Rescaling a Feature

MinMaxScaler to rescale a feature array, typically between 0 to 1 or –1 to 1. It uses the minimum and maximum values of a feature to rescale values to within a range

x i ' = x i -min(x)/max(x)-min(x)

where x is the feature vector, xi is an individual element of feature x, and x’i is the rescaled element.

```
# Load libraries
import numpy as np
from sklearn import preprocessing

# Create feature
feature = np.array([[-500.5],
                    [-100.1],
                    [0],
                    [100.1],
                    [900.9]])

# Create scaler
minmax_scale = preprocessing.MinMaxScaler(feature_range=(0, 1))

# Scale feature
scaled_feature = minmax_scale.fit_transform(feature)

# Show feature
scaled_feature
```

```
     array([[0.        ],
            [0.28571429],
            [0.35714286],
            [0.42857143],
            [1.        ]])
```

outputted array has been successfully rescaled to between 0 and 1

One option is to use fit to calculate the minimum and maximum values of the feature, then use transform to rescale the feature. The second option is to use fit_transform to do both operations at once

Standardizing a Feature

Standardization is another scaling technique where the values are centered around the mean with a unit standard deviation.We want to transform a feature to have a mean of 0 and a standard deviation of 1.

$$X' = \frac{X - \mu}{\sigma}$$

```
# Load libraries
import numpy as np
from sklearn import preprocessing

# Create feature
x = np.array([[-1000.1],
              [-200.2],
              [500.5],
              [600.6],
              [9000.9]])

# Create scaler
scaler = preprocessing.StandardScaler()

# Transform the feature
standardized = scaler.fit_transform(x)

# Show feature
standardized
```

```
array([[-0.76058269],
       [-0.54177196],
       [-0.35009716],
       [-0.32271504],
       [ 1.97516685]])
```

```
# Print mean and standard deviation
print("Mean:", round(standardized.mean()))
print("Standard deviation:", standardized.std())
```

```
Mean: 0
Standard deviation: 1.0
```

Encoding Categorical Data

Ordinal Encoding

In ordinal encoding, each unique category value is assigned an integer value.

For example, “red” is 1, “green” is 2, and “blue” is 3.

This is called an ordinal encoding or an integer encoding

```
# example of a ordinal encoding
from numpy import asarray
from sklearn.preprocessing import OrdinalEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define ordinal encoding
encoder = OrdinalEncoder()
# transform data
result = encoder.fit_transform(data)
print(result)
```

```
[['red']
 ['green']
 ['blue']]
[[2.]
 [1.]
 [0.]]
```

asarray() function is used when we want to convert input to an array

One-Hot Encoding

For categorical variables where no ordinal relationship exists

```
# example of a one hot encoding
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define one hot encoding
encoder = OneHotEncoder(sparse=False)
# transform data
onehot = encoder.fit_transform(data)
print(onehot)
```

```
[['red']
 ['green']
 ['blue']]
[[0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
```

0s completed at 10:50 AM