Pythonではじめる機械学習

３章　教師なし学習と前処理

3.3　前処理とスケール変換

3.3.1　さまざまな前処理

3.3.2　データ変換の適用

3.3.3　訓練データとテストデータを同じように変換する

3.3.4　教師あり学習における前処理の効果

15T4061L　南濱　篤

3.3　前処理とスケール変換

ニューラルネットワークやSVMなどのアルゴリズムに適したデータ表現に変換することが広く行われている。

- 例：特徴量ごとにスケールを変更してずらす方法

3.3.1　さまざまな前処理

標準的なレンジに変換する方法

- StandardScaler
- RobustScaler
- MinMaxScaler
- Normalizer

外れ値(outliner)

極端に他の値と異なる値

3.3.1　さまざまな前処理

- StandardScaler

個々の特徴量の平均０、分散１に特徴量を変換
特徴量の最大・最小値がある範囲内に入ることを保証しない

- RobustScaler

中央値と四分位数を用いての変換
外れ値を無視して変換が行われる

3.3.1　さまざまな前処理

- MinMaxScaler

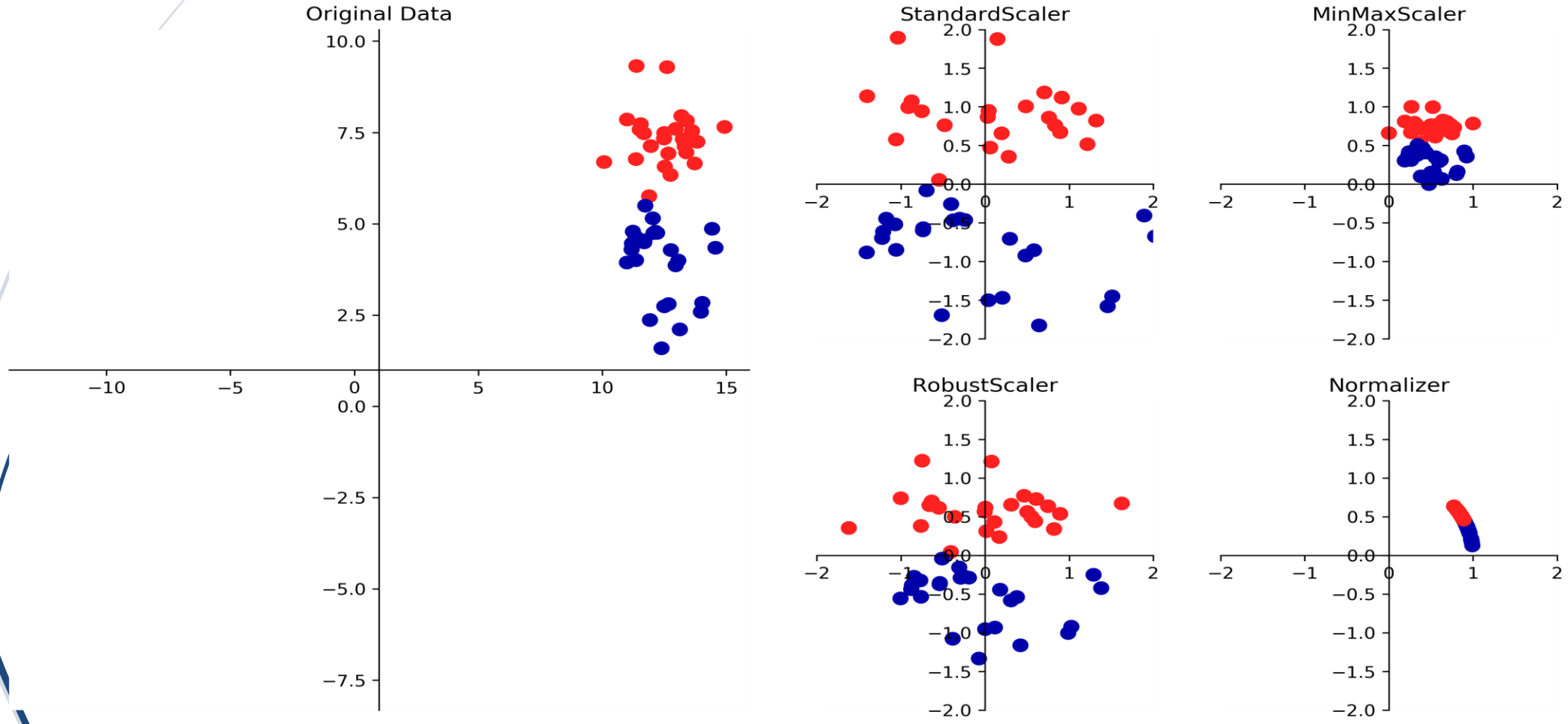
データが０から１の間になるよう変換

- Normalizer

特徴量ベクトルがユークリッド長１になるよう変換
　⇒半径１の円、もしくは超球面に投射
特徴ベクトルの方向(もしくは角度)だけが問題の場合有効

3.3.1　さまざまな前処理

実行例：



3.3.2　データ変換の適用

scikit-learnを用いて実際に適用する

- スケール変換器などの前処理手法は
- 　教師あり学習アルゴリズム適用前に行う
- カーネル法を用いたSVMをcancerデータセットに適用し、
- 　MinMaxScalerを前処理に用いる

3.3.2　データ変換の適用

データロードを行い、訓練セットとテストセットに分割

```
In[2]:
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
cancer = load_breast_cancer()
X_train,X_test,y_train,y_test = train_test_split(cancer.data, cancer.target, random_state=1)
(X_train.shape)
(X_test.shape)
```

```
Out[2]:
(426, 30)
(143, 30)
```

出力から30の測定結果を表す569のデータポイントを
訓練セット426、テストセット143サンプルに分割

3.3.2　データ変換の適用

前処理を実装したクラスをインポートし、インスタンス生成

```
In[3]:
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
```

fitメソッドを訓練データに適用し、スケール変換器を適合
MinMaxScalerの場合、
　fitメソッドは訓練データ中の各特徴量の最大値と最小値を計算

```
In[4]:
scaler.fit(X_train)
```

```
Out[4]:
MinMaxScaler(copy=True, feature_range=(0, 1))
```

3.3.2　データ変換の適用

訓練データをスケール変換するには、transformメソッドを使用

```
In[5]:
X_train_scaled = scaler.transform(X_train)
print("transformed shape: {}".format(X_train_scaled.shape))
print("per-feature minimum before scaling:¥n
    {}".format(X_train.min(axis=0)))
print("per-feature maximum before scaling:¥n
    {}".format(X_train.max(axis=0)))
print("per-feature minimum after scaling:¥n {}".format(
    X_train_scaled.min(axis=0)))
print("per-feature maximum after scaling:¥n {}".format(
    X_train_scaled.max(axis=0)))
```

3.3.2　データ変換の適用

```
Out[5]:
transformed shape: (426, 30)
per-feature minimum before scaling:
 [   6.981    9.71    43.79   143.5      0.053    0.019    0.       0.
    0.106    0.05     0.115    0.36     0.757    6.802    0.002    0.002
    0.       0.       0.01     0.001    7.93    12.02    50.41   185.2
    0.071    0.027    0.       0.       0.157    0.055]
per-feature maximum before scaling:
 [   28.11     39.28    188.5    2501.       0.163     0.287     0.427
     0.201     0.304     0.096     2.873     4.885    21.98    542.2
     0.031     0.135     0.396     0.053     0.061     0.03     36.04
    49.54    251.2    4254.       0.223     0.938     1.17      0.291
     0.577     0.149]
per-feature minimum after scaling:
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
   0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
per-feature maximum after scaling:
 [ 1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.
   1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.  1.]
```

3.3.2　データ変換の適用

SVM をスケール変換したデータに適用するために、
テストセットも変換する必要がある

```
In[6]:
X_test_scaled = scaler.transform(X_test)

print("per-feature minimum after
    scaling:¥n{}".format(X_test_scaled.min(axis=0)))
print("per-feature maximum after
    scaling:¥n{}".format(X_test_scaled.max(axis=0)))
```

```
Out[6]:
per-feature minimum after scaling:
[ 0.034  0.023  0.031  0.011  0.141  0.044  0.     0.      0.154 -0.006
 -0.001  0.006  0.004  0.001  0.039  0.011  0.     0.     -0.032  0.007
  0.027  0.058  0.02   0.009  0.109  0.026  0.     0.     -0.     -0.002]
per-feature maximum after scaling:
[ 0.958  0.815  0.956  0.894  0.811  1.22   0.88   0.933  0.932  1.037
  0.427  0.498  0.441  0.284  0.487  0.739  0.767  0.629  1.337  0.391
  0.896  0.793  0.849  0.745  0.915  1.132  1.07   0.924  1.205  1.631]
```

3.3.3 訓練データとテストデータを同じように変換する

教師ありモデルをテストセットに対して適用する際
テストセットと訓練セットと全く同じように変換する

```
In[7]:
from sklearn.datasets import make_blobs
# 合成データ作成
X, _ = make_blobs(n_samples=50, centers=5, random_state=4, cluster_std=2)
# 訓練セットとデータセットに分割
X_train, X_test = train_test_split(X, random_state=5, test_size=.1)

# 訓練セットとテストセットをプロット
fig, axes = plt.subplots(1, 3, figsize=(13, 4))
axes[0].scatter(X_train[:, 0], X_train[:, 1],
                c=mglearn.cm2(0), label="Training set", s=60)
axes[0].scatter(X_test[:, 0], X_test[:, 1], marker='^',
                c=mglearn.cm2(1), label="Test set", s=60)
axes[0].legend(loc='upper left')
axes[0].set_title("Original Data")
```

3.3.3 訓練データとテストデータを同じように変換する

```
In[7]:
# MinMaxScalerでデータをスケール変換
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# スケール変換されたデータの特性を可視化
axes[1].scatter(X_train_scaled[:, 0], X_train_scaled[:, 1],
                c=mglearn.cm2(0), label="Training set", s=60)
axes[1].scatter(X_test_scaled[:, 0], X_test_scaled[:, 1], marker='^',
                c=mglearn.cm2(1), label="Test set", s=60)
axes[1].set_title("Scaled Data")
```

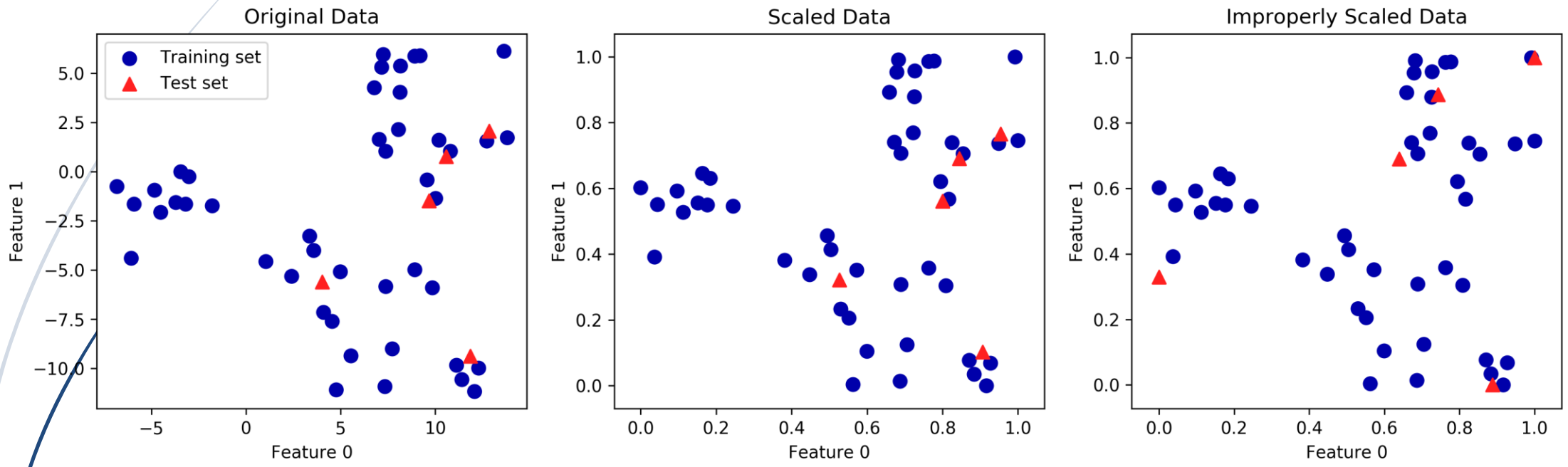
3.3.3 訓練データとテストデータを同じように変換する

```
In[7]:
# テストセットを訓練セットとは別にスケール変換
# 最小値と最大値が0,1になる。ここではわざとやっている
# 実際にはやってはいけない！
test_scaler = MinMaxScaler()
test_scaler.fit(X_test)
X_test_scaled_badly = test_scaler.transform(X_test)

# 間違ってスケール変換されたデータを可視化
axes[2].scatter(X_train_scaled[:, 0], X_train_scaled[:, 1],
                c=mglearn.cm2(0), label="training set", s=60)
axes[2].scatter(X_test_scaled_badly[:, 0], X_test_scaled_badly[:, 1],
                marker='^', c=mglearn.cm2(1), label="test set", s=60)
axes[2].set_title("Improperly Scaled Data")

for ax in axes:
    ax.set_xlabel("Feature 0")
    ax.set_ylabel("Feature 1")
fig.tight_layout()
```

3.3.3 訓練データとテストデータを同じように変換する



3.3.4 教師あり学習における前処理の効果

SVCの学習び対するMI n MaxScalerの効果を確認する

```
In[8]:
from sklearn.svm import SVC

X_train, X_test, y_train, y_test =
train_test_split(cancer.data,cancer.target,random_state=0)

svm = SVC(C=100)
svm.fit(X_train, y_train)
print("Test set accuracy: {:.2f}".format(svm.score(X_test, y_test)))
```

```
Out[8]:
Test set accuracy: 0.63
```

3.3.4 教師あり学習における前処理の効果

SVCに掛ける前に、MinMaxScalerを使ってスケール変換

```
In[9]:
scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

svm.fit(X_train_scaled, y_train)

print("Scaled test set accuracy: {:.2f}".format(
    svm.score(X_test_scaled, y_test)))
```

```
Out[9]:
Scaled test set accuracy: 0.97
```

3.3.4 教師あり学習における前処理の効果

SVCに掛ける前に、StandardScalerを使ってスケール変換

```
In[10]:
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

svm.fit(X_train_scaled, y_train)

print("SVM test accuracy: {:.2f}".format(svm.score(X_test_scaled,
y_test)))
```

```
Out[10]:
SVM test accuracy: 0.96
```