クレジット:
UTokyo Online Educationデータマイニング入門 2018 森　純一郎

東京大学
THE UNIVERSITY OF TOKYO

CC BY NC ND

課題8 線形回帰

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

Q1 最急降下法

irisデータセットの特徴量petal_lengthとpetal_widthの関係を散布図で可視化すると以下のように2つの特徴量間に関係があることがわかります。以下では、線形回帰により特徴量petal_lengthから特徴量petal_widthを予測するような仮説関数のパラメータを学習ことを考えます。

In [ ]:
```
iris = pd.read_csv('iris.csv')
plt.figure(figsize=(7,5))
plt.xlabel('petal_length')
plt.ylabel('petal_width')
plt.scatter(iris['petal_length'], iris['petal_width'], c='blue');

### scikit-learnからデータロードする場合
# from sklearn.datasets import load_iris
# iris = load_iris()
#plt.figure(figsize=(7,5))
#plt.xlabel(iris["feature_names"][2])
#plt.ylabel(iris["feature_names"][3])
#plt.scatter(iris['data'][:,2],iris['data'][:,3],c='blue');
```

まず準備として、特徴量petal_lengthを入力X、特徴量petal_lengthを出力yとしてそれぞれを標準化します。また、入力の各データにバイアス項($x_0 = 1$)を追加するため、入力の先頭列に1を要素とする列ベクトルを挿入します。これにより、入力, 出力はデータ数をmとして以下のような行列（$m \times 2$) とベクトル($m \times 1$)になります。

$$X = \begin{pmatrix} 1 & x^{(1)} \\ 1 & x^{(2)} \\ \dots & \dots \\ 1 & x^{(m)} \end{pmatrix}$$

$x^{(i)}$ は標準化された特徴量petal_length。

$$y = \begin{pmatrix} y^{(1)} \\ \dots \\ y^{(m)} \end{pmatrix}$$

$y^{(i)}$ は標準化された特徴量petal_width。

In [ ]:
```
X=iris[['petal_length']].values
y=iris[['petal_width']].values

### scikit-learnからデータロードした場合
# X=(iris['data'][:,2])[:,np.newaxis] # 列ベクトルにする
# y=(iris['data'][:,3])[:,np.newaxis] # 列ベクトルにする

X_norm=(X-np.mean(X, axis=0))/np.std(X, axis=0) # 標準化
y_norm=(y-np.mean(y, axis=0))/np.std(y, axis=0) # 標準化

X_norm=np.insert(X_norm, 0, np.ones((1,X.shape[0]), dtype=int),
                 axis=1) # バイアス項の追加

print(X_norm[:10,:])
print(y_norm[:10])
```

以下では、最急降下法を用いた線形回帰により、訓練データセットを元に入力から出力を予測する仮説関数のパラメータを学習する `graddes` 関数を実装します。

`graddes` 関数では第1引数に入力のデータ行列（データ数(m)×次元数(n))、第2引数に入力の各データに対する出力（正解）のベクトル（データ数(m)×1）、第3引数に学習率、第4引数に学習の繰り返し（各繰り返しをエポックと呼ぶ）の回数を受け取ります。

これらの引数を元に、 `graddes` 関数ではパラメータの学習を行いエポックごとのコスト関数の値を要素とするリストと最終的なパラメータの値を要素とする配列（次元数(n)×1）を返します。

パラメータを$\theta = (\theta_0, \theta_1, \dots, \theta_{n-1})^T$

仮説関数を$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_{n-1} x_{n-1}$

入力を$X = \begin{pmatrix} x_0^{(1)} & x_1^{(1)} & \dots & x_{n-1}^{(1)} \\ \dots & \dots & \dots & \dots \\ x_0^{(m)} & x_1^{(m)} & \dots & x_{n-1}^{(m)} \end{pmatrix}$

Xにおいて$x_0^{(i)} = 1$

出力を$y = (y^{(1)}, y^{(2)}, \dots, y^{(m)})^T$

とすると、最急降下法ではコスト関数を$J(\theta) = \frac{1}{2m}\Sigma_{i=1}^{m}(h(x^{(i)}) - y^{(i)})^2$として、入力$X$の各特徴量$x_j$に対するパラメータ$\theta_j$を以下の様に更新していきます。

$\theta_j := \theta_j - \alpha\frac{\partial J(\theta)}{\partial \theta_j} = \theta_j - \frac{\alpha}{m}\Sigma_{i=1}^{m}((h(x^{(i)}) - y^{(i)})x_j^{(i)})$

パラメータ全体を以下のように一度に更新することもできます。

$\theta := \theta - \frac{\alpha}{m}X^T(X\theta - y)$

入力が1特徴量（変数）の時は、バイアス項に対するパラメータをθ_0、入力変数に対するパラメータをθ_1としてパラメータは以下の様に更新されます。

$\theta_0 := \theta_0 - \frac{\alpha}{m}\Sigma_{i=1}^{m}(h(x^{(i)}) - y^{(i)})$

$\theta_1 := \theta_1 - \frac{\alpha}{m}\Sigma_{i=1}^{m}((h(x^{(i)}) - y^{(i)})x^{(i)})$

具体的に、 `graddes` 関数では以下の手順によりパラメータの学習を行います。

- 引数 `n_iter` で指定されたエポックの回数だけ以下を繰り返す
  - すべてのm個のデータについて以下を求める
    - 入力データ$x^{(i)}$について仮説関数$h(x^{(i)})$の値を求める
    - 出力$y^{(i)}$との誤差$h(x^{(i)}) - y^{(i)}$の値を求める
  - すべてのm個のデータの誤差を用いてコスト関数$J(\theta)$の値を求め、各エポックのコスト関数の値を要素とするリスト `costs` に追加
  - すべてのm個のデータの誤差を用いて各パラメータ$\theta_j (j = 0, .., n-1)$を更新し、パラメータの値を要素とする配列 `w` を更新
    - `w[0,0]` :=(x_0に対するパラメータθ_0), `w[1,0]` :=(x_1に対するパラメータθ_1), ...

すべての繰り返しが終了したらリスト `costs` と配列 `w` を返す。上記に従って、 `graddes` 関数を完成させてください。

In [ ]:
```
def graddes(X, y, alpha, n_iter):
    m = X.shape[0] # データ数
    n =  X.shape[1] # 次元（特徴量）数

    costs=[] # エポックごとのコスト関数の値を入れるリスト
    w = np.zeros((n,1)) #  各特徴量に対するパラメータ（重み）の初期化

    for i in range(n_iter):

```

`graddes` 関数が完成したら以下のセルを実行して動作を確認してください。上記のirisデータセットの特徴量petal_lengthを入力、特徴量petal_lengthを出力とした訓練データセットを与え、学習率を0.01、学習のエポック数を100とした時の各エポックごとのコスト関数の値を示しています。パラメータの学習が進むにつれてコスト関数の値が減少していくことがわかります。

In [ ]:
```
n_iter=100
alpha=0.05
costs, w = graddes(X_norm, y_norm, alpha, n_iter)

plt.figure(figsize=(7,5))
plt.ylabel('Cost')
plt.xlabel('Iteration');
plt.plot(range(1,n_iter+1),costs);
```

100回のエポックで学習されたパラメータθ_0, θ_1を用いて特徴量petal_lengthを入力x、特徴量petal_lengthを出力yとした時の直線$y = \theta_0 + \theta_1 x$は以下のようになります。この時の最終的なコスト関数の値は約0.036、パラメータθ_0, θ_1の値はそれぞれ約 `[[-3.72239276e-16], [ 9.57057066e-01]]` となります。

In [ ]:
```
n_iter=100
alpha=0.05
costs, w = graddes(X_norm, y_norm, alpha, n_iter)
print(costs[-1])
print(w)
plt.figure(figsize=(7,5))
plt.xlabel('petal_length')
plt.ylabel('petal_width')
plt.scatter(X_norm[:,1],y_norm[:,0],c='blue')
plt.plot(X_norm[:,1], np.dot(X_norm,w)[:,0], color='red');
```

一方、5回、20回のエポックで学習されたパラメータを用いた時の直線はそれぞれ以下のようになります。エポックが進むにつれて、訓練データセットにフィッティングするようにパラメータが学習されていることがわかります。

In [ ]:
```
n_iter=5
alpha=0.05
costs, w = graddes(X_norm, y_norm, alpha, n_iter)
print(costs[-1])
print(w)
plt.figure(figsize=(7,5))
plt.xlabel('petal_length')
plt.ylabel('petal_width')
plt.scatter(X_norm[:,1],y_norm[:,0],c='blue')
plt.plot(X_norm[:,1], np.dot(X_norm,w)[:,0], color='red');
```

In [ ]:
```
n_iter=20
alpha=0.05
costs, w = graddes(X_norm, y_norm, alpha, n_iter)
print(costs[-1])
print(w)
plt.figure(figsize=(7,5))
plt.xlabel('petal_length')
plt.ylabel('petal_width')
plt.scatter(X_norm[:,1],y_norm[:,0],c='blue')
plt.plot(X_norm[:,1], np.dot(X_norm,w)[:,0], color='red');
```

Q2 正規方程式

線形回帰のパラメータは訓練データセットの入力Xと出力yに対して以下の正規方程式を解くことで解析的に求めることができます。（ただし、X^TXが正則（フルランク）であること。）

$$\theta = (X^TX)^{-1}X^Ty$$

ここで、行列Aの転置A^T、逆行列A^{-1}は `NumPy` を用いてそれぞれ以下の様に計算できます。

転置A^T

```
A.T
```

逆行列A^{-1}

```
np.linalg.inv(A)
```

正規方程式を用いて、訓練データセットを元に入力から出力を予測するパラメータを求める `normal_equation` 関数を実装してください。 `normal_equation` 関数では第1引数に入力のデータ行列（データ数(m)×次元数(n)）、第2引数に入力の各データに対する出力（正解）のベクトル（データ数(m)×1）を受け取り、パラメータθの値を要素とする配列（次元数(n)×1）（Q1のパラメータの配列 `w` と同様の形式）を返します。

In [ ]:
```
def normal_equation(X, y):
    return ### 正規方程式を用いてX,yからパラメータを求めるコード ###
```

`normal_equation` 関数が完成したら以下のセルを実行して動作を確認してください。先の最急降下法で求めたパラメータの値が正規方程式に基づくパラメータの解析解とよく近似していることがわかります。

In [ ]:
```
w=normal_equation(X_norm, y_norm)
print(w)
plt.figure(figsize=(7,5))
plt.xlabel('petal_length')
plt.ylabel('petal_width')
plt.scatter(X_norm[:,1],y_norm[:,0],c='blue')
plt.plot(X_norm[:,1], np.dot(X_norm,w)[:,0], color='red');
```

参考：scikit-learnでの線形回帰

`scikit-learn` では、以下の手順でデータからモデルの学習を行います。

- 使用するモデルのクラスの選択
- モデルのハイパーパラメータの選択とインスタンス化
- データの準備
  - 教師あり学習では、特徴量データとラベルデータを準備
  - 教師あり学習では、特徴量・ラベルデータをモデル学習用の学習データとモデル評価用のテストデータに分ける
- モデルをデータに適合（ `fit()` メソッド）
- モデルの評価
  - 教師あり学習では、 `predict()` メソッドを用いてテストデータの特徴量データからラベルデータを予測しその精度を評価を行う

以下では、回帰を行うモデルの一つである**線形回帰**（ **`LinearRegression`** ）クラスをインポートしています。 `mean_squared_error()` は平均二乗誤差によりモデルの予測精度を評価するための関数です。

データセットを訓練データ（ `X_train` , `y_train` ）とテストデータ（ `X_test` , `y_test` ）に分割し、線形回帰クラスのインスタンスの `fit()` メソッドによりモデルを訓練データに適合させています。そして、 `predict()` メソッドを用いてテストデータのpetal_lengthの値からpetal_widthの値を予測し、 `mean_squared_error()` 関数で実際のpetal_widthの値（ `y_test` ）と比較して予測精度の評価を行なっています。

In [ ]:
```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

iris = load_iris()
X=(iris['data'][:,2])[:,np.newaxis]
y=(iris['data'][:,3])[:,np.newaxis]

# 訓練データとテストデータ
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3,
                                                    random_state=0)

# 標準化
sc=StandardScaler()
sc.fit(X_train)
norm_X_train=sc.transform(X_train)
norm_X_test=sc.transform(X_test)

sc.fit(y_train)
norm_y_train=sc.transform(y_train)
norm_y_test=sc.transform(y_test)

# バイアス項の追加
norm_X_train=np.insert(norm_X_train, 0,
                       np.ones((norm_X_train.shape[0],1), dtype=int).T,
                       axis =1)
norm_X_test=np.insert(norm_X_test, 0,
                      np.ones((norm_X_test.shape[0],1), dtype=int).T,
                      axis =1)

# 学習モデルの訓練データへの適合
model=LinearRegression()
model.fit(norm_X_train, norm_y_train)

# 学習モデルを用いてテストデータから予測と評価
y_predicted=model.predict(norm_X_test)
print(mean_squared_error(norm_y_test,y_predicted))

plt.figure(figsize=(7,5))
plt.xlabel(iris["feature_names"][2])
plt.ylabel(iris["feature_names"][3])
plt.scatter(norm_X_train[:,1], norm_y_train[:,0],c='blue')
plt.plot(norm_X_test[:,1], y_predicted[:,0], color='red');
```