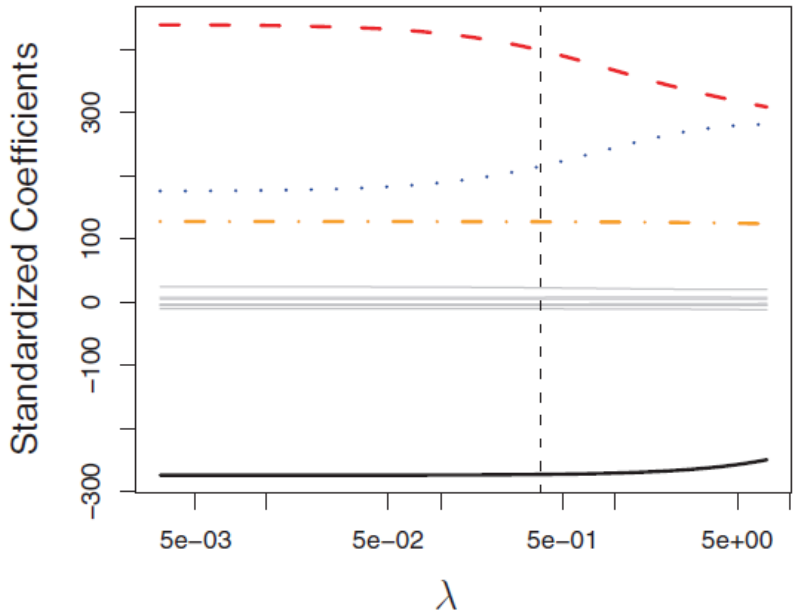
Lasso は引き算で縮む

縮小推定

- 交差検証による λ のチューニング
  - 様々な λ の値に対して交差検証で Test MSE を推定し、最も精度が良くなる λ を選ぶ
  - 例）リッジ回帰

Credit データセット、LOO交差検証

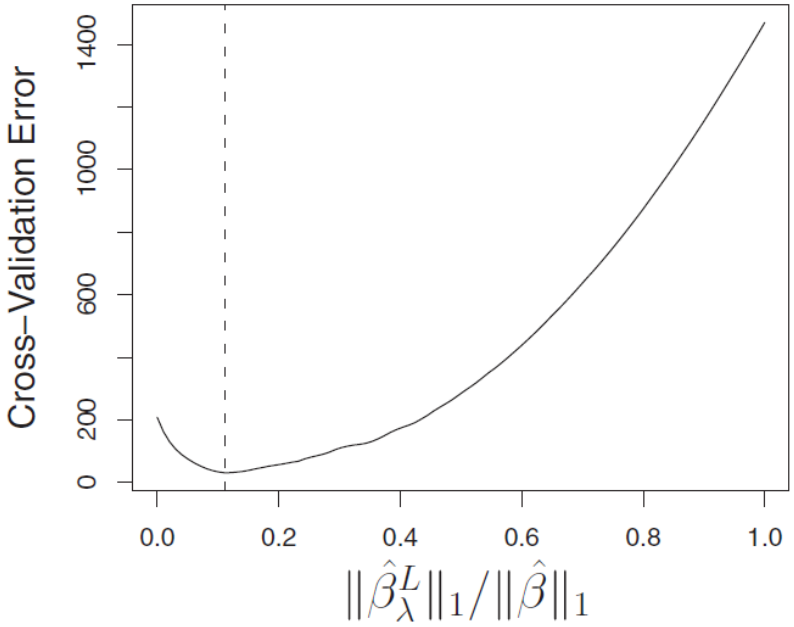


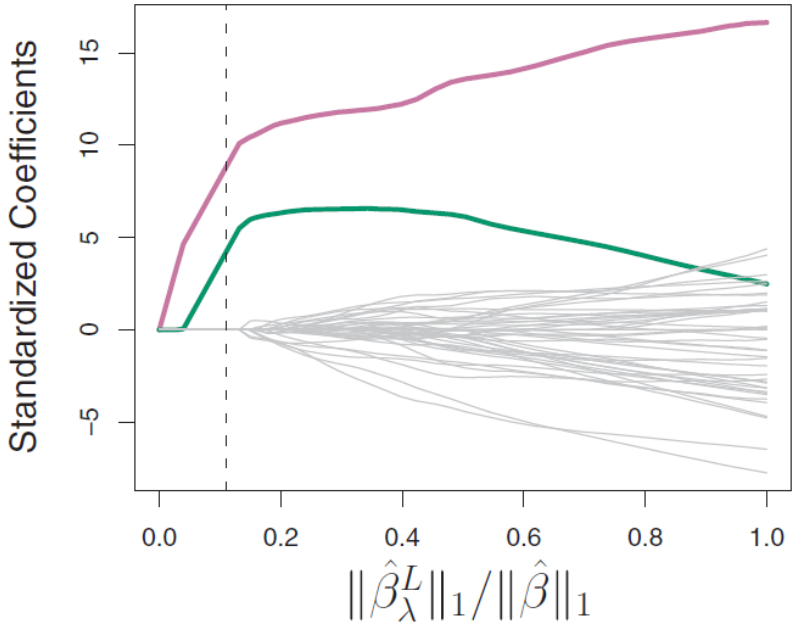


縮小推定

- 交差検証による λ のチューニング
  - 例）Lasso

    人工データ（p = 45, n = 50, 予測変数は2個を除きすべてノイズ）
    10分割交差検証





縮小推定

- ロジスティック回帰モデル等、他の回帰・分類モデルでも同様の拡張
  - もともとの目的関数に正則化項を追加
- Elastic net
  - L1正則化項とL2正則化項の両方を追加
- 最適化（パラメータ推定）
  - L1正則化では勾配（gradient）の計算が難しくなる

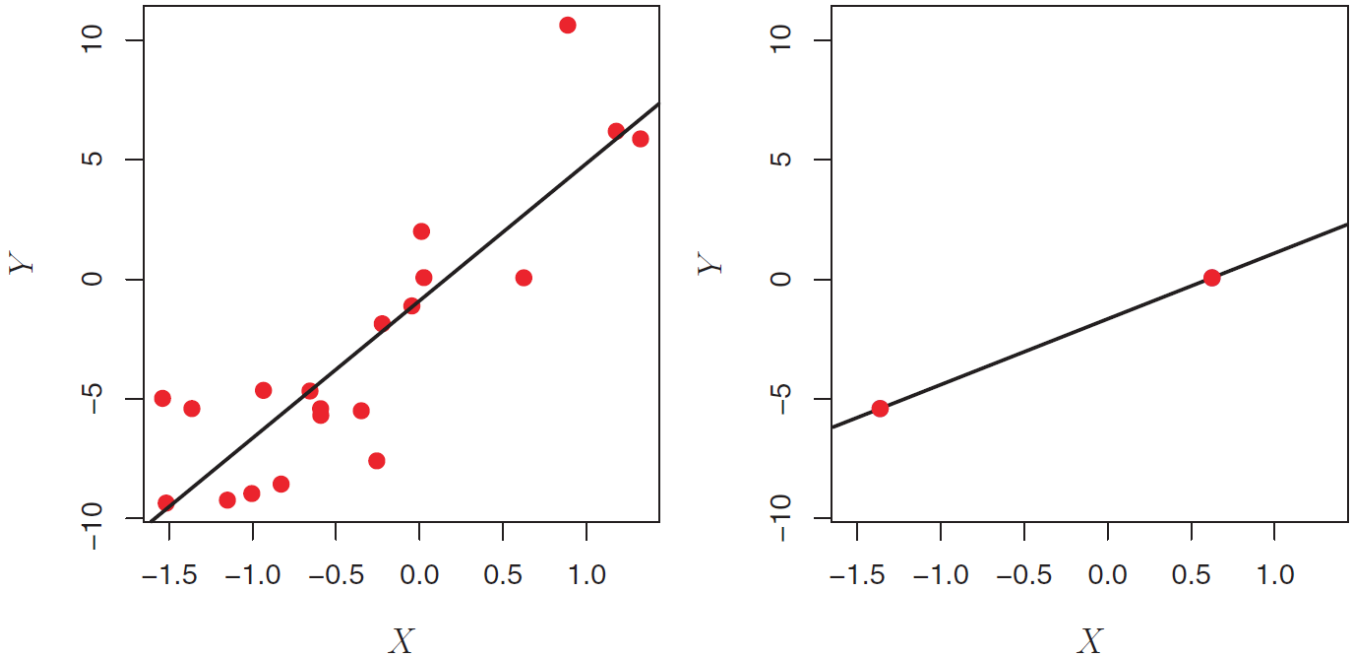
高次元データでの学習

- 高次元データ
  - 予測変数（特徴量）の数 p が学習事例の個数 n よりも大きい
  - 例
    - 血圧の予測
      - 特徴量：年齢、性別、BMI、**遺伝子情報（SNPs）**
      - $n \approx 200$, $p \approx 500{,}000$
    - オンラインショッピングの購買行動予測
      - 特徴量：顧客の検索語の履歴
      - $n \approx 1{,}000$, $p \approx 500{,}000$
  - 正則化なしの最小二乗法による学習は不適切

高次元データでの学習

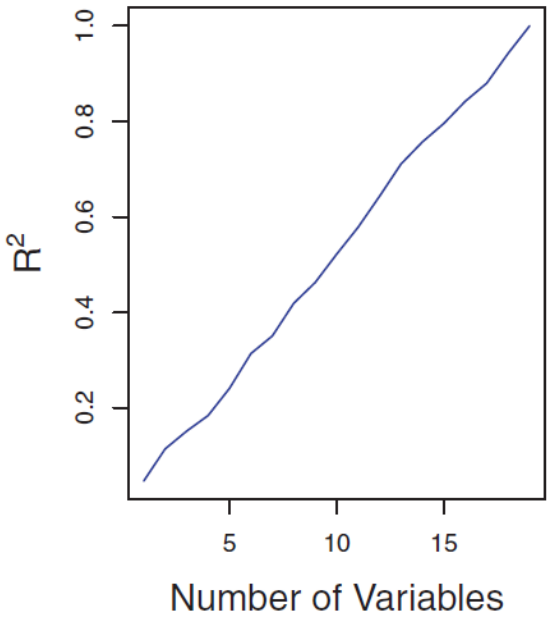
- $p \geq n$ の場合に何が問題か？
  - 常に学習データに完全にフィット（RSS = 0）
  - 予測変数が完全にノイズでも同様

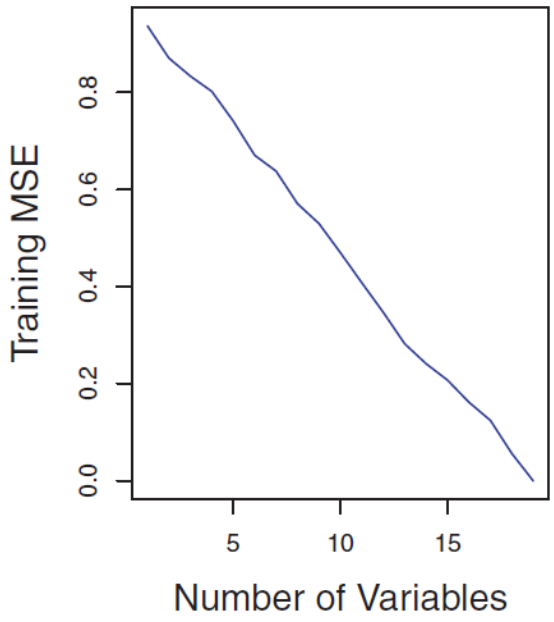
→ 過学習

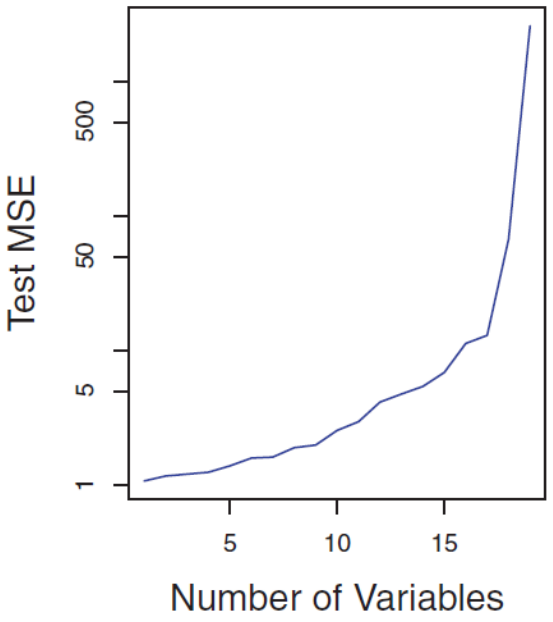


高次元データでの学習

- 例）人工データ
  - 線形回帰: $n = 20$
  - 予測変数の値は応答変数の値と無関係





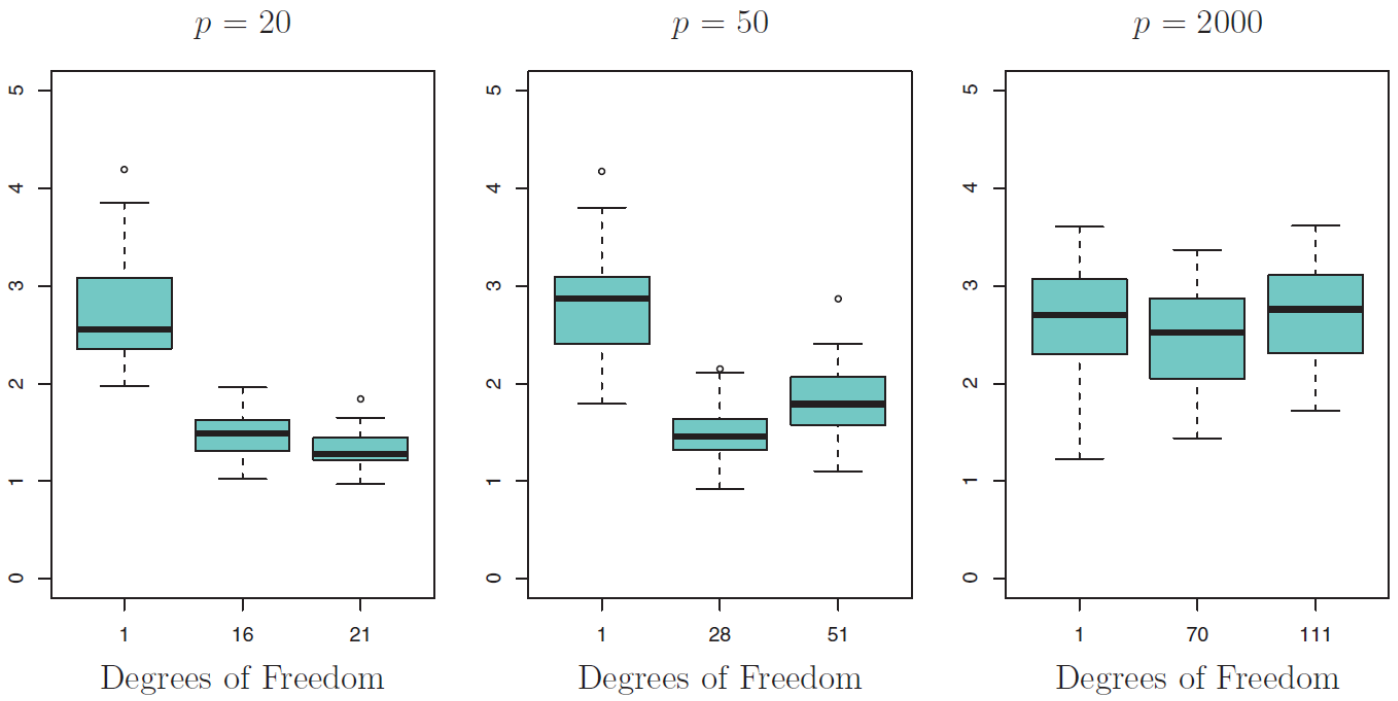


変数を増やすと学習データでの性能は向上

テストデータでは悪化

高次元データでの学習

- 対処法
  - 正則化や変数選択などによりモデルの柔軟性を下げる
  - 例）Lasso
    - n = 100、予測変数のうち応答変数と関係しているのは20個だけ



（ゼロでないパラメータの数）

高次元データでの学習

- 注意点
  - たとえ正則化を行っても無駄な予測変数が増えればモデルの精度は低下する
  - （多重共線性のため）変数選択によって選ばれた変数以外にも有用な変数が多数存在しうる
  - 学習データで得られるRSSや決定係数はモデルの性能の指標としては無意味