機会学習　実装演習

1.回帰分析

[1]:
```
#pandasのインポート
import pandas as pd

#ライブラリの宣言
from sklearn.ensemble import RandomForestRegressor

#データの読み込み
boston = pd.read_csv('boston.csv')
```

[2]:
```
#データを説明変数Xと目的変数yに分ける.
X_boston = boston.drop("house_price",axis=1)
y_boston = boston["house_price"]
```

[3]:
```
#ライブラリの宣言
from sklearn.model_selection import train_test_split

#全データを説明変数と目的変数に分ける
X_boston = boston.drop("house_price",axis=1)
y_boston = boston["house_price"]

#説明変数と目的変数をそれぞれ、trainのX,testのX,trainのy,testのyに分ける
X_boston_train , X_boston_test , y_boston_train , y_boston_test = train_test_split(X_boston , y_boston , random_state=0)

#trainとtestの形を確認して分けられていることを確認
print(boston.shape)
print(X_boston_train.shape , X_boston_test.shape , y_boston_train.shape , y_boston_test.shape)
```

```
(506, 14)
(379, 13) (127, 13) (379,) (127,)
```

[4]:
```
# モデルを定義して、学習する.
model_boston = RandomForestRegressor(n_estimators=100,random_state=0)
model_boston.fit(X_boston_train,y_boston_train)

#モデルの精度を出力
print("学習用データのスコアは",model_boston.score(X_boston_train,y_boston_train))
print("検証用データのスコアは",model_boston.score(X_boston_test,y_boston_test))
```

```
学習用データのスコアは 0.9824381817739267
検証用データのスコアは 0.7952684623500126
```

2. ロジスティック回帰

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# ロジスティック回帰モデルインポート
from sklearn.linear_model import LogisticRegression

import warnings
warnings.filterwarnings("ignore")

from sklearn.model_selection import train_test_split
```

[2]:
```
#irisデータの読み込み
from sklearn.datasets import load_iris
iris = load_iris()
```

[3]:
```
#説明変数と目的変数を設定
iris_X = iris.data
iris_Y = iris.target

#データを訓練用とテスト用に分割
iris_X_train,iris_X_test,iris_Y_train,iris_Y_test = train_test_split(iris_X,iris_Y,random_state=0)
```

[4]:
```
# 使用するモデルを指定
logreg = LogisticRegression()

# モデルの訓練
logreg.fit(iris_X_train, iris_Y_train)
```

[4]:
```
LogisticRegression()
```

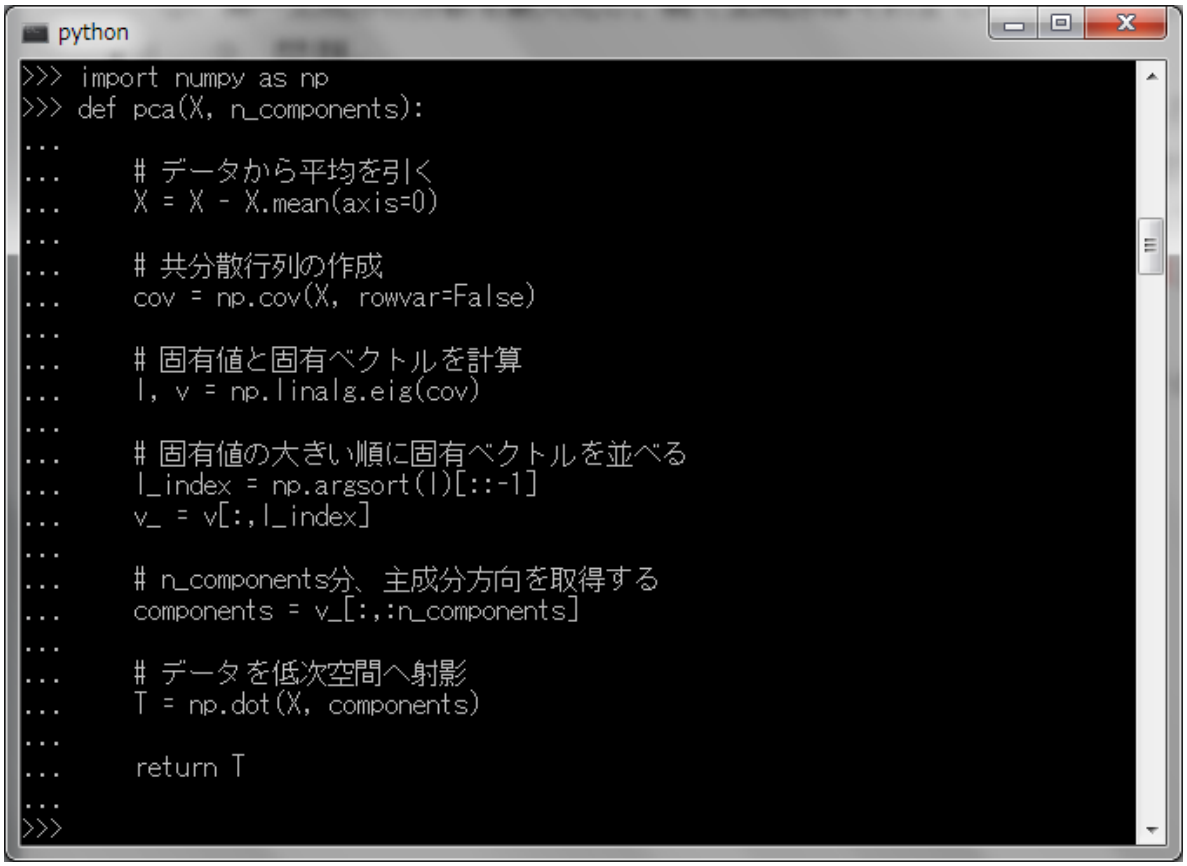
[5]:
```
# テストデータでモデルを検証
Y_pred = logreg.predict(iris_X_test)
```

[6]:
```
# 精度確認
from sklearn.metrics import accuracy_score

# 予測結果を確認する
acc_logreg = round(accuracy_score(Y_pred, iris_Y_test) * 100, 2)
print(acc_logreg)
```

```
97.37
```

3．主成分分析


python
>>> import numpy as np
>>> def pca(X, n_components):
...
...   # データから平均を引く
...   X = X - X.mean(axis=0)
...
...   # 共分散行列の作成
...   cov = np.cov(X, rowvar=False)
...
...   # 固有値と固有ベクトルを計算
...   l, v = np.linalg.eig(cov)
...
...   # 固有値の大きい順に固有ベクトルを並べる
...   l_index = np.argsort(l)[::-1]
...   v_ = v[:,l_index]
...
...   # n_components分、主成分方向を取得する
...   components = v_[:,:n_components]
...
...   # データを低次空間へ射影
...   T = np.dot(X, components)
...
...   return T
...
>>>

4 . K-近傍法(KNN)

[7]:
```
from sklearn.neighbors import KNeighborsClassifier

#n_neighbors=3を変えることでKを変えられる
kn = KNeighborsClassifier(n_neighbors=3)
kn.fit(iris_X_train,iris_Y_train)
```

[7]:
```
KNeighborsClassifier(n_neighbors=3)
```

[8]:
```
#性能評価
kn_score = kn.score(iris_X_test,iris_Y_test)
print('3-nearest neighbor score:{}'.format(kn_score))
```

```
3-nearest neighbor score:0.9736842105263158
```

[9]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold

#分割交差検証
stratifiedkfold = StratifiedKFold(n_splits=5)

#交差検証
stkfold_scores = cross_val_score(kn,iris.data,iris.target,cv=stratifiedkfold)
print('stratifiedkfold Cross-Validation scores:{}'.format(stkfold_scores))
print('stratifiedkfold Average score:{}'.format(np.mean(stkfold_scores)))
```

```
stratifiedkfold Cross-Validation scores:[0.96666667 0.96666667 0.93333333 0.96666667 1.        ]
stratifiedkfold Average score:0.9666666666666668
```

5. Vector Machine(SVM：Suppoort Vector Machine)

[10]:
```
#データの設定
iris = load_iris()
x = iris.data[:,[2,3]]#petal_lengthとpetal_width
y = iris.target
X_train,X_test,Y_train,Y_test = train_test_split(x,y,random_state=0)
```

[11]:
```
from sklearn.svm import LinearSVC
#線形SVM
svml1 = LinearSVC(random_state=0)
svml1.fit(X_train,Y_train)

#性能評価
svml1_score = svml1.score(X_test,Y_test)
print('linear SVM score:{}'.format(svml1_score))
```

```
linear SVM score:0.8157894736842105
```

[12]:
```
#CV
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold

#分割交差検証
stratifiedkfold = StratifiedKFold(n_splits=5)

svml1_scores = cross_val_score(svml1,iris.data[:,[2,3]],iris.target,cv=stratifiedkfold)
print('linear SVM Cross-Validation scores:{}'.format(svml1_scores))
print('linear SVM Average score:{}'.format(np.mean(svml1_scores)))
```

```
linear SVM Cross-Validation scores:[0.96666667 0.96666667 0.9        0.9        0.96666667]
linear SVM Average score:0.9400000000000001
```

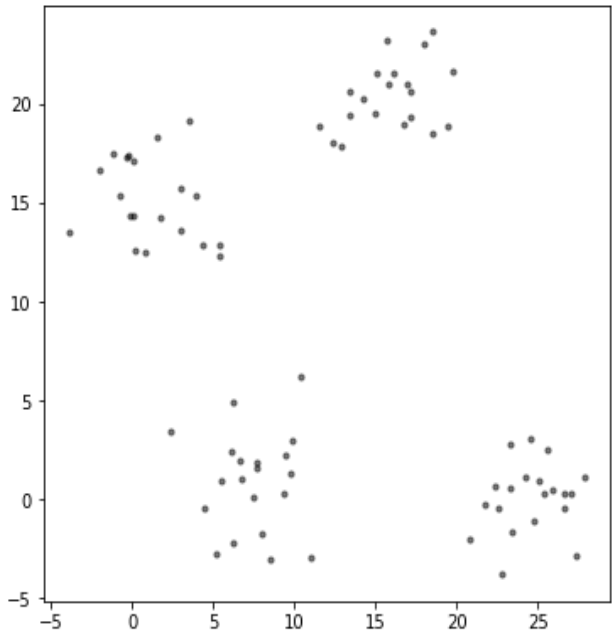
6. k-means

[14]:
```
def kmeans(X,K,centers,n_iter):
    #Xのサンプル数だけ空のラベルを作成
    idx = np.zeros(X.shape[0])
    for _ in range(n_iter):
        #データ点と中心点の距離を計算し、一番近い中心点のインデックス(クラスター番号)を返す。
        for i in range(X.shape[0]):
            idx[i] = np.argmin(np.sum((X[i,:] - centers)**2,axis=1))
        #重心を計算して中心点を移動
        for k in range(K):
            centers[k,:] = X[idx==k,:].mean(axis=0)

    return idx,centers
```

[15]:
```
#データの生成
np.random.seed(123)
x1 = np.r_[np.random.normal(size=20,loc=1,scale=2),np.random.normal(size=20,loc=8,scale=2)
           ,np.random.normal(size=20,loc=15,scale=2),np.random.normal(size=20,loc=25,scale=2)]
x2 = np.r_[np.random.normal(size=20,loc=15,scale=2),np.random.normal(size=20,loc=1,scale=2)
           ,np.random.normal(size=20,loc=20,scale=2),np.random.normal(size=20,loc=0,scale=2)]
X = np.c_[x1,x2]

#可視化
plt.figure(figsize=(6,6))
plt.scatter(X[:,0],X[:,1],c="black",s=10,alpha=0.5)
plt.show()
```



[16]:
```
#k-means法で4グループにクラスタリング
K=4
centers = np.array([[0,5],[5,0],[10,15],[20,10]])
n_iter = 4

plt.figure(figsize=(8,8))

for j in range(n_iter):

    idx = kmeans(X, K, centers, j)[0]
    centers = kmeans(X, K, centers, j)[1]
    data = pd.DataFrame(X, columns=["X","Y"])
    data["idx"] = idx

    data0 = data[data.idx==0]
    data1 = data[data.idx==1]
    data2 = data[data.idx==2]
    data3 = data[data.idx==3]

    plt.subplot(2, 2, j+1)
    plt.scatter(data0.X, data0.Y, color="r", s=10, alpha=0.5)
    plt.scatter(data1.X, data1.Y, color="b", s=10, alpha=0.5)
    plt.scatter(data2.X, data2.Y, color="g", s=10, alpha=0.5)
    plt.scatter(data3.X, data3.Y, color="orange", s=10, alpha=0.5)
    plt.scatter(centers[:,0], centers[:,1], color=["r","b","g","orange"])

plt.show()
```