1 Feature scaling

Many machine learning algorithms work best when numerical data for each of the features (the characteristics such as petal length and sepal length in the iris data set) are on approximately the same scale. The conversion to a similar scale is called data normalisation or data scaling. We perform this as part of out data pre-processing before training an algorithm.

Two common methods of scaling data are 1) to scale all data between 0 and 1 when 0 and 1 are the minimum and maximum values for each feature, and 2) scale the date so that the all features have a mean value of zero and a standard deviation of 1. We will perform this latter normalisation. Scikit learn will perform this for us, but the principle is simple: to normalise in this way, subtract the feature mean from all values, and then divide by the feature standard deviation.

The normalisation parameters are established using the training data set. These parameters are then applied also to the test data set.

Let's first import our packages including StandardScaler from sklearn.preprocessing:

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

To improve readability of the results, we'll set up a function to format NumPy output, controlling the width and decimal places.

```
def set_numpy_decimal_places(places, width=0):
    set_np = '{0:' + str(width) + '.' + str(places) + 'f}'
    np.set_printoptions(formatter={'float': lambda x: set_np.format(x)})

set_numpy_decimal_places(3,6)
```

We'll load up our iris data:

```
# Load iris data

iris = datasets.load_iris()
X=iris.data
y=iris.target

# Create training and test data sets
X_train,X_test,y_train,y_test=train_test_split(
        X,y,test_size=0.3,random_state=0)
```

We'll now apply our scalar, training it on the training data set, and applying it to both the training and test data set.

```
# Initialise a new scaling object
sc=StandardScaler()

# Set up the scaler just on the training set
sc.fit(X_train)

# Apply the scaler to the training and test sets
X_train_std=sc.transform(X_train)
X_test_std=sc.transform(X_test)
```

Let's see what it has done to our data sets. Our training data set appears not to be perfectly normalised, but it is on exactly the same scale as the training data set, which is what our machine learning algorithms will need.

```
print ('Original training set data:')
print ('Mean: ', X_train.mean(axis=0))
print ('StDev:', X_train.std(axis=0))
```

```
print ('\nScaled training set data:')
print ('Mean: ', X_train_std.mean(axis=0))
print ('StDev:', X_train_std.std(axis=0))

print ('\nOriginal test set data:')
print ('Mean: ', X_test.mean(axis=0))
print ('StDev:', X_test.std(axis=0))

print ('\nScaled test set data:')
print ('Mean: ', X_test_std.mean(axis=0))
print ('StDev:', X_test_std.std(axis=0))

OUT:

Original training set data:
Mean:  [ 5.893  3.045  3.829  1.227]
StDev: [ 0.873  0.439  1.796  0.778]

Scaled training set data:
Mean:  [ 0.000 -0.000 -0.000 -0.000]
StDev: [ 1.000  1.000  1.000  1.000]

Original test set data:
Mean:  [ 5.727  3.076  3.596  1.133]
StDev: [ 0.688  0.414  1.656  0.715]

Scaled test set data:
Mean:  [-0.191  0.070 -0.130 -0.120]
StDev: [ 0.789  0.943  0.922  0.919]
```