Data Range

Create a function to compress data into a specific range of values.

Chapter Goals:

- Learn how to compress data values to a specified range

A. Range scaling

Apart from standardizing data, we can also scale data by compressing it into a fixed range. One of the biggest use cases for this is compressing data into the range [0, 1]. This allows us to view the data in terms of proportions, or percentages, based on the minimum and maximum values in the data.

The formula for scaling based on a range is a two-step process. For a given data value, x, we first compute the proportion of the value with respect to the min and max of the data d_{min} and d_{max}, respectively).

$$x_{prop} = \frac{x - d_{min}}{d_{max} - d_{min}}$$

The formula above computes the proportion of the data value, x_{prop}. Note that this only works if not all the data values are the same (i.e. $d_{\text{max}} \neq d_{\text{min}}$).

We then use the proportion of the value to scale to the specified range, [r_{min}, r_{max}]. The formula below calculates the new scaled value, x_{scale}.

$$x_{scale} = x_{prop} \cdot (r_{max} - r_{min}) + r_{min}$$

B. Range compression in scikit-learn

The scikit-learn library provides a variety of *transformers*, modules that perform transformations on data. While in the previous chapter we used a single function, `scale`, to perform the data standardization, the remaining chapters will focus on using these transformer modules.

The `MinMaxScaler` transformer performs the range compression using the previous formula. Specifically, it scales each feature (column) of the data to a given range (where the default range is [0, 1]).

The code below shows how to use the `MinMaxScaler` (with the default range and a custom range).

The `MinMaxScaler` object contains a function called `fit_transform`, which allows it to take in the input data array and then output the scaled data. The function is a combination of the object's `fit` and `transform` functions, where the former takes in an input data array and the latter transforms a (possibly different) array based on the data from the input to the `fit` function.

```
# predefined data
print('{}\n'.format(repr(data)))

from sklearn.preprocessing import MinMaxScaler
default_scaler = MinMaxScaler() # the default range is [0,1]
transformed = default_scaler.fit_transform(data)
print('{}\n'.format(repr(transformed)))

custom_scaler = MinMaxScaler(feature_range=(-2, 3))
transformed = custom_scaler.fit_transform(data)
print('{}\n'.format(repr(transformed)))
```

Now lets run the `fit` and `transform` functions separately and compare them with the `fit_transform` function. `fit` takes in an input data array and `transform` transforms a (possibly different) array based on the data from the input to the fit function.

```
# predefined new_data
print('{}\n'.format(repr(new_data)))

from sklearn.preprocessing import MinMaxScaler
default_scaler = MinMaxScaler() # the default range is [0,1]
transformed = default_scaler.fit_transform(new_data)
print('{}\n'.format(repr(transformed)))

default_scaler = MinMaxScaler()  # new instance
default_scaler.fit(data)  # different data value fit
transformed = default_scaler.transform(new_data)
print('{}\n'.format(repr(transformed)))
```

The code above scales the `new_data` array to the range [0, 1], based on the (column-wise) minimum and maximum values from the `data` array in the original code example.

Time to Code!

The coding exercise in this chapter uses `MinMaxScaler` (imported in backend) to complete the `ranged_data` function.

The function will compress the input NumPy array, `data`, into the range given by `value_range`.

Set `min_max_scaler` equal to `MinMaxScaler` initialized with `value_range` for the `feature_range` keyword argument.

Set `scaled_data` equal to `min_max_scaler.fit_transform` applied with `data` as the only argument. Then return `scaled_data`.

```
def ranged_data(data, value_range):
  # CODE HERE
  pass
```