FEATURE SCALING WITH S

10TH MAY 2017 | IN PYTHON | BY

Feature Scaling with scikit-l

In this post we explore 3 methods of feature scaling that are imple

- `StandardScaler`
- `MinMaxScaler`
- `RobustScaler`
- `Normalizer`

Standard Scaler

The `StandardScaler` assumes your data is normally distributed wit
the distribution is now centred around 0, with a standard deviation

The mean and standard deviation are calculated for the feature anc

$$\frac{x_i - mean(x)}{stdev(x)}$$

If data is not normally distributed, this is not the best scaler to use.

Let’s take a look at it in action:

In [1]:

```
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
matplotlib.style.use('ggplot')
```

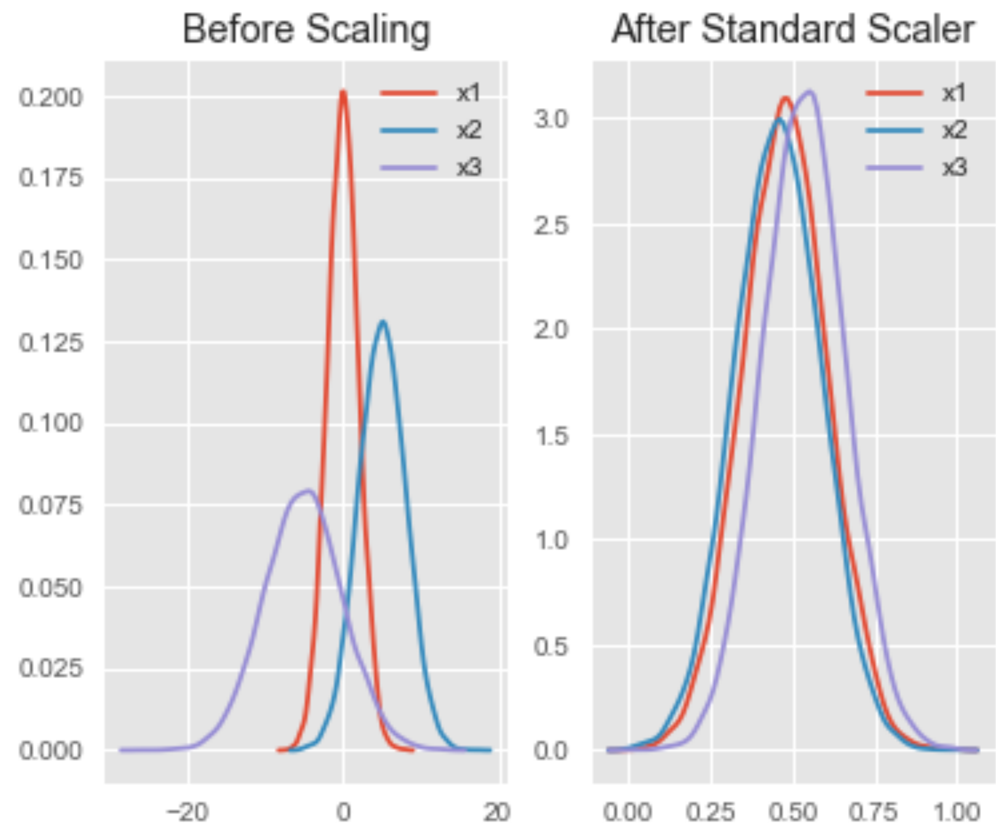
In [2]:

```
np.random.seed(1)
df = pd.DataFrame({
    'x1': np.random.normal(0, 2, 10000),
    'x2': np.random.normal(5, 3, 10000),
    'x3': np.random.normal(-5, 5, 10000)
})

scaler = preprocessing.StandardScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2',

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6, 5))

ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)
ax2.set_title('After Standard Scaler')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
plt.show()
```



All features are now on the same scale relative to one another.

Min-Max Scaler

The `MinMaxScaler` is the probably the most famous scaling algorit
feature:

$$\frac{x_i - min(x)}{max(x) - min(x)}$$

It essentially shrinks the range such that the range is now between

This scaler works better for cases in which the standard scaler migh
Gaussian or the standard deviation is very small, the min-max scale

However, it is sensitive to outliers, so if there are outliers in the dat
`Scaler` below.

For now, let's see the `min-max` scaler in action

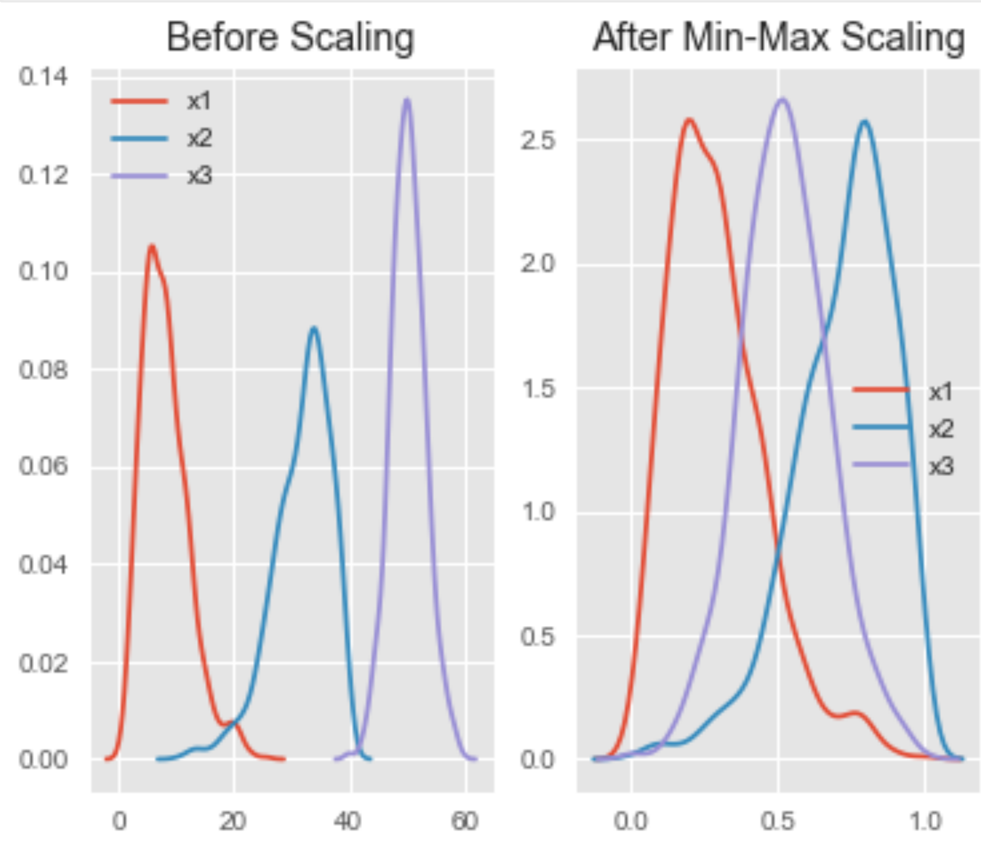
In [3]:

```
df = pd.DataFrame({
    # positive skew
    'x1': np.random.chisquare(8, 1000),
    # negative skew
    'x2': np.random.beta(8, 2, 1000) * 40,
```

```
    # no skew
    'x3': np.random.normal(50, 3, 1000)
})

scaler = preprocessing.MinMaxScaler()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=['x1', 'x2',

fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(6, 5))
ax1.set_title('Before Scaling')
sns.kdeplot(df['x1'], ax=ax1)
sns.kdeplot(df['x2'], ax=ax1)
sns.kdeplot(df['x3'], ax=ax1)
ax2.set_title('After Min-Max Scaling')
sns.kdeplot(scaled_df['x1'], ax=ax2)
sns.kdeplot(scaled_df['x2'], ax=ax2)
sns.kdeplot(scaled_df['x3'], ax=ax2)
plt.show()
```



Notice that the skewness of the distribution is maintained but the
so that they overlap.

Robust Scaler

The `RobustScaler` uses a similar method to the Min-Max scaler bu
than the min-max, so that it is robust to outliers. Therefore it follow

$$\frac{x_i - Q_1(x)}{Q_3(x) - Q_1(x)}$$

For each feature.

Of course this means it is using the less of the data for scaling so it'

the data.

Let's take a look at this one in action on some data with outliers

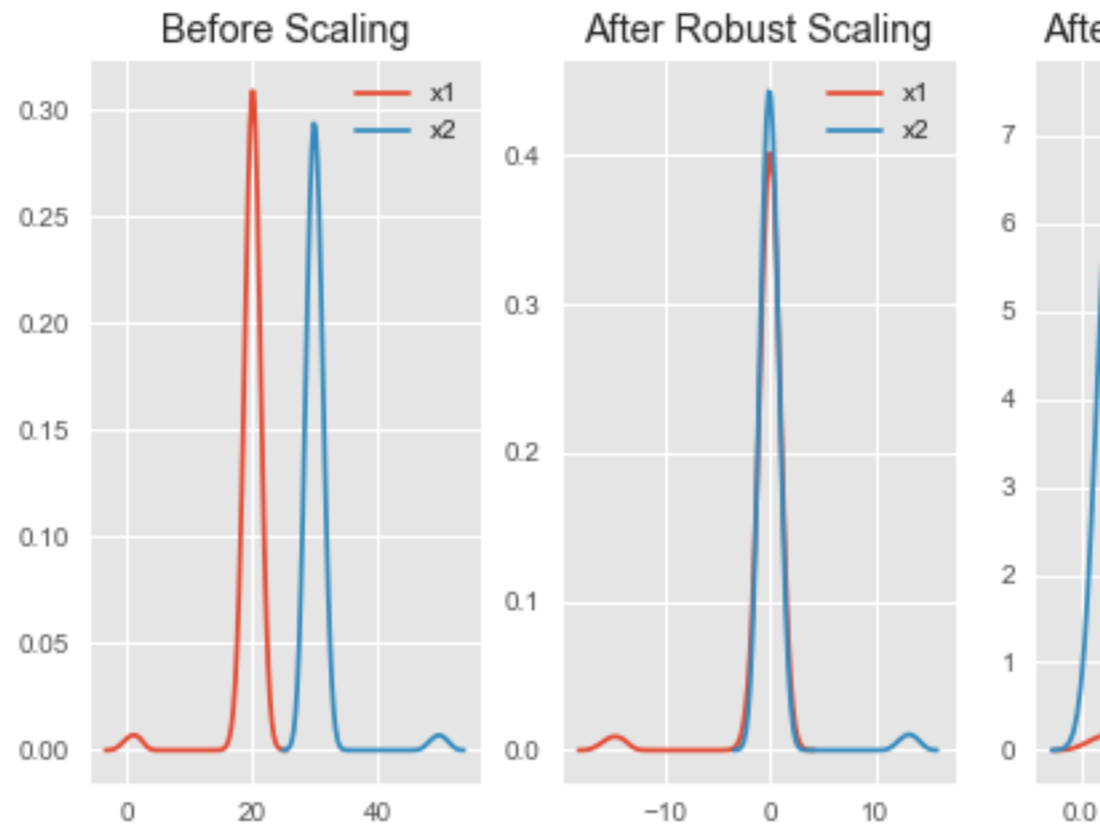
In [4]:

```
x = pd.DataFrame({
    # Distribution with lower outliers
    'x1': np.concatenate([np.random.normal(20, 1, 1000),
    # Distribution with higher outliers
    'x2': np.concatenate([np.random.normal(30, 1, 1000),
})

scaler = preprocessing.RobustScaler()
robust_scaled_df = scaler.fit_transform(x)
robust_scaled_df = pd.DataFrame(robust_scaled_df, column

scaler = preprocessing.MinMaxScaler()
minmax_scaled_df = scaler.fit_transform(x)
minmax_scaled_df = pd.DataFrame(minmax_scaled_df, column

fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(9,
ax1.set_title('Before Scaling')
sns.kdeplot(x['x1'], ax=ax1)
sns.kdeplot(x['x2'], ax=ax1)
ax2.set_title('After Robust Scaling')
sns.kdeplot(robust_scaled_df['x1'], ax=ax2)
sns.kdeplot(robust_scaled_df['x2'], ax=ax2)
ax3.set_title('After Min-Max Scaling')
sns.kdeplot(minmax_scaled_df['x1'], ax=ax3)
sns.kdeplot(minmax_scaled_df['x2'], ax=ax3)
plt.show()
```



Notice that after Robust scaling, the distributions are brought into
remain outside of bulk of the new distributions.

However, in Min-Max scaling, the two normal distributions are kep
1 range.

Normalizer

The normalizer scales each value by dividing each value by its magr
features.

Say your features were x, y and z Cartesian co-ordinates your scale

$$\frac{x_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}}$$

Each point is now within 1 unit of the origin on this Cartesian co-or

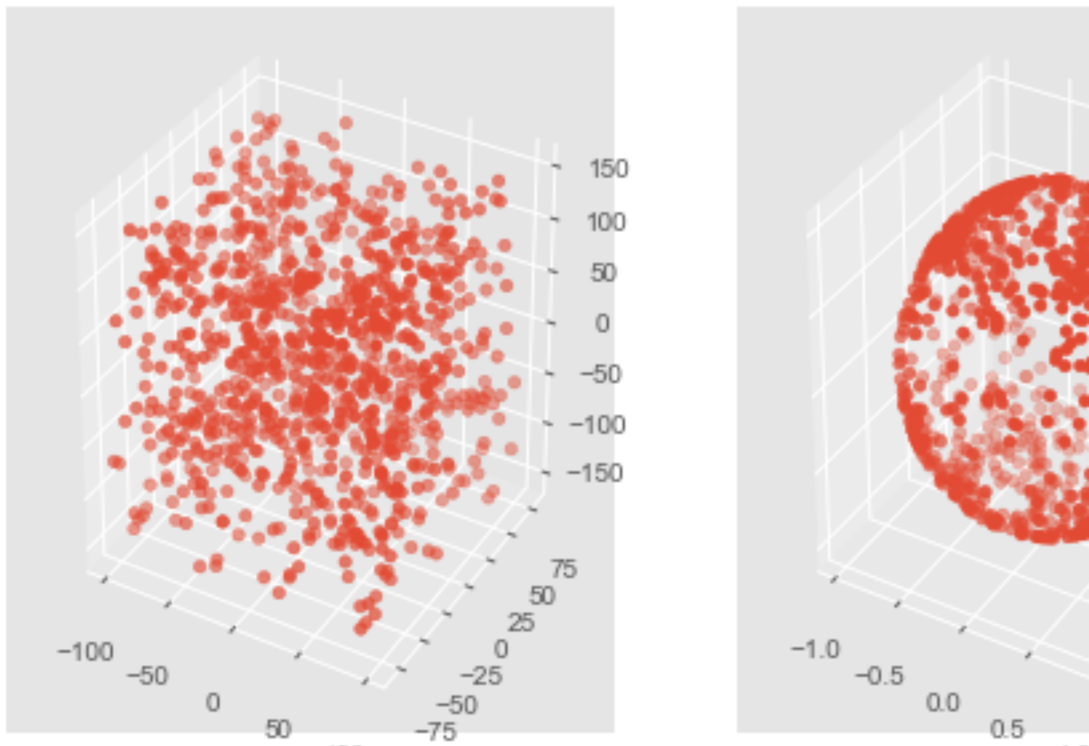
In [5]:

```
from mpl_toolkits.mplot3d import Axes3D

df = pd.DataFrame({
    'x1': np.random.randint(-100, 100, 1000).astype(floa
    'y1': np.random.randint(-80, 80, 1000).astype(float)
    'z1': np.random.randint(-150, 150, 1000).astype(floa
})
```

```
scaler = preprocessing.Normalizer()
scaled_df = scaler.fit_transform(df)
scaled_df = pd.DataFrame(scaled_df, columns=df.columns)

fig = plt.figure(figsize=(9, 5))
ax1 = fig.add_subplot(121, projection='3d')
ax2 = fig.add_subplot(122, projection='3d')
ax1.scatter(df['x1'], df['y1'], df['z1'])
ax2.scatter(scaled_df['x1'], scaled_df['y1'], scaled_df[
plt.show()
```



Note that the points are all brought within a sphere that is at most
axes that were previously different scales are now all one scale.