Feature Scaling

- Feature scaling through standardization (or Z-score normalization) can be an important preprocessing step for many machine learning algorithms.
- Standardization involves rescaling the features such that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one.
    - While many algorithms (such as SVM, K-nearest neighbors, and logistic regression) require features to be normalized, intuitively we can think of Principle Component Analysis (PCA) as being a prime example of when normalization is important.
    - In PCA we are interested in the components that maximize the variance.
    - If one component (e.g. human height) varies less than another (e.g. weight) because of their respective scales (meters vs. kilos), PCA might determine that the direction of maximal variance more closely corresponds with the 'weight' axis, if those features are not scaled.
    - As a change in height of one meter can be considered much more important than the change in weight of one kilogram, this is clearly incorrect.

In [1]:
```
from sklearn.model_selection import train_test_split

from sklearn.pipeline import make_pipeline

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

from sklearn.naive_bayes import GaussianNB

from sklearn import metrics

from sklearn.datasets import load_wine

import matplotlib.pyplot as plt
```

In [2]:
```
print("Features")
print(load_wine().feature_names)

print("\nClasses")
print(load_wine().target_names)
```

```
Features
['alcohol', 'malic_acid', 'ash', 'alcalinity_of_ash', 'magnesium', 'total_phenols', 'flavanoids', 'nonflavanoid
_phenols', 'proanthocyanins', 'color_intensity', 'hue', 'od280/od315_of_diluted_wines', 'proline']

Classes
['class_0' 'class_1' 'class_2']
```

In [3]:
```
features, target = load_wine(return_X_y=True)
```

In [4]:
```
features.shape
```

Out[4]:
```
(178, 13)
```

In [5]:
```
target.shape
```

Out[5]:
```
(178,)
```

In [6]:
```
X_train, X_test, y_train, y_test = train_test_split(
    features, target, test_size=0.30, random_state=0)
```

In [7]:
```
# Fit to data and predict using pipelined GNB and PCA.

unscaled_clf = make_pipeline(PCA(n_components=2), GaussianNB())
unscaled_clf.fit(X_train, y_train)

pred_test = unscaled_clf.predict(X_test)
```

In [8]:
```
# Fit to data and predict using pipelined scaling, GNB and PCA.

std_clf = make_pipeline(StandardScaler(),
                        PCA(n_components=2), GaussianNB())
std_clf.fit(X_train, y_train)

pred_test_std = std_clf.predict(X_test)
```

In [9]:
```
# Show prediction accuracies in scaled and unscaled data.

print('\nPrediction accuracy for the normal test dataset with PCA')
print('{:.2%}\n'.format(metrics.accuracy_score(y_test, pred_test)))

print('\nPrediction accuracy for the standardized test dataset with PCA')
print('{:.2%}\n'.format(metrics.accuracy_score(y_test, pred_test_std)))
```

```

Prediction accuracy for the normal test dataset with PCA
79.63%


Prediction accuracy for the standardized test dataset with PCA
98.15%

```

In [10]:
```
# Extract PCA from pipeline

pca = unscaled_clf.named_steps['pca']

pca_std = std_clf.named_steps['pca']
```

In [11]:
```
pca.components_
```

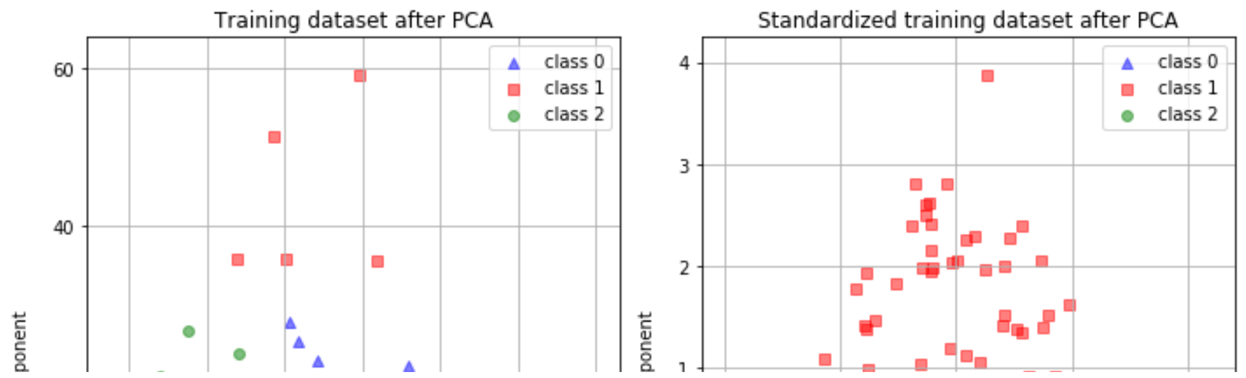
Out[11]:
```
array([[ 1.62835773e-03, -7.39332227e-04,  1.41595032e-04,
        -5.57549259e-03,  1.48399207e-02,  1.08561838e-03,
         1.74941031e-03, -1.45501403e-04,  6.83214469e-04,
         2.38657234e-03,  1.85519152e-04,  7.52190029e-04,
         9.99867216e-01],
       [ 9.92679525e-04,  3.72134412e-04,  2.81288662e-03,
         1.76663706e-02,  9.99676549e-01, -5.27746246e-04,
        -2.25866304e-03, -1.79046049e-03,  7.20197408e-03,
         5.05121401e-03, -3.53071495e-04, -4.69256202e-03,
        -1.47494368e-02]])
```

In [12]:
```
# Show first principal components

print('\nPC 1 without scaling:\n', pca.components_[0])

print('\nPC 1 with scaling:\n', pca_std.components_[0])
```

```
PC 1 without scaling:
 [ 1.62835773e-03 -7.39332227e-04  1.41595032e-04 -5.57549259e-03
  1.48399207e-02  1.08561838e-03  1.74941031e-03 -1.45501403e-04
  6.83214469e-04  2.38657234e-03  1.85519152e-04  7.52190029e-04
  9.99867216e-01]

PC 1 with scaling:
 [ 0.14669811 -0.24224554 -0.02993442 -0.25519002  0.12079772  0.38934455
  0.42326486 -0.30634956  0.30572219 -0.09869191  0.30032535  0.36821154
  0.29259713]
```

In [13]:
```
# Use PCA without and with scale on X_train data for visualization.

X_train_transformed = pca.transform(X_train)

scaler = std_clf.named_steps['standardscaler']
X_train_std_transformed = pca_std.transform(scaler.transform(X_train))
```

In [14]:
```
# visualize standardized vs. untouched dataset with PCA performed

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(10, 7))


for l, c, m in zip(range(0, 3), ('blue', 'red', 'green'), ('^', 's', 'o')):
    ax1.scatter(X_train_transformed[y_train == l, 0],
                X_train_transformed[y_train == l, 1],
                color=c,
                label='class %s' % l,
                alpha=0.5,
                marker=m
                )

for l, c, m in zip(range(0, 3), ('blue', 'red', 'green'), ('^', 's', 'o')):
    ax2.scatter(X_train_std_transformed[y_train == l, 0],
                X_train_std_transformed[y_train == l, 1],
                color=c,
                label='class %s' % l,
                alpha=0.5,
                marker=m
                )

ax1.set_title('Training dataset after PCA')
ax2.set_title('Standardized training dataset after PCA')

for ax in (ax1, ax2):
    ax.set_xlabel('1st principal component')
    ax.set_ylabel('2nd principal component')
    ax.legend(loc='upper right')
    ax.grid()

plt.tight_layout()
```



```
In [ ]:
```