機械学習入門

第3，4回　教師あり学習/線形回帰
(Linear Regression)

機械学習で扱う数学

- 線形代数(linear algebra)
- 微分積分(calculus)
- 確率論/統計学(probability theory/statistics)

今回扱う内容

- numpy
- とってもやさしいさんすう
- matplotlib, pandasの概説
- 線形回帰のアルゴリズムについて概説
- 線形回帰実装

numpyとは

- numpyとは、数値計算のライブラリ
- データサイエンス領域や研究ではほぼほぼ使う。
- pythonを使う上でnumpyは必須レベル
- なので、numpyの主な使い方であるベクトル演算や行列演算について学んでいく

ベクトル(vector)の計算

- ベクトルとは数を一列に並べたもの
- ex)

$$\mathbf{a}=\begin{pmatrix}1\\2\end{pmatrix},\ \mathbf{b}=\begin{pmatrix}1 & 2\end{pmatrix}$$

ベクトル(vector)の計算

- ベクトルには足し算がある
- ex)

$$\mathbf{a}=\begin{pmatrix}1\\2\end{pmatrix},\ \mathbf{b}=\begin{pmatrix}3\\4\end{pmatrix}$$

$$\mathbf{a}+\mathbf{b}=\begin{pmatrix}1+3\\2+4\end{pmatrix}=\begin{pmatrix}4\\6\end{pmatrix}$$

ベクトル(vector)の計算

- numpyでは次のように定義し、計算する

```
import numpy as np
a = np.array([1, 2])
b = np.array([3, 4])
print(a+b)
$>[4 6]
```

subject0-1

numpyで２つのベクトルを

(5 6), (3 4)と定義し、その和を計算せよ

ベクトル(vector)の計算

- ベクトルには引き算がある
- ex)

$$\mathbf{a}=\begin{pmatrix}1\\2\end{pmatrix},\ \mathbf{b}=\begin{pmatrix}3\\4\end{pmatrix}$$

$$\mathbf{b}\text{-}\mathbf{a}=\begin{pmatrix}3-1\\4-2\end{pmatrix}=\begin{pmatrix}2\\2\end{pmatrix}$$

ベクトル(vector)の計算

- numpyでは次のように定義し、計算する

```
import numpy as np
a = np.array([1, 2])
b = np.array([3, 4])
print(b-a)
$>[2 2]
```

subject0-2

numpyで２つのベクトルを

(5 6), (3 4)と定義し、その差を計算せよ

ベクトル(vector)の計算

- ベクトルにはスカラー倍がある
- ex)

$$\mathbf{a}=\begin{pmatrix}1\\2\end{pmatrix}$$

$$3\mathbf{a}=\begin{pmatrix}3*1\\3*2\end{pmatrix}=\begin{pmatrix}3\\6\end{pmatrix}$$

ベクトル(vector)の計算

- numpyでは次のように定義し、計算する

```
import numpy as np
a = np.array([1, 2])
print(3*a)
$>[3 6]
```

subject0-3

numpyでベクトルを

(5 6)と定義し、そのベクトルに２をかけたものを計算せよ

ベクトル(vector)の計算

- ベクトルには掛け算がある(内積)
- ex)

$$\mathbf{a}=\begin{pmatrix}1\\2\end{pmatrix},\ \mathbf{b}=\begin{pmatrix}3\\4\end{pmatrix}$$

$$\mathbf{a}\cdot\mathbf{b}=(1*3+2*4)=11$$

ベクトル(vector)の計算

- numpyでは次のように定義し、計算する

```
import numpy as np
a = np.array([1, 2])
b = np.array([3, 4])
print(a @ b)
$>11
```

subject0-4

numpyで２つのベクトルを

(5 6), (3 4)と定義し、その内積を計算せよ

行列(matrix)の計算

- 行列とは数を縦横に並べたもの

$$A=\begin{pmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{pmatrix}$$

※横の並びを行といい、縦の並びを列という

ある行列の行数をm、列数をnとすると、その行列をm×n行列という。今回のAは3×2行列になっている。

行列(matrix)の計算

- numpyでは次のように定義する

```
import numpy as np
A = np.array([[1, 4],[2, 5],[3, 6]])
print(A)
print(A.shape) ##行列の行数と列数のtuple
$>[[1 4]
    [2 5]
    [3 6]]
(3, 2)
```

subject0-5

numpyで行列Aを

$A=\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$と定義し,その行列と
行数と列数を出力せよ

行列(matrix)の計算

- 行列には足し算がある
- ex)

$$A=\begin{pmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{pmatrix},\ B=\begin{pmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{pmatrix}$$

$$A+B=\begin{pmatrix}1+1 & 4+2\\ 2+3 & 5+4\\ 3+5 & 6+6\end{pmatrix}=\begin{pmatrix}2 & 6\\ 5 & 9\\ 8 & 12\end{pmatrix}$$

※行列の形が同じでないと計算できない

行列(matrix)の計算

- numpyでは次のように計算する

```
import numpy as np
A = np.array([[1, 4],[2, 5],[3, 6]])
B = np.array([[1, 2],[3, 4],[5, 6]])
print(A+B)
$>[[2 6]
    [5 9]
    [8 12]]
```

subject0-6

numpyで行列Aを

$A=\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$, $B=\begin{pmatrix}1 & 3 & 5\\ 2 & 4 & 6\end{pmatrix}$と定義し,その行列の和と差を計算せよ

行列(matrix)の計算

- **行列には掛け算がある(内積)**
- **ex)**

$$A=\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix},\ B=\begin{pmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{pmatrix}$$

$$AB=\begin{pmatrix}1*1+2*3+3*5 & 1*2+2*4+3*6\\ 4*1+5*3+6*5 & 4*2+5*4+6*6\end{pmatrix}=\begin{pmatrix}22 & 28\\ 49 & 64\end{pmatrix}$$

※行列積は一般に非可換。AB≠BAであり、さらにAの列数とBの行数が一致していないと計算できない。

今回はA=(2 3), B=(3 2) -> AB=(2 2)

行列(matrix)の計算

- 行列には転置がある(行と列をひっくり返すこと)
- ex)

$$A=\begin{pmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{pmatrix},\ B=\begin{pmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{pmatrix}$$

$$A^T=\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{pmatrix}$$

ABは計算できない。Aの転置を取ればABが計算できる。

行列(matrix)の計算

- numpyでは次のように計算する

```
import numpy as np
A = np.array([[1, 4],[2, 5],[3, 6]])
print(A.T)
$>[[ 1 2 3 ]
    [4 5 6]]
```

subject0-7

numpyで行列Aを

$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$とする

A^Tを求めよ

行列(matrix)の計算

- numpyでは次のように計算する

```
import numpy as np
A = np.array([[1, 4],[2, 5],[3, 6]])
B = np.array([[1, 2],[3, 4],[5, 6]])
print(A.T@ B)
$>[[22 28]
    [49 64]]
```

subject0-7

numpyで

$A^T = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$, B= $\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$**とするとき,行列積BAを計算せよ**

pandasとは

- pandasとは、Pythonのための強力なデータ操作ライブラリ
- 先のnumpy配列でデータを扱うのではなくExcelみたいな表形式でデータを扱ったり、NaNデータを埋めたり、データの平均とか分散を出したり、、、なんでもできる。
- 後期にPandasについては深く触れていきます～

pandas DataFrame型

pandasを使っていて多く使うのがこのPandas DataFrame型

カラム(columns)

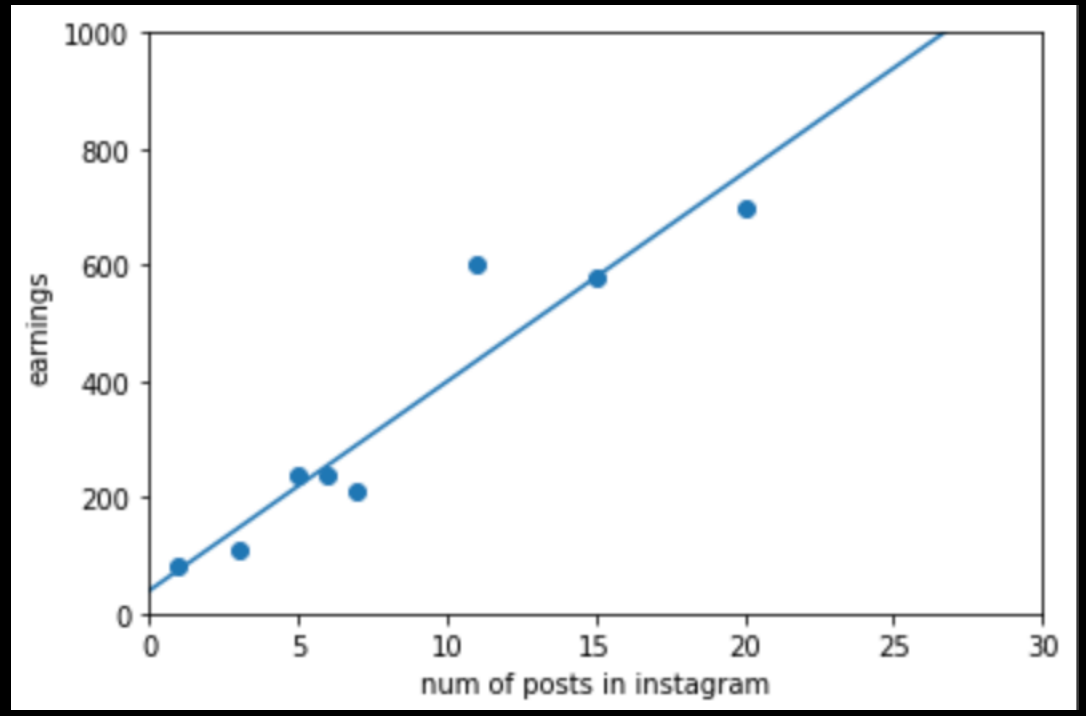
インデックス(index)

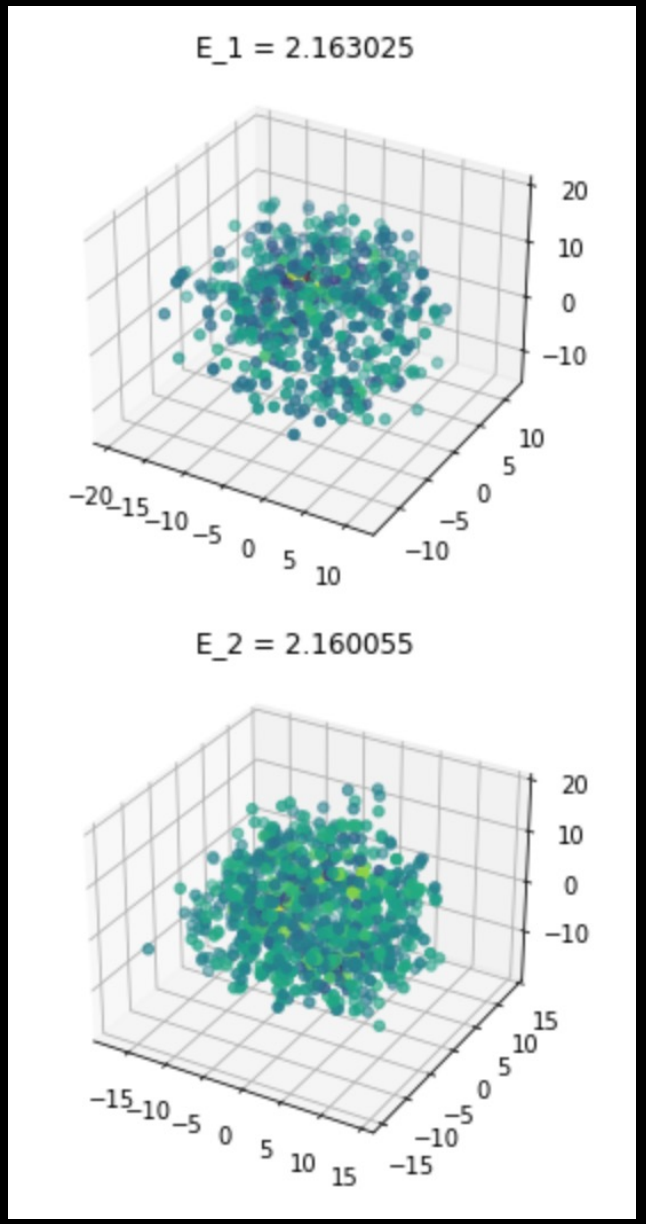
	インスタ投稿数	売り上げ
0	5	240
1	3	110
2	6	240
3	11	600
4	7	210
5	15	580
6	20	700
7	1	80

matplotlibとは

- Matplotlibとは、Python で静的、アニメーション、インタラクティブな可視化を作成するための包括的なライブラリ
- データサイエンス領域では可視化が死ぬほど重要！
- 可視化できるものばかりではないが、可視化できる時には積極的にコイツやseabornなんかをつかって可視化したい

sample

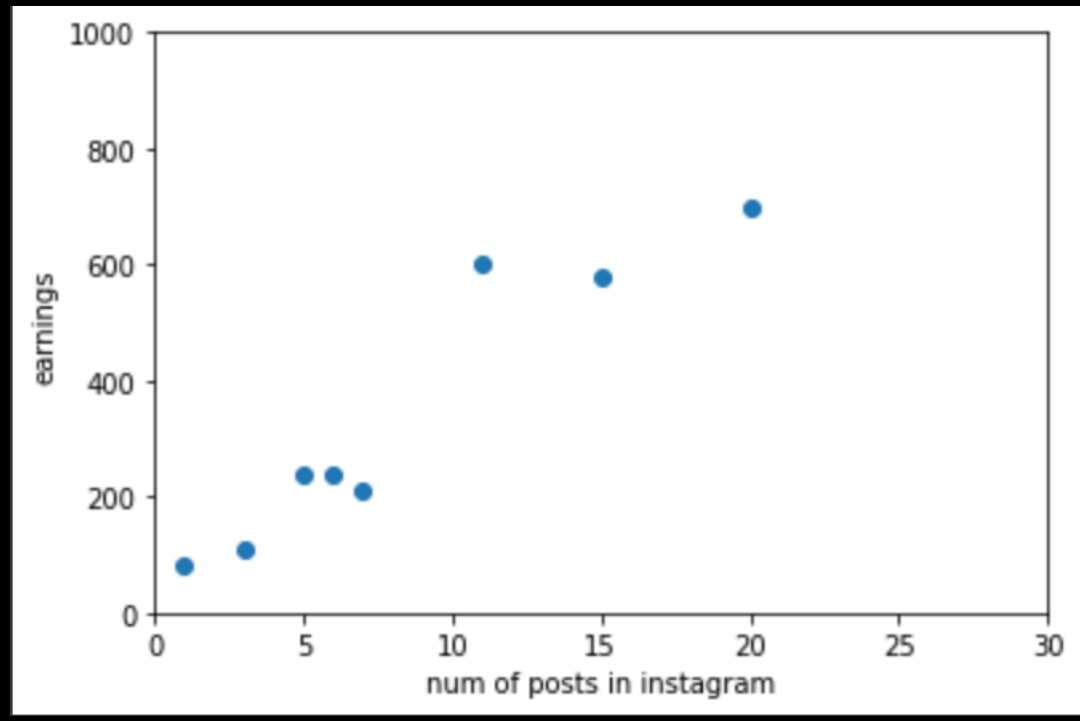

earnings
num of posts in instagram
0
200
400
600
800
1000
0
5
10
15
20
25
30


E_1 = 2.163025
E_2 = 2.160055

それでは，線形回帰やっていきましょう！

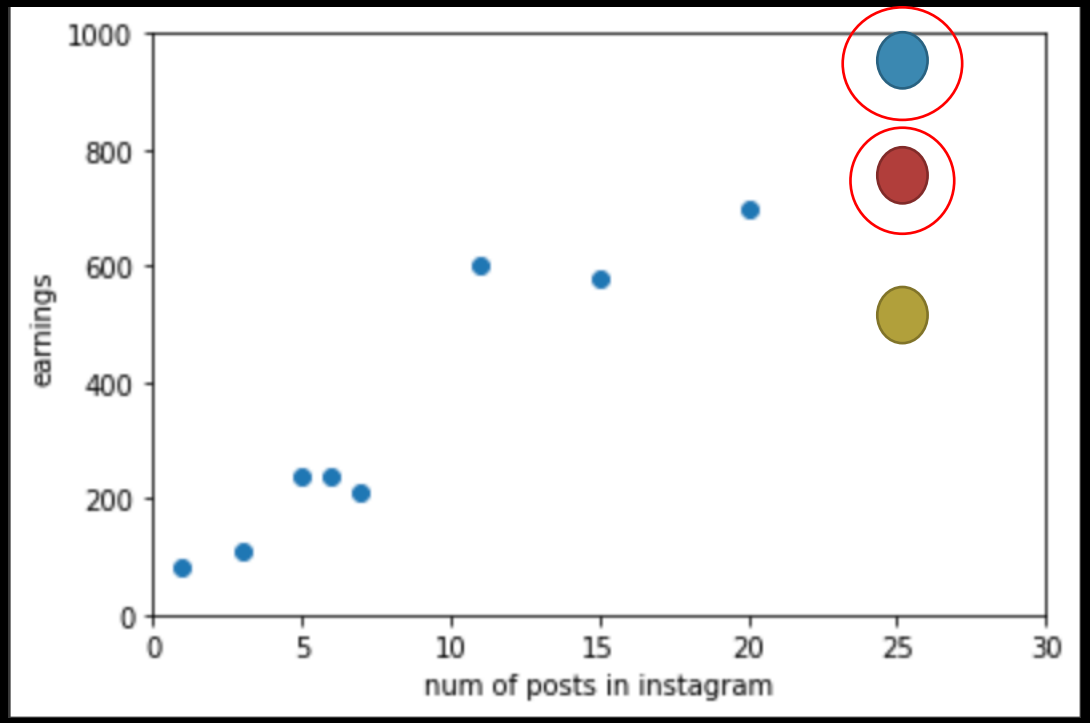
突然ですが、問題です。

説明変数	目的変数
Instagramの投稿数	1年間のケーキの売上金額（万円）
5	240
10	750
3	110
6	240
11	**600**
7	210
15	**580**
20	700
1	80



インスタの投稿数が25の時、売り上げはいくらくらいだろうか？？

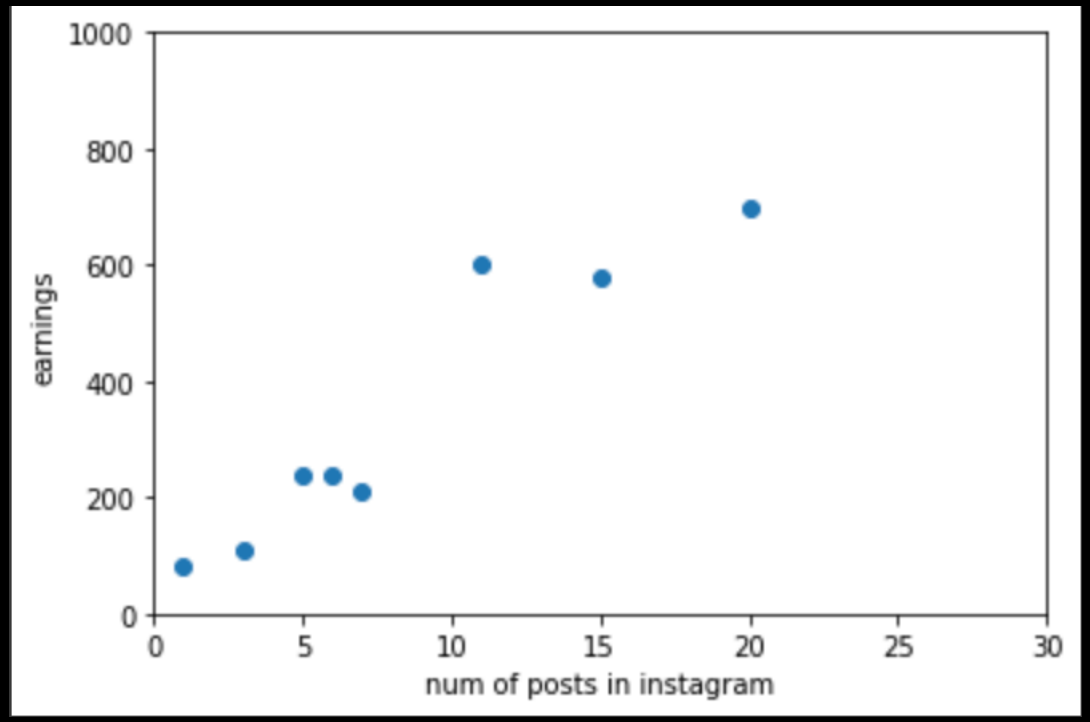
Instagramの投稿数	1年間のケーキの売上金額（万円）
5	240
10	750
3	110
6	240
11	**600**
7	210
15	**580**
20	700
1	80

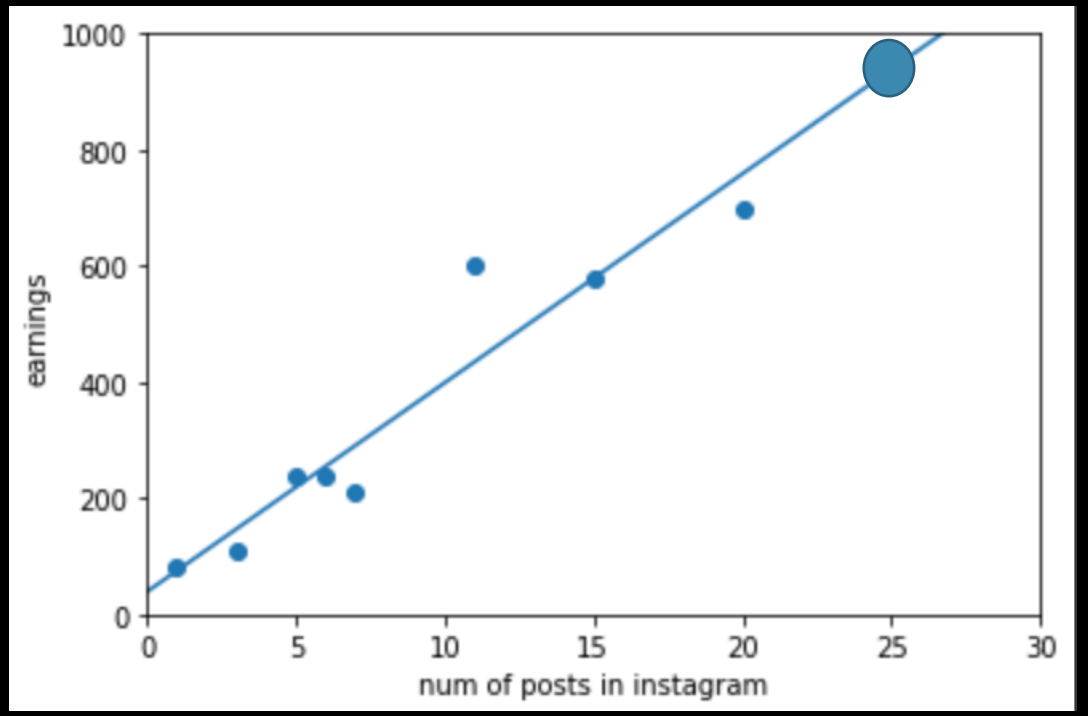


なぜ？？？？？？

おそらくこう考えたのでは？？

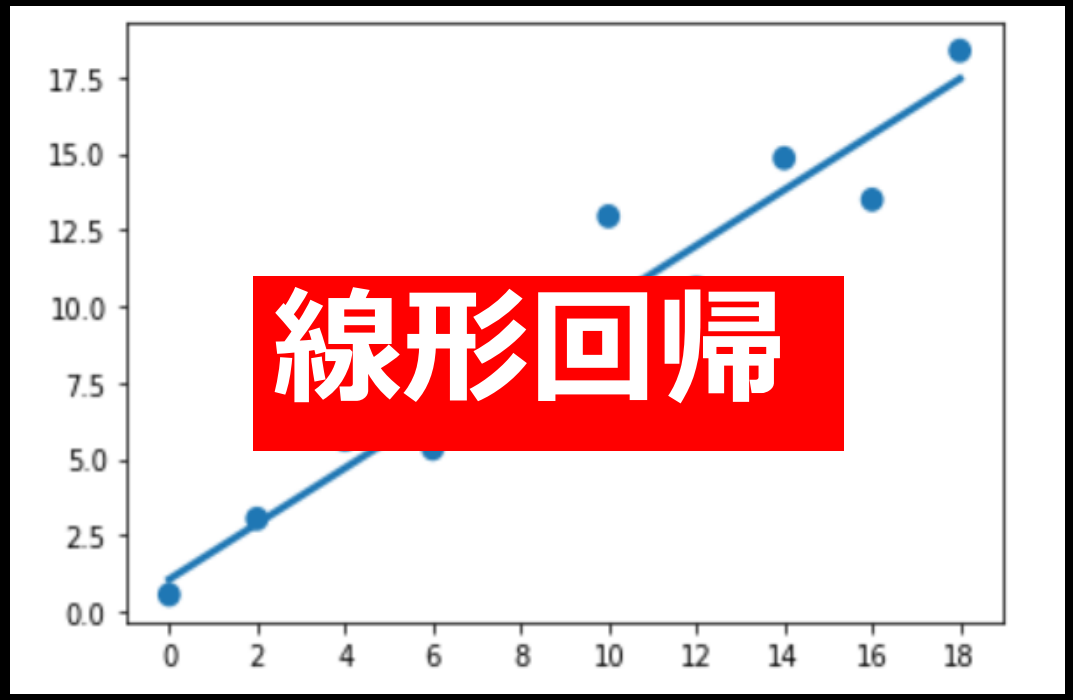
→これが線形回帰の考え方

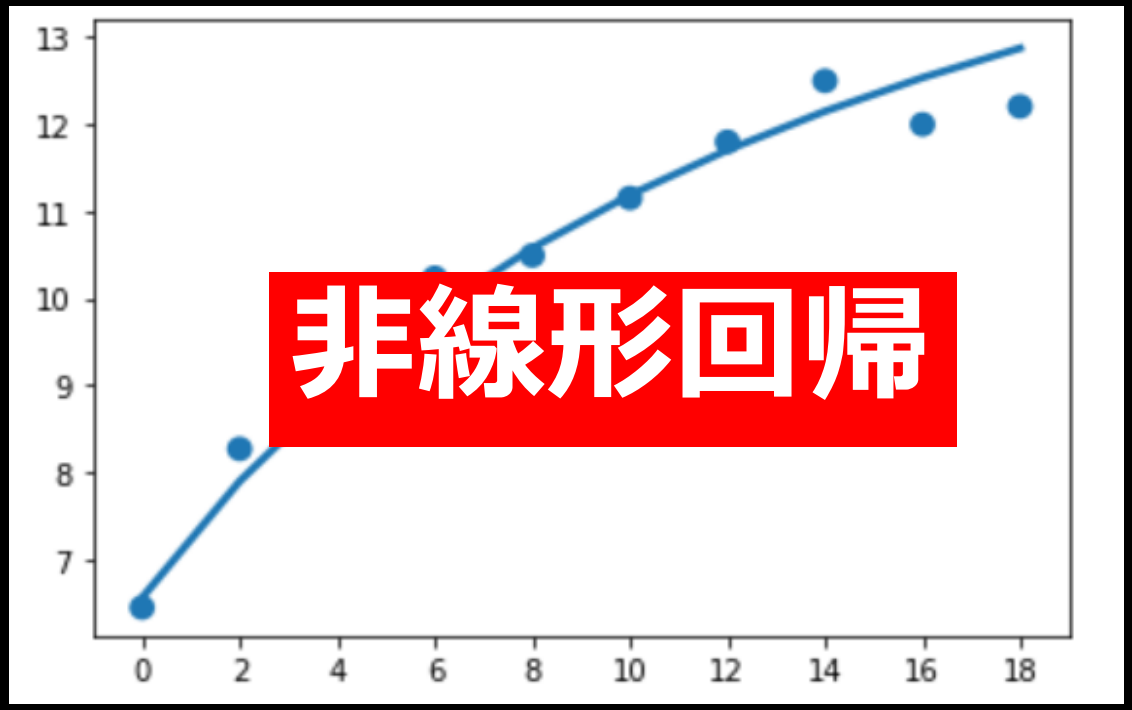




(非)線形回帰((Non-)Linear Regression)

与えられたデータ間に、線形/非線形関係を仮定してモデルを作成していくこと





線形回帰

- データの関係が

Y(出力)=aX(入力)+b

のような関係になっているという仮定に基づき、回帰係数aと切片bを求めていく

線形回帰

因みに。

- 単回帰(説明変数が1種類)
- 重回帰(説明変数が2種類以上)

と言います。

線形回帰(単回帰)

説明変数	目的変数
Instagramの投稿数	1年間のケーキの売上金額（万円）
5	240
10	750
3	110
6	240
11	**600**
7	210
15	**580**
20	700
1	80

線形回帰(単回帰)

①仮定をする
②本物と予測モデルの
誤差を考える

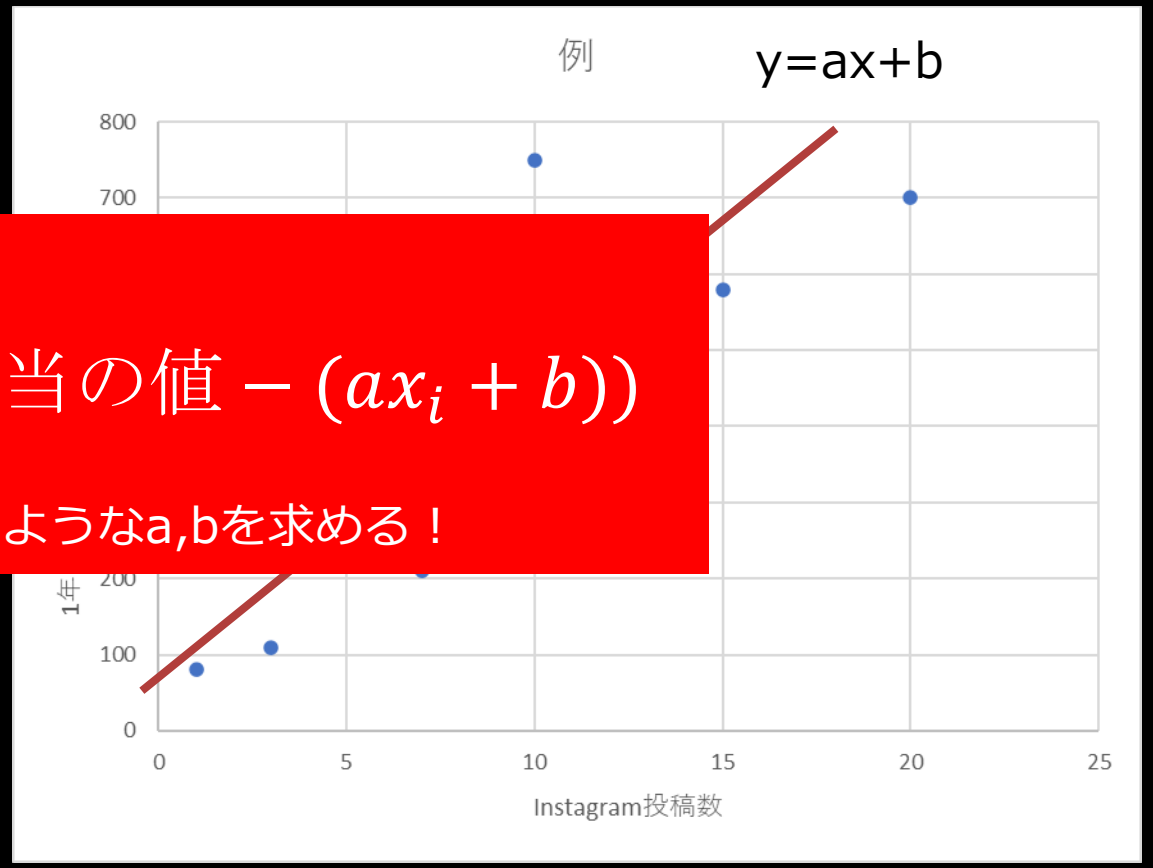
予測モデルy_1 = 1*a +
誤差$_1$ = 本当の値-y_1 =
予測モデルy_2 = 3*a +
誤差$_2$ = 本当の値- y_2 =

・
・
・

予測モデルy_9 = 20*a+b
誤差$_9$ = 本当の値-y_9 = 700-(20a+b)

データの数
$$\sum_{i}$$
(本当の値 $-(ax_i + b)$)

上の式が一番小さくなるようなa,bを求める！



線形回帰

③全体の誤差を見るために全部の誤差を足し合わせる
しかし！いろんな足し算が考えられる！
どれがいいだろう？

A:誤差$_1$＋誤差$_2$＋…＋誤差$_9$

B:|誤差$_1$|＋|誤差$_2$|＋…＋|誤差$_9$|

C:(誤差$_1$)^2＋ (誤差$_2$)^2＋… ＋(誤差$_9$)^2

線形回帰

$$J(a,b) = \sum_{k=1}^{\text{データ数}} (y_k - (aX + b))\text{^}2$$

$$(\hat{a}, \hat{b}) = argmin\, J(a,b)$$

この時のJを損失関数(cost function)という

（この損失関数を最小化する手法を最小二乗法(least square estimation)という）

線形回帰(チャレンジ問題に関わる)

最小二乗法

$$J(a,b) = \sum_{k=1}^{\text{データ数}} (y_k - (aX + b))^2$$

$$(\hat{a}, \hat{b}) = argmin\, J(a,b)$$

これの最小値は,

$$(\hat{a}, \hat{b}) = \left(\frac{S_{xy}}{S_x^2}, \bar{y} - \frac{S_{xy}}{S_x^2} \bar{x} \right)$$

で与えられる（証明は後期）

S_{xy} ：X,yの共分散

S_x^2 ：Xの分散

$\bar{y}$ ：yの平均

$\bar{x}$ ：Xの平均

線形回帰(チャレンジ問題に関わる)

最小二乗法

$$J(a,b) = \sum_{k=1}^{\text{データ数}} (y_k - (aX + b))^2$$

$$(\hat{a}, \hat{b}) = argmin\, J(a,b)$$

これの最小値は,

$$(\hat{a}, \hat{b}) = \left(\frac{S_{xy}}{S_x^2}, \bar{y} - \frac{S_{xy}}{S_x^2} \bar{x} \right)$$

で与えられる（証明は後期）

S_{xy} ：X,yの共分散

S_x^2 ：Xの分散

$\bar{y}$ ：yの平均

$\bar{x}$ ：Xの平均

線形回帰
sklearnで線形回帰を実装

線形回帰(単回帰)

まず、次のデータをnp.ndarray型の変数X,yに代入する
こんな感じ。
X=np.array([5, 10, 3, 6, 11, 7, 15, 20, 1])
y=…

説明変数	目的変数
Instagramの投稿数	1年間のケーキの売上金額（万円）
5	240
10	750
3	110
6	240
11	**600**
7	210
15	**580**
20	700
1	80

線形回帰(単回帰)

必要なライブラリのインポート

```
from sklearn.linear_model import LinearRegression
import pandas as pd
import matplotlib.pyplot as plt
```

線形回帰(単回帰)予測モデルを作る

1,インポートしたLinearRegressionクラスからインスタンスを生成

2,インスタンスのfitメソッドを使用して学習！（a,bを求めてる）

3,インスタンスのcoef_属性とintercept_属性から回帰係数と切片を確認

4,その傾きと切片を使用して、直線を描画+元データの散布図も書く

5,インスタンスのpredictメソッドを使ってインスタ投稿数が25の時の売り上げを予測

6,ついでにその予測値もグラフに描画

Tips.

クラス(型)とインスタンス(オブジェクト)

・クラスとは抽象化された何か

・インスタンスとはクラスから生成される実体(具体物)

クラスはしばしば設計図と言われることもある。

実際に扱うのは設計図ではなく、実体の方。

インスタンスには、メソッドやプロパティと呼ばれるアトリビュート(属性)がある。

線形回帰(単回帰)予測モデルを作る

california housing datasetsを用いて単回帰モデルを作ってみよう

sklearnライブラリのdatasets内に格納されているデータセットを使う

from sklearn.datasets import

fetch_california_housing

線形回帰(単回帰)予測モデルを作る

データをDataFrameに格納してみよう

df＝pd.DataFrame(data=california.data,

columns=california.feature_names)

df[“MEDV”] = california.target

データを眺めてみよう

よく見る指標：平均値、中央値、分散、標準偏差

df.mean(),df.median(),df.var(),df.std()

線形回帰(単回帰)予測モデルを作る

DataFrameからデータ列を抜き出そう

X = df[“columns_name”] -> pandas.Series型

好きなデータを抜き出して

X = np.array(X) -> np.ndarray型

と、変換し、説明変数vs目的変数の散布図を３つ作成してみよう

(subject14)

線形回帰(単回帰)予測モデルを作る

モデルは過学習しても適合不足でもいけないゆえに性能評価が必要となる

データをいくつかに分けて、それを判断していく

data -> train_data, test_data

train_dataでモデルを作ってtest_dataでモデルの性能を評価する

X_train, X_test, Y_train, Y_test = train_test_split(X, y, train_size = 0.7, test_size = 0.3, random_state = 0)

Tips.どうしてデータを分けるの？？

訓練データは参考書の問題のようなもの.

テストデータは過去問のようなもの.

高校受験や資格試験などを思い出して欲しい.

参考書でしっかりと勉強して，過去問を解いてその理解力を図るだろう.

そして自分がその資格に合格できるかなどを見極めていく.

最終的に本番のテストでしっかりと勉強したことを発揮していく.

つまり，今回モデル作成において，訓練データで学習して，テストデータでその理解力を測り，まだ知らない本番データでもそのモデルが通用するようにデータを分けて学習させた.

線形回帰(単回帰)予測モデルを作る

今できたtrain_dataでモデルを学習させてみよう

その後、test_dataで性能を見てみよう

print(model.score(X_train, Y_train))

print(model.score(X_test, Y_test))

線形回帰(単回帰)予測モデルを作る

scoreの意味

scoreは決定係数を表す.

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y})^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}$$

決定係数とは得られた回帰式で，どの程度データに当てはまっているかを指す.

次回

- 教師あり学習/重回帰分析/行列(勾配降下法)