

データサイエンス 第5回

～回帰分析（１）～

情報理工学系研究科
創造情報学専攻
中山 英樹

本日の内容

- 先週の復習（と追加の話題）
 - Locality preserving projection
 - 回帰分析
 - 線形回帰分析
-

復習

- 多変量解析

- 目的変数がない場合

説明変数	手法
量的データ(比尺度)	主成分分析、因子分析
量的データ(間隔尺度)	クラスター分析、多次元尺度構成法、数量化Ⅳ類
質的データ	数量化Ⅲ類、対応分析

- 目的変数がある場合

目的変数	説明変数	手法
量的データ	量的データ	回帰分析
	質的データ	数量化Ⅰ類
質的データ	量的データ	判別分析
	質的データ	数量化Ⅱ類

ダミー変数

主成分分析：Principal Component Analysis (PCA)

- p次元の特徴ベクトル $\mathbf{x} = (x_1, x_2, \dots, x_p)^T$ を、元のデータの構造をできるだけ保ったまま低次元へ圧縮したい

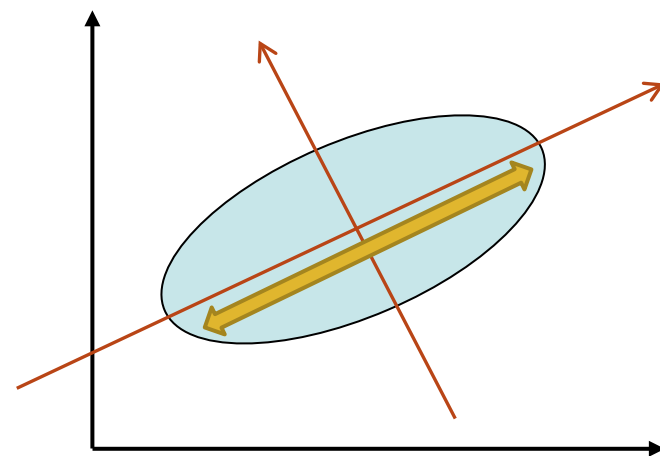
線形射影： $z = a_1x_1 + a_2x_2 + \dots + a_px_p = \mathbf{a}^T \mathbf{x}$ (ただし $\mathbf{a}^T \mathbf{a} = 1$)

- データの分布を最もよく記述する軸は？

⇒ 分散最大基準

$$\text{var}(z) = \frac{1}{n} \sum_{i=1}^n (z_i - \bar{z})^2$$

を最大化する \mathbf{a} を求めたい



PCA : 分散最大基準による導出

$$\begin{aligned}\text{var}(z) &= \frac{1}{n} \sum_{i=1}^n (z_i - \bar{z})^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\mathbf{a}^T \mathbf{x}_i - \mathbf{a}^T \bar{\mathbf{x}})^2 \\ &= \frac{1}{n} \sum_{i=1}^n (\mathbf{a}^T \mathbf{x}_i - \mathbf{a}^T \bar{\mathbf{x}})(\mathbf{a}^T \mathbf{x}_i - \mathbf{a}^T \bar{\mathbf{x}})^T \\ &= \frac{1}{n} \sum_{i=1}^n \mathbf{a}^T (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \mathbf{a} \\ &= \mathbf{a}^T \left(\frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \right) \mathbf{a} \\ &= \mathbf{a}^T \underline{C_X} \mathbf{a}\end{aligned}$$

\mathbf{x} の共分散行列

$$J_{PCA} = \mathbf{a}^T C_X \mathbf{a} \text{ を}$$

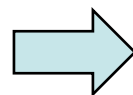
$$\mathbf{a}^T \mathbf{a} = 1 \text{ のもとで最大化}$$



$$J'_{PCA} = \mathbf{a}^T C_X \mathbf{a} - \lambda (\mathbf{a}^T \mathbf{a} - 1) \text{ を最大化}$$

(ラグランジュの未定乗数法)

$$\frac{\partial J'_{PCA}}{\partial \mathbf{a}} = 2C_X \mathbf{a} - 2\lambda \mathbf{a} = 0 \text{ (停留点)}$$



$$C_X \mathbf{a} = \lambda \mathbf{a}$$

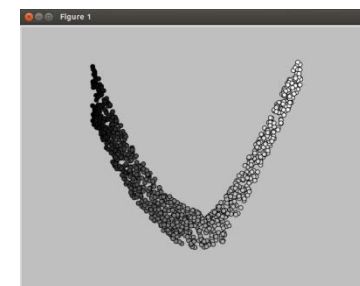
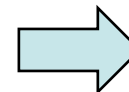
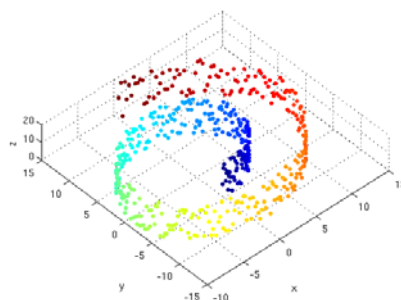
※行列の微分についてはmatrix cookbook等を参照
<http://orion.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf>

Locally linear embedding (LLE) [Roweis & Saul, 2000]

- PCAはデータの分布の非線形構造をつぶしてしまう
- LLEでは局所構造を保存した圧縮を行う
- ポイント：局所的には線形な構造を持っているとみなせる
 - 近傍サンプルの重みづけ和で表せる

$$\hat{\mathbf{x}}_i = \sum_{j \in N(i)} W_{ij} \mathbf{x}_j \quad \left(\sum_j W_{ij} = 1 \right)$$

j の近傍サンプル



LLE概要

- 1. 各サンプルの二乗誤差を最小とする W を求める
(解析的に計算できる)

$$\mathcal{E}_i = \left\| \mathbf{x}_i - \sum_{j \in N(i)} W_{ij} \mathbf{x}_j \right\|^2$$

- 2. 求まった W のもとで、同じ基準で誤差を最小とするように低次元のベクトル \mathbf{y} を設定する

$$\sum_{i=1}^n \left\| \mathbf{y}_i - \sum_{j \in N(i)} W_{ij} \mathbf{y}_j \right\|^2 \quad \leftarrow \text{学習サンプルしか埋め込めない}$$
$$= \text{tr}(\mathbf{Y}^T \mathbf{M} \mathbf{Y}) \quad \text{ただし} \quad \mathbf{M} = (\mathbf{I} - \mathbf{W})^T (\mathbf{I} - \mathbf{W}), \quad \mathbf{Y}^T \mathbf{Y} = \mathbf{I}$$

\mathbf{Y} の第 i 列が \mathbf{y}_i

Locality preserving projection (LPP)

- 局所的な類似度構造を保存する線形射影を学習

- 1. 類似度行列 W を計算

- ガウス類似度: $W_{i,j} = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\delta}\right)$

- k 最近傍類似度: $W_{i,j} = \begin{cases} 1 & \mathbf{x}_i \text{が}\mathbf{x}_j \text{の}k\text{最近傍に含まれるか、} \\ & \mathbf{x}_j \text{が}\mathbf{x}_i \text{の}k\text{最近傍に含まれる場合} \\ 0 & \text{それ以外の場合} \end{cases}$

- 2. 類似度で重み付けたサンプル間の埋め込み距離を最小化

$$J_{LPP} = \frac{1}{2} \sum_{i,j}^N W_{i,j} \|\mathbf{a}^T \mathbf{x}_i - \mathbf{a}^T \mathbf{x}_j\|^2$$

LPP つづき

$$\begin{aligned}\frac{1}{2} \sum_{i,j}^N W_{i,j} \|\mathbf{a}^T \mathbf{x}_i - \mathbf{a}^T \mathbf{x}_j\|^2 &= \sum_i^N \mathbf{a}^T \mathbf{x}_i D_{ii} \mathbf{x}_i^T \mathbf{a} - \sum_{i,j}^N \mathbf{a}^T \mathbf{x}_i W_{ij} \mathbf{x}_j^T \mathbf{a} \\ &= \mathbf{a}^T X (D - W) X^T \mathbf{a} = \mathbf{a}^T X L X^T \mathbf{a}\end{aligned}$$

$$X = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$$

D : (対角行列) $D_{ii} = \sum_j^N W_{ij}$ サンプル i にかかる重みの総和

$$L = D - W \quad \text{グラフラプラシアン行列}$$

上記を $\mathbf{a}^T X D X^T \mathbf{a} = 1$ の条件下で最小化 (重み付の分散正規化)

$$\Rightarrow X L X^T \mathbf{a} = \lambda X D X^T \mathbf{a}$$

の最小 m 固有値に対応する固有ベクトル

使いどころ

- 機械学習の言葉でいうと…
- データから何かを発見したい → 教師なし学習

説明変数	手法
量的データ(比尺度)	主成分分析、因子分析、LPP
量的データ(間隔尺度)	クラスター分析、多次元尺度構成法、数量化Ⅳ類
質的データ	数量化Ⅲ類、対応分析

- データを使って何かを予測したい → 教師あり学習

目的変数	説明変数	手法
量的データ	量的データ	回帰分析
	質的データ	数量化Ⅰ類
質的データ	量的データ	判別分析
	質的データ	数量化Ⅱ類

回帰分析

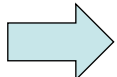
- 説明変数から目的変数を予測するモデル（関数）を構築
 - （典型的には）量的データから量的データを予測
 - 例）人口・気温から消費電力を予測

以下では

p 次元のベクトル入力 \mathbf{x} から実数値 y を出力するモデル $y = f_{\theta}(\mathbf{x})$ を、 N 個の学習サンプル $\{\mathbf{x}_i, y_i\}_{i=1}^N$ から学習する

という問題を考える

(θ はモデルのパラメータ)

- 学習サンプルに最もフィットするモデルを求めたい
 - “フィット”するとは？  予測と真の値のズレを最小化する！

最小二乗法 (least squares)

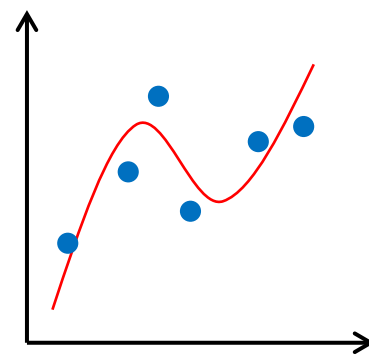
- 学習サンプルにおける、モデルの出力 $\hat{y}_i = f_{\theta}(\mathbf{x}_i)$ と真値 y_i の二乗誤差

$$E(\theta) = \sum_{i=1}^N (y_i - \hat{y}_i)^2 = \sum_{i=1}^N (y_i - f_{\theta}(\mathbf{x}_i))^2 \text{ を最小化}$$

- 誤差の分布に分散一定の正規分布を仮定
 - 他にも誤差の基準はいろいろある (異なる分布に対応)
 - 二乗誤差を基準にすると実用上はとても便利
- 例) 多項式フィッティング (xは1次元)

$$f(x) = w_0 + w_1x + w_2x^2 + w_3x^3 + \cdots + w_Mx^M$$

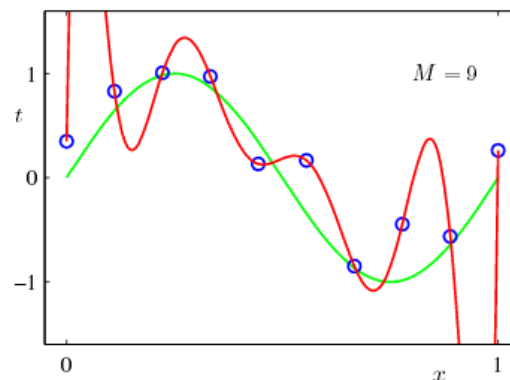
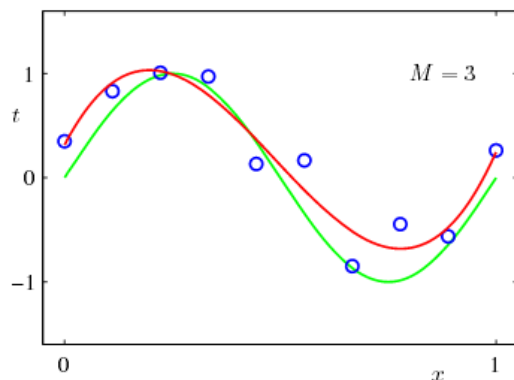
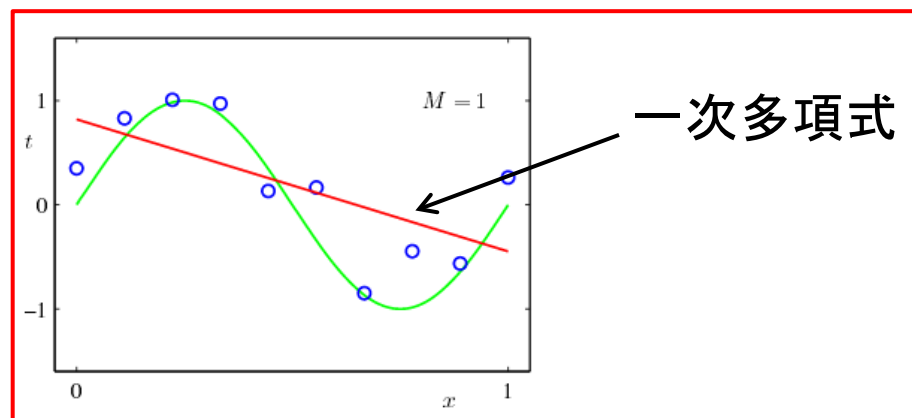
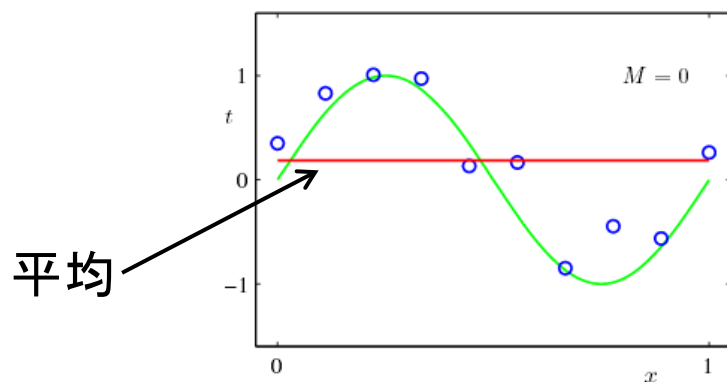
モデルパラメータ $(w_0, w_1, w_2, \dots, w_M)$ を学習



Q. 複雑なモデル(パラメータが多い)ほど性能がよいのだろうか？

過学習（オーバーフィッティング）

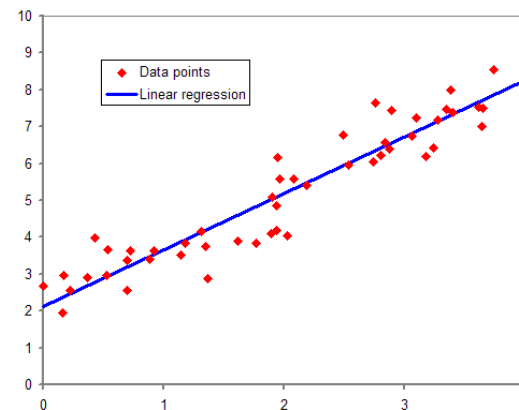
- 訓練誤差は小さいが、汎化誤差（予測誤差）が著しく大きい状態
 - データ数や実際の構造に対して複雑すぎるモデルを用いることで生じる
- 多項式回帰の次数を上げていけば、全ての学習データを通る線は引ける
 - 実際には意味のない解！（訓練誤差はゼロだが汎化誤差は非常に大きくなる）
 - 学習サンプルは有限



C.M.ビショップ
「パターン認識と
機械学習」より

線形回帰: linear regression

- 説明変数の一次式（線形結合）によるモデル
 - 単回帰：説明変数が一つ $y = ax + b$
 - 重回帰：説明変数が複数 $y = a_0x_0 + a_1x_1 + a_2x_2 + \cdots + a_px_p + b = \mathbf{a}^T \mathbf{x} + b$
 - 説明変数は互いに無相関であることを仮定
- 対象が線形の関係性を有する場合に力を発揮
 - 実用上は厳密に線形であることは少ない（そもそも分からないことの方が多い）
 - 予測性能が実用に耐えるかの評価が大事
 - 前処理（対数をとったり）は重要
- シンプルなモデルであり、学習はスケーラブル



線形回帰

- 最小二乗法によるモデルフィッティング

$$E(\mathbf{a}, b) = \sum_{i=1}^N (y_i - \hat{y}_i)^2 = \sum_{i=1}^N (y_i - (\mathbf{a}^T \mathbf{x}_i + b))^2$$

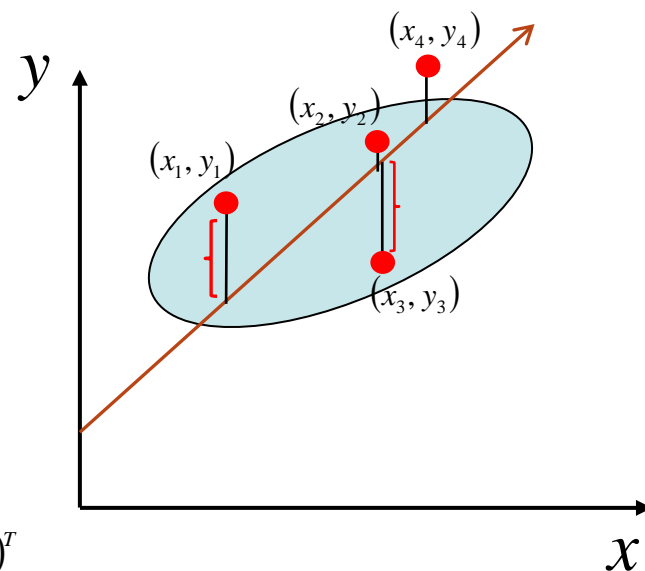
\mathbf{a}, b でそれぞれ偏微分して0とおくと

$$\frac{\partial E}{\partial \mathbf{a}} = -2 \sum_{i=1}^N \mathbf{x}_i (y_i - \mathbf{a}^T \mathbf{x}_i) = 0 \quad \Rightarrow \quad \hat{\mathbf{a}} = \left(\sum_{i=1}^N \mathbf{x}_i \mathbf{x}_i^T \right)^{-1} \left(\sum_{i=1}^N \mathbf{x}_i y_i \right)$$
$$= \underbrace{(X X^T)^{-1}}_{\text{自己相関行列}} X \mathbf{y}$$

ただし $X = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N)$, $\mathbf{y} = (y_1, y_2, \dots, y_N)^T$

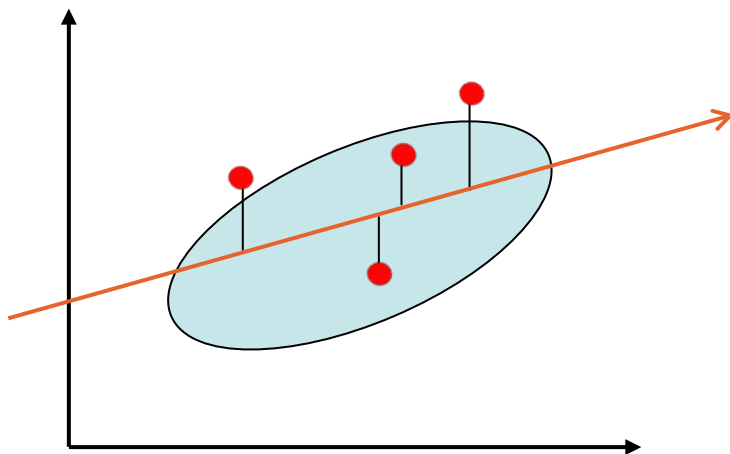
$$\frac{\partial E}{\partial b} = -2 \sum_{i=1}^N (y_i - (\mathbf{a}^T \mathbf{x}_i + b)) = 0 \quad \Rightarrow \quad \hat{b} = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{\mathbf{a}}^T \mathbf{x}_i)$$

定数項 b を用意する代わりに、特徴ベクトル \mathbf{x} に常に1の要素を付与してもよい

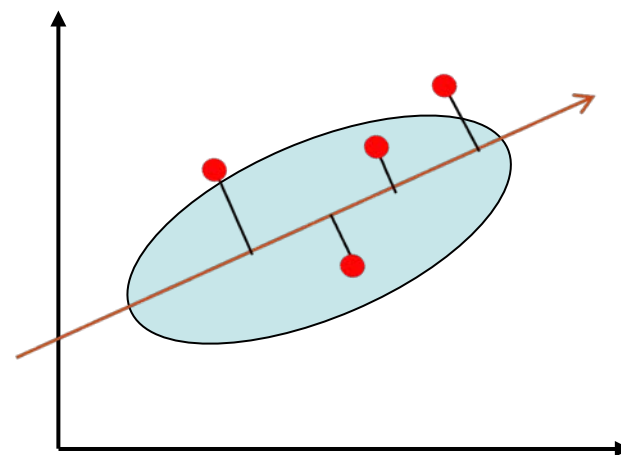


主成分分析(PCA)との違い

- 誤差の測り方が違う



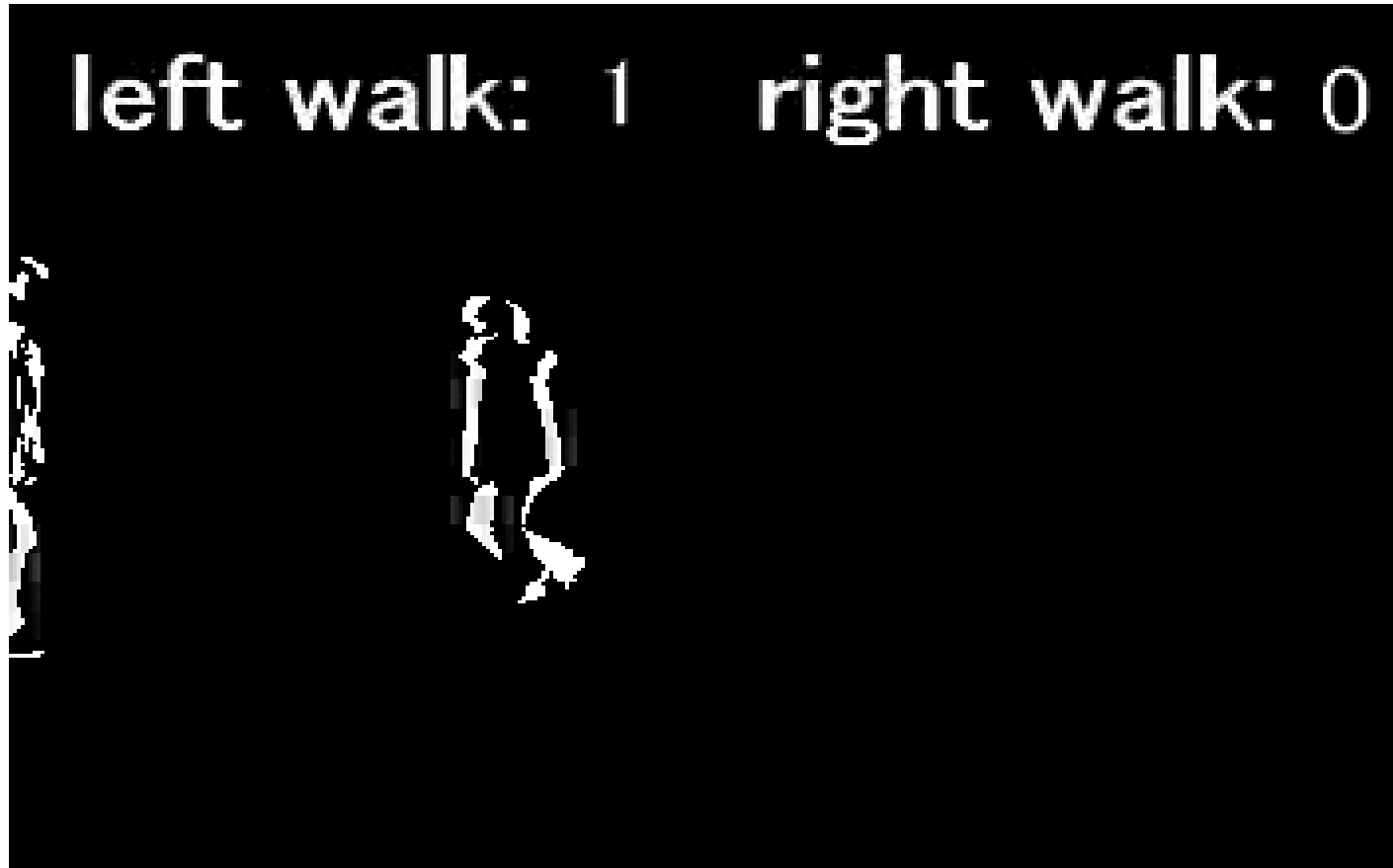
線形回帰



PCA

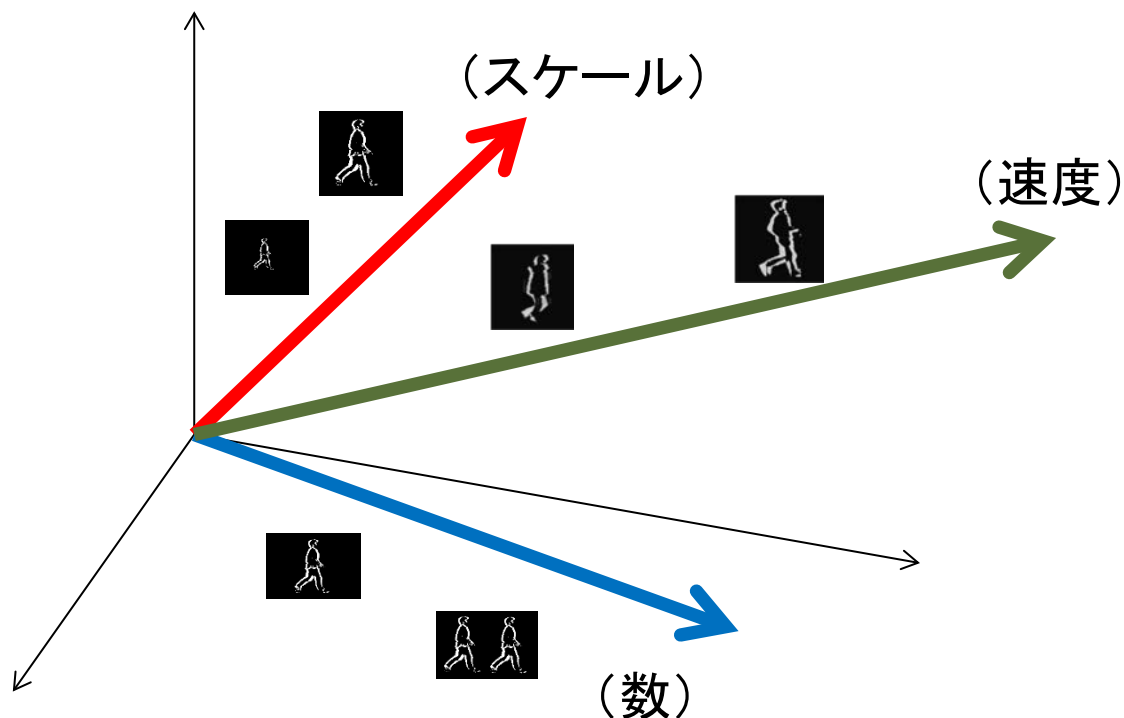
例) 線形回帰で歩行者数を計測

- [Shimohata and Otsu, 2006]



例) 線形回帰で歩行者数を計測

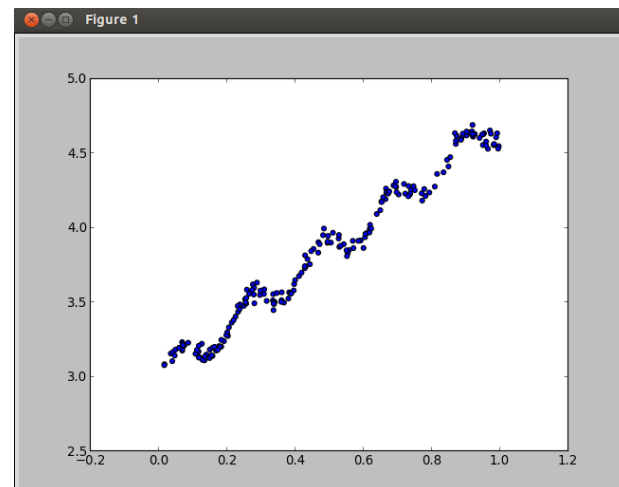
- 特徴空間の線形構造を上手に利用 (CHLAC特徴)



対象と特徴(説明変数)の性質をうまく使えば十分実用的

実行例 (1/2)

```
>>> import regression
>>> from numpy import *
>>> X,Y=regression.loadDataSet('ex0.txt')
>>> X[0:2] #最初の2サンプルを表示
[[1.0, 0.067732], [1.0, 0.42781]] #最初の要素は常に1
>>> Y[0:2]
[3.176513, 3.816464]
>>> w = regression.standRegres(X,Y)
>>> w
matrix([[ 3.00774324],
        [ 1.69532264]])
matrix([[ 3.00774324],
        [ 1.69532264]]) ←  $y = 1.6953 * x + 3.0077$ 
>>> xMat = mat(X) #matrix型へ変換
>>> yMat = mat(Y)
>>> yHat = xMat*w
```



※数式では一つのサンプルは通常列ベクトルで表記されるが、実際の実装では行ベクトルの形で格納されることが多いので注意

実行例 (2/2)

```
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.scatter(xMat[:,1],flatten().A[0], yMat.T[:,0].flatten().A[0])
    (オリジナルのデータのプロットを準備. plt.show()するまでは表示されない)
```

```
>>> xCopy = xMat.copy()
>>> xCopy.sort(0)  #コピーして並び替え pyplotの都合
>>> yHat = xCopy*w
>>> ax.plot(xCopy[:,1],yHat)
>>> plt.show()
```

コード

```
def standRegres(xArr,yArr):  
    xMat = mat(xArr); yMat = mat(yArr).T  
    xTx = xMat.T*xMat  
    if linalg.det(xTx) == 0.0:  
        print "This matrix is singular, cannot do inverse"  
        return  
    ws = xTx.I * (xMat.T*yMat)  
    return ws
```

※普通に利用する分には、numpy.linalg.lstsq()などで十分

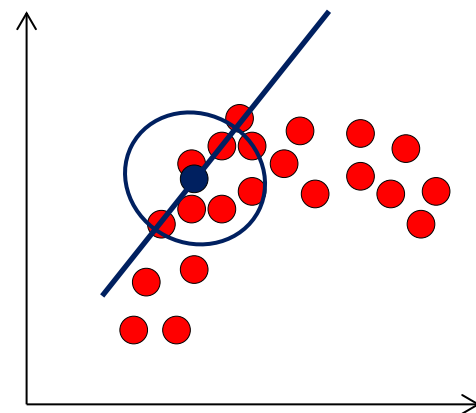
局所重み付線形回帰

- 入力されたテストデータの近傍学習サンプルに大きな重みを付与して線形回帰を実行する

$$\hat{\mathbf{a}} = \left(X W X^T \right)^{-1} X W \mathbf{y} \quad \leftarrow \text{これを入力されるテストデータごとに解く}$$

W は学習サンプルの重みをもつ対角行列 $W_{i,i} = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}_i\|}{2\delta}\right)$

- 非線形回帰手法の一つ（局所的には線形）



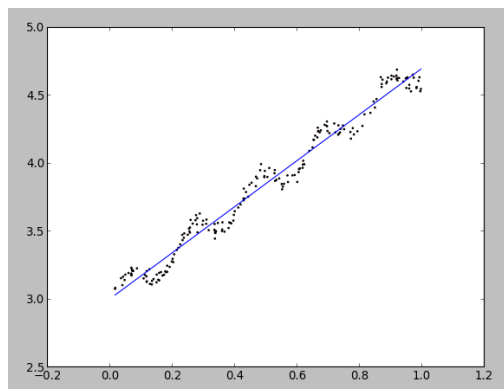
実行例 (1/2)

```
>>> import regression
>>> from numpy import *
>>> X,Y=regression.loadDataSet('ex0.txt')
>>> regression.lwlr(X[0],X,Y,1.0)
matrix([[ 3.12204471]])
>>> regression.lwlr(X[0],X,Y,0.001)
matrix([[ 3.20175729]])
```

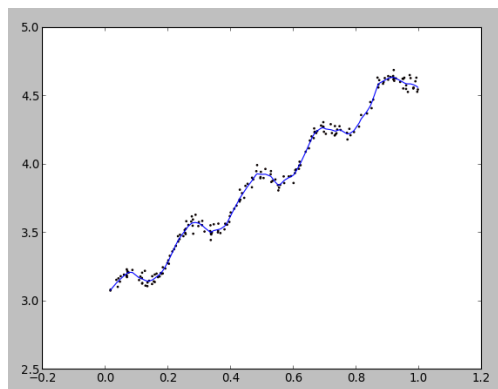
最初の1サンプル
だけ、kの値を変え
て局所線形回帰

実行例 (2/2)

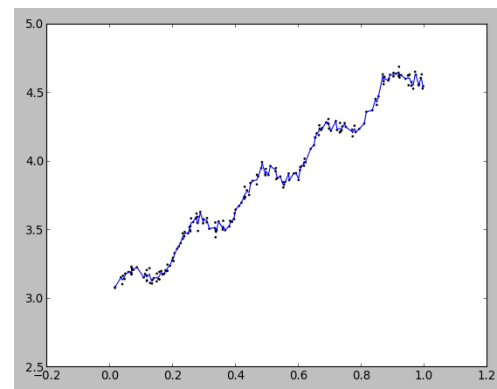
```
>>> yHat = regression.lwlrTest(X,X,Y,0.003) #全サンプルで実行
>>> xMat = mat(X)
>>> srtInd = xMat[:,1].argsort(0)
>>> xSort = xMat[srtInd][:,0,:]
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.plot(xSort[:,1],yHat[srtInd])
>>> ax.scatter(xMat[:,1].flatten().A[0], mat(Y).T.flatten().A[0], s=2, c='red')
>>> plt.show()
```



k = 1.0



k = 0.01



k = 0.003

UCI abalone データ

- アワビ貝の年齢予測

#性別	長さ	直径	厚さ	総重量	...			年齢
1	0.455	0.365	0.095	0.514	0.2245	0.101	0.15	15
1	0.35	0.265	0.09	0.2255	0.0995	0.0485	0.07	7
-1	0.53	0.42	0.135	0.677	0.2565	0.1415	0.21	9
1	0.44	0.365	0.125	0.516	0.2155	0.114	0.155	10
0	0.33	0.255	0.08	0.205	0.0895	0.0395	0.055	7
0	0.425	0.3	0.095	0.3515	0.141	0.0775	0.12	8
-1	0.53	0.415	0.15	0.7775	0.237	0.1415	0.33	20
-1	0.545	0.425	0.125	0.768	0.294	0.1495	0.26	16
1	0.475	0.37	0.125	0.5095	0.2165	0.1125	0.165	9
-1	0.55	0.44	0.15	0.8945	0.3145	0.151	0.32	19
-1	0.525	0.38	0.14	0.6065	0.194	0.1475	0.21	14
1	0.43	0.35	0.11	0.406	0.1675	0.081	0.135	10
1	0.49	0.38	0.135	0.5415	0.2175	0.095	0.19	11

実行してみる

```
>>> import regression
>>> from numpy import *

>>> abX,abY = regression.loadDataSet('abalone.txt')

>>> yHat01=regression.lwlrTest(abX[0:99],abX[0:99],abY[0:99],0.1)  #学習・テ
ストと同じデータ（最初の100サンプル）で局所線形回帰
>>> yHat1=regression.lwlrTest(abX[0:99],abX[0:99],abY[0:99],1)
>>> yHat10=regression.lwlrTest(abX[0:99],abX[0:99],abY[0:99],10)

>>> regression.rssError(abY[0:99],yHat01.T)  #訓練誤差
56.784938106908534
>>> regression.rssError(abY[0:99],yHat1.T)
429.89056187025272
>>> regression.rssError(abY[0:99],yHat10.T)
549.1181708825427
```

k = 0.1が一番いいのかな…？

汎化誤差の評価

```
>>> yHat01=regression.lwlrTest(abX[100:199],abX[0:99],abY[0:99],0.1) #テストデータを別にとる
>>> yHat1=regression.lwlrTest(abX[100:199],abX[0:99],abY[0:99],1)
>>> yHat10=regression.lwlrTest(abX[100:199],abX[0:99],abY[0:99],10)
```

```
>>> regression.rssError(abY[100:199],yHat01.T)
```

```
22339.480891855663
```

```
>>> regression.rssError(abY[100:199],yHat1.T)
```

```
573.52614418969961
```

```
>>> regression.rssError(abY[100:199],yHat10.T)
```

```
517.5711905383497
```

ダミー変数

- 名義尺度データを説明変数に使いたい場合
 - カテゴリ情報を二値(0,1など)で表す
 - 数量化 I 類と呼ばれる方法と実質的に同等

#性別	長さ	直径	厚さ	総重量	...			年齢
1	0.455	0.365	0.095	0.514	0.2245	0.101	0.15	15
1	0.35	0.265	0.09	0.2255	0.0995	0.0485	0.07	7
-1	0.53	0.42	0.135	0.677	0.2565	0.1415	0.21	9
1	0.44	0.365	0.125	0.516	0.2155	0.114	0.155	10
0	0.33	0.255	0.08	0.205	0.0895	0.0395	0.055	7
0	0.425	0.3	0.095	0.3515	0.141	0.0775	0.12	8



本当は0は欠損値扱いにしないとよろしくない...

線形回帰の弱点（１）

- 説明変数の間に強い相関や線形従属の関係があると正しい解がでない
(= 多重共線性, multicollinearity)

$$\hat{\mathbf{a}} = \left(\mathbf{X}\mathbf{X}^T \right)^{-1} \mathbf{X} \mathbf{y}$$

相関行列がランク落ちし、逆行列が求まらない

- 実データでは、説明変数の間に相関がある方がふつう
 - 解けないだけならまだいいが、不安定な解を出すことがある
(行列式がゼロに近いので、係数 \mathbf{a} が異常に大きな値になる)
 - あらかじめ相関の高い説明変数は除外しておくことが
- サンプルが次元数に対して少ない場合も不安定になりやすい

リッジ回帰

- 自己相関行列に単位行列を微小な重みをつけて加算し、安定的に逆行列を計算（正則化）

$$\hat{\mathbf{a}} = (XX^T + \gamma I)^{-1} X \mathbf{y}$$

- いろんな解釈が可能
 - 学習データにホワイトノイズを加えている
- 最小化する目的関数として、誤差項に二乗ノルムを加えている
= **L2正則化**

$$E'(\mathbf{a}, b) = \sum_{i=1}^N (y_i - (\mathbf{a}^T \mathbf{x}_i + b))^2 + \gamma \|\mathbf{a}\|^2$$



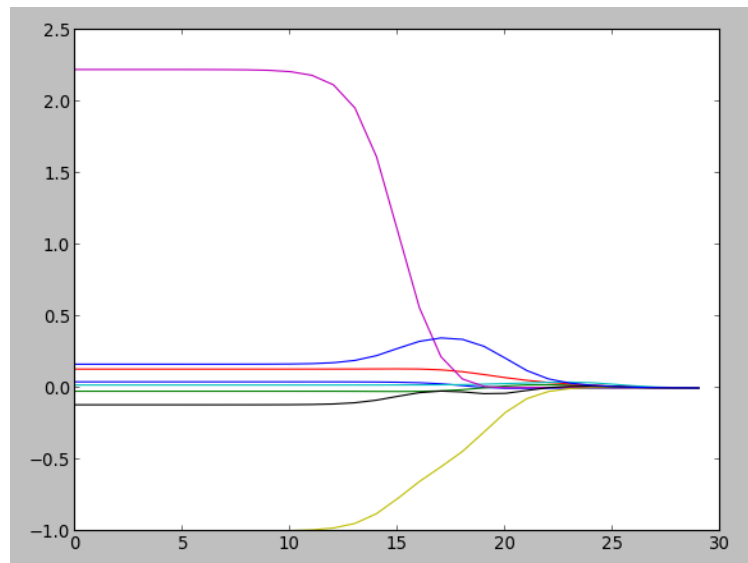
係数(の絶対値)をできるだけ小さくするような作用

正則化パラメータの影響

```
>>> reload(regression)
>>> abX,abY = regression.loadDataSet('abalone.txt')
>>> ridgeWeights=regression.ridgeTest(abX,abY)
>>> import matplotlib.pyplot as plt
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111)
>>> ax.plot(ridgeWeights)
>>> plt.show()
```

ガンマを大きくすると係数はゼロに収束

coefficients



log (gamma)

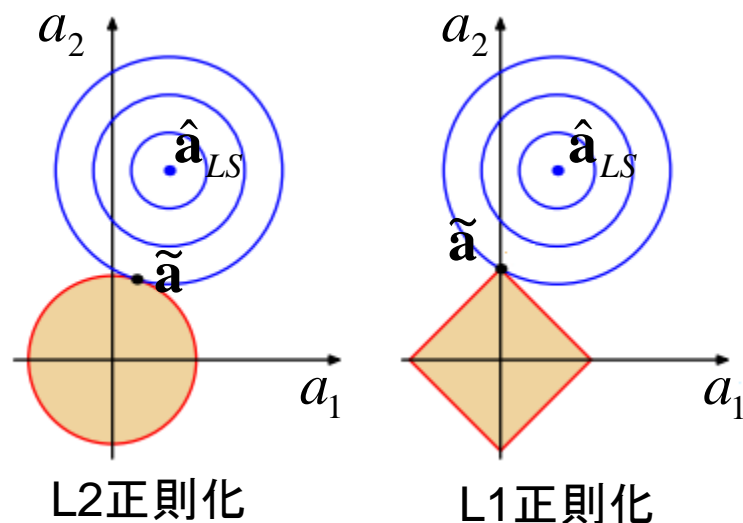
スパース線形回帰

- L1正則化(Lasso)を用いた線形回帰

$$E''(\mathbf{a}, b) = \sum_{i=1}^N \left(y_i - (\mathbf{a}^T \mathbf{x}_i + b) \right)^2 + \gamma \underbrace{\|\mathbf{a}\|_1}_{\text{L1ノルム}} = \sum_i |a_i|$$

- スパース（ゼロ要素が多い）係数ベクトルが出やすい
- データ容量や計算コストの削減につながる

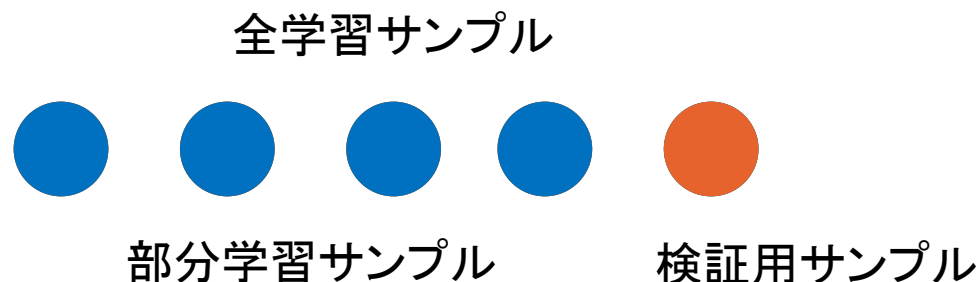
sklearn.linear_model.Lasso など



パラメータのチューニング方法

- 交差検定法（クロスバリデーション）

1. もともとの学習サンプルを分割し、新しい学習サンプルと検証用サンプルに分割する（例えば5:1などに）
2. 新しい学習サンプルで、候補となるモデル（この場合パラメータごとに）を学習し、検証用データの予測誤差を調べる
3. 学習サンプルと検証用サンプルを順番に入れ替え、2を繰り返す
4. 全試行の平均予測誤差が最も小さかったものを採用する



今日のまとめ

- 回帰入門
 - データの性質・量と、適用すべきモデルの複雑さをよく考える
 - 訓練誤差、汎化誤差の違いに注意
 - オーバーフィッティングに注意
- 線形回帰分析
 - 単純だが全ての基本。意外と実問題にも使える。
 - 多重共線性、正則化の意味を理解すること
 - リッジ回帰を使いましょう（交差検定も）
- 来週
 - 一般化線形モデル（誤差が正規分布以外の場合）
 - 非線形回帰