線形回帰

線形回帰とは、回帰式が線形なものである。ここでは、まず、単回帰について説明し、そのあと、単回帰を拡張した重回帰について説明する。

単回帰

単回帰とは、あるデータ分布に対して、当てはまりの良い近似直線を求めることである。つまり、学習データ$(\boldsymbol{X}, \boldsymbol{Y})$がある時、そのデータ分布に対して、$y = ax + b$の近似式を求め、近似式の$x$に検証データの$\boldsymbol{X}$を代入することで、$\boldsymbol{Y}$を推定することができる。

理論

近似式$y = ax + b$のパラメータa, bは、最小二乗法（誤差の二乗和の最小化）により求めることができる。まず、学習データを

$$\boldsymbol{x} = (x_1, ..., x_n)^T, \quad \boldsymbol{y} = (y_1, ..., y_n)^T$$

とすると、誤差の二乗和Eは

$$E = \sum_{i=1}^{n} (ax_i + b - y_i)^2$$

Eが最小となる時のパラメータa, bを求めるためには、$\frac{\partial E}{\partial a} = 0$, $\frac{\partial E}{\partial b} = 0$の連立方程式を解けばよい。まず、$\frac{\partial E}{\partial a} = 0$より

$$\sum_{i=1}^{n} x_i(ax_i + b - y_i) = 0 \tag{1}$$

また、$\frac{\partial E}{\partial b} = 0$より

$$\sum_{i=1}^{n} (ax_i + b - y_i) = \sum_{i=1}^{n} ax_i + nb - \sum_{i=1}^{n} y_i = 0$$

$$\therefore b = \frac{1}{n} \sum_{i=1}^{n} (y_i - ax_i) \tag{2}$$

式(1)に式(2)を代入して計算していくと

$$\begin{aligned} \sum_{i=1}^{n} x_i(ax_i + b - y_i) &= a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i - \sum_{i=1}^{n} x_i y_i \\ &= \dots \\ &= a\left[\sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2\right] + \frac{1}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i - \sum_{i=1}^{n} x_i y_i \end{aligned}$$

上式 = 0より、aは

$$\therefore a = \frac{\sum_{i=1}^{n} x_i y_i - \frac{1}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{\sum_{i=1}^{n} x_i^2 - \frac{1}{n}(\sum_{i=1}^{n} x_i)^2} \tag{3}$$

よって、式(2)と式(3)を計算すればよい。以下に実装したコードを示した。

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

class SimpleRegression:
    def __init__(self):
        self.intercept = None # 切片
        self.coefficient = None # 傾き

    def fit(self, x, y):
        n = len(x) # 要素数
        self.coefficient = (np.dot(x, y) - x.sum() * y.sum() / n) / ((x**2).sum()
- x.sum()**2 / n)
        self.intercept = (y.sum() - self.coefficient * x.sum()) / n

    def predict(self, x):
        return self.coefficient * x + self.intercept

if __name__=="__main__":
    x = np.array([1, 2, 3, 6, 7, 9]) # 学習データの説明変数x
    y = np.array([1, 3, 3, 5, 4, 6]) # 学習データの目的変数y
    model = SimpleRegression()
    model.fit(x, y)

    # グラフの表示
    plt.scatter(x, y) # 学習データの分布を表示
    x_max = x.max()
    plt.plot([0, x_max], [model.intercept, model.coefficient * x_max +
model.intercept]) # 回帰直線の表示
    plt.show()
```

重回帰

上述した単回帰は、説明変数$\boldsymbol{X}$が一つであった。対して、重回帰では、説明変数$\boldsymbol{X}$をd個に拡張したもので、回帰直線（d = 1）だけでなく、回帰平面（d = 2）・回帰超平面（d $\geq$ 3）を求めることができる。

理論

まず、説明変数がd個ある時の行列$\boldsymbol{X}$は

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x_1}^T \\ \boldsymbol{x_2}^T \\ \vdots \\ \boldsymbol{x_n}^T \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

ここで、定数項を考えるために、行列$\tilde{\boldsymbol{X}}$を次のように定義する。

$$\tilde{\boldsymbol{X}} = \begin{pmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1d} \\ 1 & x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}$$

d次元の重回帰モデルは

$$\hat{y} = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \cdots + \omega_d x_d \tag{4}$$

上式を行列で考えると

$$\hat{\boldsymbol{y}} = \tilde{\boldsymbol{X}}\boldsymbol{\omega}$$

よって、最小二乗法より、次式を最小化するようなパラメータωを求めればよい。

$$\begin{aligned} E(\boldsymbol{\omega}) &= ||\boldsymbol{y} - \tilde{\boldsymbol{X}}\boldsymbol{\omega}||^2 \\ &= (\boldsymbol{y} - \tilde{\boldsymbol{X}}\boldsymbol{\omega})^T(\boldsymbol{y} - \tilde{\boldsymbol{X}}\boldsymbol{\omega}) \\ &= \boldsymbol{y}^T\boldsymbol{y} - \boldsymbol{\omega}^T\tilde{\boldsymbol{X}}^T\boldsymbol{y} - \boldsymbol{y}^T\tilde{\boldsymbol{X}}\boldsymbol{\omega} + \boldsymbol{\omega}^T\tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}}\boldsymbol{\omega} \end{aligned}$$

上式の勾配$\nabla E(\boldsymbol{\omega}) = 0$の時の$\omega$を求めればよいので、

$$\nabla E(\boldsymbol{\omega}) = -2\tilde{\boldsymbol{X}}\boldsymbol{y} + 2\tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}}\boldsymbol{\omega} = 0$$

よって、

$$\begin{aligned} \tilde{\boldsymbol{X}}\boldsymbol{y} = \tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}}\boldsymbol{\omega} \\ \boldsymbol{\omega} = (\tilde{\boldsymbol{X}}^T\tilde{\boldsymbol{X}})^{-1}\tilde{\boldsymbol{X}}^T\boldsymbol{y} \end{aligned}$$

上式を用いてパラメータ$\boldsymbol{\omega}$を算出する。以下に実装したコードを示した。

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

class LinearRegression:
    def __init__(self):
        self.weight = None # 重み
        self.intercept = None # 定数項
        self.coefficient = None # 偏回帰係数

    def fit(self, x, y):
        x_tilda = np.c_[np.ones(x.shape[0]), x]
        A = np.dot(x_tilda.T, x_tilda)
        B = np.dot(x_tilda.T, y)
        self.weight = np.dot(np.linalg.inv(A), B) # np.linalg.inv: 逆行列に変換
        self.intercept = self.weight[0]
        self.coefficient = self.weight[1:]

    def predict(self, x):
        if x.ndim == 1:
            x = x.reshape(1, -1)
```

```
        x_tilda = np.c_[np.ones(x.shape[0]), x]
        return np.dot(x_tilda, self.weight)

if __name__=="__main__":
    # 適当な学習データを生成
    n = 50 # 要素数
    scale = 100
    np.random.seed(0) # 乱数生成器のシード値を固定
    x = np.random.random((n, 2)) * scale # 説明変数x
    w0, w1, w2 = 1, 2, 3
    y = np.random.randn(n) + w0 + w1 * x[:, 0] + w2 * x[:, 1] # 目的変数y

    model = LinearRegression()
    model.fit(x, y)

    print("定数項:", model.intercept)
    print("偏回帰係数:", model.coefficient)
    print("重み:", model.weight)

    # グラフの表示
    x_grid, y_grid = np.meshgrid(np.linspace(0, scale, 20), np.linspace(0, scale,
20))
    z = (model.weight[0] + model.weight[1] * x_grid.flatten() + model.weight[2] *
y_grid.flatten()).reshape(x_grid.shape)
    ax = Axes3D(plt.figure(figsize = (8, 5)))
    ax.scatter3D(x[:, 0], x[:, 1], y, color = "blue") # 入力データの分布を表示
    ax.plot_wireframe(x_grid, y_grid, z, color = "red") # 回帰平面の表示
```

多項式回帰

多項式回帰とは、多項式を利用した回帰のことで、説明変数xに対して予測値の目的変数yがxの多項式関数（2次関数、3次関数などの高次関数）で表される。非線形性を取り入れることで、回帰式は曲線（または、曲面、超曲面）で表される。

理論

説明については、簡単のため、学習データの説明変数Xは一つ（一次元）と仮定する。この時、多項式の次数をdとすると、予測値$\tilde{y}$は

$$\tilde{y} = \omega_0 + \omega_1 x + \omega_2 x^2 + \cdots + \omega_d x^d \tag{5}$$

式(5)において、x^iをx_iに置き換えれば、式(4)と一致する。これより、学習データの説明変数$(x_1, x_2, \ldots, x_n)^T$について、各要素0次からd次のべきまでを並べた行列$\boldsymbol{M}$を考えればよく、

$$\boldsymbol{M} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^d \end{pmatrix}$$

よって、予測値$\tilde{y}$は次のように表すことができ、

$$\tilde{\boldsymbol{y}} = \begin{pmatrix} \omega_0 + \omega_1 x_1 + \omega_2 x_1^2 + \cdots + \omega_d x_1^d \\ \omega_0 + \omega_1 x_2 + \omega_2 x_2^2 + \cdots + \omega_d x_2^d \\ \vdots \\ \omega_0 + \omega_1 x_n + \omega_2 x_n^2 + \cdots + \omega_d x_n^d \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^d \\ 1 & x_2 & x_2^2 & \cdots & x_2^d \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^d \end{pmatrix} \begin{pmatrix} \omega_0 \\ \omega_1 \\ \vdots \\ \omega_d \end{pmatrix} = \boldsymbol{M\omega}$$

上式を用いて、線形回帰と同様、最小二乗法によりパラメータωを求めればよい。

$$min||\tilde{y}(x, \omega) - y||^2$$

以下に実装したコードを示した。

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from linreg import LinearRegression # 先ほどの線形回帰のクラスをimport

class PolynomialRegression:
    def __init__(self, degree):
        self.degree = degree
        self.weight = None

    def fit(self, x, y):
        x_pow = []
        xx = x.reshape(len(x), 1)
        for i in range(1, self.degree + 1):
            x_pow.append(xx**i)

        matrix = np.concatenate(x_pow, axis = 1)
        lr = LinearRegression() # 線形回帰を利用
        lr.fit(matrix, y)
        self.weight = lr.weight

if __name__=="__main__":
    # 適当な学習データを生成
    np.random.seed(0) # 乱数生成器のシード値を固定
    x = np.random.random(10) * 10
    y = x + np.random.randn(10) # 2*x + 1 にノイズが乗った点

    model = PolynomialRegression(6) # 6次関数で近似 (例)
    model.fit(x, y)

    print("重み:", model.weight)

    # グラフの表示
    plt.scatter(x, y, color = "blue")
    xx = np.linspace(x.min(), x.max(), 300) # x_minからx_maxまで等間隔に300の点を生成
    yy = np.array([model.predict(i) for i in xx])
    plt.plot(xx, yy, color = "red")
```