SciKit Learn Preprocessing Overview

In [1]:
```
import numpy as np
from sklearn.preprocessing import MinMaxScaler
```

Complete reproducability, if it can be done in under 10 minutes

In [2]:
```
#print("Only here for reproducibility")
#np.random.seed(101)
```

In [3]:
```
print("Only here for reproducibility")
np.random.seed(101)
#np.random.randint(1, 1000, (1, 10))
np.random.randint(0, 1000, (11, 10))
data = np.random.randint(0, 100, (10, 2))
print("Only here for reproducibility (specifically or the random integer array)")
```

```
Only here for reproducibility
Only here for reproducibility (specifically or the random integer array)
```

In [4]:
```
print(str(data) + "\n\n" + str(type(data)))
```

```
[[92 11]
 [10 94]
 [35 28]
 [ 3 83]
 [84 47]
 [14 69]
 [60 69]
 [51  6]
 [88 71]
 [68 23]]

<class 'numpy.ndarray'>
```

In [5]:
```
data
```

Out[5]:
```
array([[92, 11],
       [10, 94],
       [35, 28],
       [ 3, 83],
       [84, 47],
       [14, 69],
       [60, 69],
       [51,  6],
       [88, 71],
       [68, 23]])
```

< SKIP >

No more time on this part. `-- v --`

In [6]:
```
# #  First, scaling between 0 and 1 based on the:
# #+ min (3) ; and the max (94). My guess (DWB, 2023-11-13)
# #+ is that it's fine tuning on something like
# #+   output(in) = (in - min) / (max - min) = (in - 3) / (94 - 3)
# #+ There are problems with the 92 -> 1. and the 6 -> 0., which
# #+ is where the fine tuning comes in
# scaler_model = MinMaxScaler()
```

Oh, here we go from the docs.

> The transformation is given by::
> ` `
` X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0)) `
` X_scaled = X_std * (max - min) + min `
` `
` where min, max = feature_range. `
` `
` This transformation is often used as an alternative to zero mean, `
` unit variance scaling. `

In [7]:
```
# type(scaler_model)
```

In [8]:
```
# def scikitlearn_transform(in_val, min_val=3, max_val=94):
#     '''
#     Simple (not vectorized) test of the normalization transformation
#     performed by sklearn.preprocessing.data.MinMaxScaler.fit
#     When I say not vectorized, I mean it just takes one input number
#     to be transformed, along with the max and min, and only gives
#     one output. It is specialized for the data from the lecture.

#     DWB, 2023-11-13
#     '''

#     do_debug = False

#     x_in_val = float(in_val)

#     x_std_skl = standard_transform(x_in_val, min_val, max_val)

#     if do_debug:
#         print("x_std_skl:" + str(x_std_skl))
#     ##endof:  if do_debug

#     theory_data_min = 0
#     theory_data_max = 100

#     theory_feat_max = 1.0
#     theory_feat_min = 0.0


#     # #This one would just give you back what you put in
#     # x_scaled_skl = x_std_skl * (max_val - min_val) + min_val
#     #                 # exactly the same output as input

#     # # I'm pretty sure these two are wrong, too, but let's investigate
#     # x_scaled_skl = \
#     #     x_std_skl * (theory_feat_max - theory_feat_min) + theory_feat_min
#     #     # exactly the same output as input
#     # x_scaled_skl = \
#     #     x_std_skl * (theory_data_max - theory_data_min) + theory_data_min
#     #     # weird messed up

#     min_val_to_use, max_val_to_use = \
#         data_min_max_4_normalize(
```

```
#                       x_in_val,
#                       min_val, max_val,
#                       theory_data_min, theory_data_max,
#                       theory_feat_min, theory_feat_max)

#     if do_debug:
#         print("min_val_to_use: " + str(min_val_to_use))
#         print("max_val_to_use: " + str(max_val_to_use))
#     ##endof:  if do_debug

#     x_scaled_skl = \
#         x_std_skl * (max_val_to_use - min_val_to_use) + min_val_to_use

#     if do_debug:
#         print("x_scaled_skl:" + str(x_scaled_skl))
#     ##endof:  if do_debug

#     return x_scaled_skl

# ##endof:  scikitlearn_transform(in_val, max_val = 94, min_val = 3)

# def standard_transform(in_val_std, min_value, max_value):
#     '''
#     The standard way of normalizing
#     I think this is the "zero mean, unit variance scaling"
#     '''

#     in_val = in_val_std
#     min_val = min_value
#     max_val = max_value

#     return float( (in_val - min_val) / (max_val - min_val) )

# ##endof:  standard_transform(in_val, max_val, min_val)

# def data_min_max_4_normalize(
#                       in_val_data,
#                       min_val_data=3., max_val_data=94.,
#                       min_theoretical_data=0.,
#                       max_theoretical_data=100.,
#                       min_theoretical_normed_feature=0.,
#                       max_theoretical_normed_feature=1.):
#     '''
```

```
#     '''

#     # y = m*x + b
#     conv_m_for_data2normed = ((max_theoretical_normed_feature - min_theoretical_normed_feature) / ( max_the
oretical_data - min_theoretical_data ))
#      # rise / run

#     # b = y_given - m*x_given, (0, 0) is trivial, but right, let's do (100, 1)
#     conv_b_for_data2normed = \
#       max_theoretical_normed_feature - (conv_m_for_data2normed * max_theoretical_data)

#     min_val_ret = conv_m_for_data2normed * min_val_data + conv_b_for_data2normed

#     max_val_ret = conv_m_for_data2normed * max_val_data + conv_b_for_data2normed

#     return min_val_ret, max_val_ret

# ##endof:  data_min_max_scoring()
```

In [9]:

```
# # Remember the data
# data
```

In [10]:
```
# t00 = scikitlearn_transform(92)
# print(t00)
# t01 = scikitlearn_transform(11)
# print(t01)
# t10 = scikitlearn_transform(10)
# print(t10)
# t11 = scikitlearn_transform(94)
# print(t11)
# t20 = scikitlearn_transform(35)
# print(t20)
# t21 = scikitlearn_transform(28)
# print(t21)
# t30 = scikitlearn_transform(3)
# print(t30)

# lets_see = [[t00, t01],[t10, t11],[t20, t21], [t30, "..."]]

# import pprint

# pprint.pprint(lets_see)
```

End of the part for which there's no more time. `-- ^ --`

< / SKIP >

In [11]:
```
#  First, scaling between 0 and 1 based on the:
#+ min (3) ; and the max (94). This will include
#+ three lines of code:
#
# % scaler_model = MinMaxScaler()
# % scaler_model.fit(data)
# % scaler_model.transform(data)

scaler_model = MinMaxScaler()
```

In [12]:
```
type(scaler_model)
```

Out[12]:
```
sklearn.preprocessing.data.MinMaxScaler
```

In [13]:
```
scaler_model.fit(data) # A warning will come up, because it converts ints to floats
```

```
C:\Users\Anast\.conda\envs\tfdeeplearning\lib\site-packages\sklearn\utils\validation.py:444: DataConversionWa
rning: Data with input dtype int32 was converted to float64 by MinMaxScaler.
  warnings.warn(msg, DataConversionWarning)
```

Out[13]:
```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

In [14]:
```
scaler_model.transform(data)
```

Out[14]:
```
array([[1.        , 0.05681818],
       [0.07865169, 1.        ],
       [0.35955056, 0.25      ],
       [0.        , 0.875     ],
       [0.91011236, 0.46590909],
       [0.12359551, 0.71590909],
       [0.64044944, 0.71590909],
       [0.53932584, 0.        ],
       [0.95505618, 0.73863636],
       [0.73033708, 0.19318182]])
```

In [15]:
```
normalized_data = scaler_model.transform(data)

print(str(normalized_data) + "\n\n" + str(type(normalized_data)))
```

```
[[1.         0.05681818]
 [0.07865169 1.        ]
 [0.35955056 0.25      ]
 [0.         0.875     ]
 [0.91011236 0.46590909]
 [0.12359551 0.71590909]
 [0.64044944 0.71590909]
 [0.53932584 0.        ]
 [0.95505618 0.73863636]
 [0.73033708 0.19318182]]

<class 'numpy.ndarray'>
```

In [16]:
```
normalized_data
```

Out[16]:
```
array([[1.        , 0.05681818],
       [0.07865169, 1.        ],
       [0.35955056, 0.25      ],
       [0.        , 0.875     ],
       [0.91011236, 0.46590909],
       [0.12359551, 0.71590909],
       [0.64044944, 0.71590909],
       [0.53932584, 0.        ],
       [0.95505618, 0.73863636],
       [0.73033708, 0.19318182]])
```

Usually, you fit to your training data and use the resulting fit to transform both training and test data. (No fitting on the test data!) However, for possible learning exercises or quick tests, there is the following function that both fits and transforms the data.

In [17]:
```
#  Not usually good practice.
#+ Still, so you can see it gives the same thing.
one_step_result = scaler_model.fit_transform(data)

print(str(one_step_result) + "\n\n" + str(type(one_step_result)))
```

```
[[1.         0.05681818]
 [0.07865169 1.        ]
 [0.35955056 0.25      ]
 [0.         0.875     ]
 [0.91011236 0.46590909]
 [0.12359551 0.71590909]
 [0.64044944 0.71590909]
 [0.53932584 0.        ]
 [0.95505618 0.73863636]
 [0.73033708 0.19318182]]

<class 'numpy.ndarray'>

C:\Users\Anast\.conda\envs\tfdeeplearning\lib\site-packages\sklearn\utils\validation.py:444: DataConversionWa
rning: Data with input dtype int32 was converted to float64 by MinMaxScaler.
  warnings.warn(msg, DataConversionWarning)
```

In [18]:
```
one_step_result
```

Out[18]:
```
array([[1.        , 0.05681818],
       [0.07865169, 1.        ],
       [0.35955056, 0.25      ],
       [0.        , 0.875     ],
       [0.91011236, 0.46590909],
       [0.12359551, 0.71590909],
       [0.64044944, 0.71590909],
       [0.53932584, 0.        ],
       [0.95505618, 0.73863636],
       [0.73033708, 0.19318182]])
```

In [19]:
```
# That can be compared to the original.
data
```

Out[19]:
```
array([[92, 11],
       [10, 94],
       [35, 28],
       [ 3, 83],
       [84, 47],
       [14, 69],
       [60, 69],
       [51,  6],
       [88, 71],
       [68, 23]])
```

In [20]:
```
print(str(data) + "\n\n" + str(type(data)))
```

```
[[92 11]
 [10 94]
 [35 28]
 [ 3 83]
 [84 47]
 [14 69]
 [60 69]
 [51  6]
 [88 71]
 [68 23]]

<class 'numpy.ndarray'>
```

And now, some Pandas stuff!

We'll do the train/test split, here.

In [21]:
```
import pandas as pd
```

In [22]:
```
mydata = np.random.randint(0, 101, (50, 4))
```

In [23]:
```
mydata
```

Out[23]:
```
array([[ 35,  79,  98,  67],
       [ 82,  57,  77,  46],
       [  3,  46,  29,  86],
       [ 21,  21,  81,  23],
       [ 94, 100,  71,  20],
       [ 27,  75,   5,  49],
       [ 86,  89,  63,  82],
       [ 77,   3,  56,  14],
       [ 49,  87,  52,  13],
       [ 47,  49,  24,  20],
       [ 64,  52,  60,  47],
       [ 29,  60,  53,  11],
       [ 40,  91,  45,  97],
       [ 24,  36,  38,   9],
       [ 52,  67,  43,   1],
       [ 79,  68,  68, 100],
       [ 61,  18,  51,  14],
       [ 28,  17,  87,  46],
       [ 52,  16,  70,  71],
       [ 84,  10,  62,  96],
       [ 57,  23,  86,  85],
       [ 26,  76,  66,  54],
       [ 17,  65,  57,  89],
       [  2,  80,  50,  66],
       [ 88,  79,  93,   6],
       [ 92,  42,  22,  20],
       [ 25,  97,  54,  71],
       [ 72,  80,  93,  64],
       [ 63,  80,  38,  45],
       [ 35,  25,  95,  75],
       [ 72,  11,  76,  79],
       [ 50,  22,  59,  66],
       [  1,  34,  37,  57],
       [ 35,  42,  44,  49],
       [ 31,  79,  85,   3],
       [ 55,  73,  93,  94],
       [ 99,  40,  54,  88],
       [ 94,  86,  17,  68],
       [ 17,  18,  60,  83],
       [ 82,   7,  67,  34],
       [ 76,  94,  20,  69],
       [ 73,  59,  34,  69],
       [ 25,  78,  92,  74],
```

```
       [ 75,  33,   9,  43],
       [ 20,  82,  30,   3],
       [ 46,  29,  47,  27],
       [ 81,  71,  25,  94],
       [ 57,  21,  29,   6],
       [ 54,  47,  47,  60],
       [  6,  75,  97,  53]])
```

In [24]:

```
df = pd.DataFrame(data=mydata)
```

In [25]:
```
df
```

Out[25]:

	0	1	2	3
0	35	79	98	67
1	82	57	77	46
2	3	46	29	86
3	21	21	81	23
4	94	100	71	20
5	27	75	5	49
6	86	89	63	82
7	77	3	56	14
8	49	87	52	13
9	47	49	24	20
10	64	52	60	47
11	29	60	53	11
12	40	91	45	97
13	24	36	38	9
14	52	67	43	1
15	79	68	68	100
16	61	18	51	14
17	28	17	87	46
18	52	16	70	71
19	84	10	62	96
20	57	23	86	85
21	26	76	66	54
22	17	65	57	89
23	2	80	50	66
24	88	79	93	6
25	92	42	22	20

	0	1	2	3
26	25	97	54	71
27	72	80	93	64
28	63	80	38	45
29	35	25	95	75
30	72	11	76	79
31	50	22	59	66
32	1	34	37	57
33	35	42	44	49
34	31	79	85	3
35	55	73	93	94
36	99	40	54	88
37	94	86	17	68
38	17	18	60	83
39	82	7	67	34
40	76	94	20	69
41	73	59	34	69
42	25	78	92	74
43	75	33	9	43
44	20	82	30	3
45	46	29	47	27
46	81	71	25	94
47	57	21	29	6
48	54	47	47	60
49	6	75	97	53

Let's name the columns.

In [26]:
```
df2 = pd.DataFrame(data=mydata, columns=['f1', 'f2', 'f3', 'label'])
```

In [27]:
```
df2
```

Out[27]:

	f1	f2	f3	label
0	35	79	98	67
1	82	57	77	46
2	3	46	29	86
3	21	21	81	23
4	94	100	71	20
5	27	75	5	49
6	86	89	63	82
7	77	3	56	14
8	49	87	52	13
9	47	49	24	20
10	64	52	60	47
11	29	60	53	11
12	40	91	45	97
13	24	36	38	9
14	52	67	43	1
15	79	68	68	100
16	61	18	51	14
17	28	17	87	46
18	52	16	70	71
19	84	10	62	96
20	57	23	86	85
21	26	76	66	54
22	17	65	57	89
23	2	80	50	66
24	88	79	93	6
25	92	42	22	20

	f1	f2	f3	label
26	25	97	54	71
27	72	80	93	64
28	63	80	38	45
29	35	25	95	75
30	72	11	76	79
31	50	22	59	66
32	1	34	37	57
33	35	42	44	49
34	31	79	85	3
35	55	73	93	94
36	99	40	54	88
37	94	86	17	68
38	17	18	60	83
39	82	7	67	34
40	76	94	20	69
41	73	59	34	69
42	25	78	92	74
43	75	33	9	43
44	20	82	30	3
45	46	29	47	27
46	81	71	25	94
47	57	21	29	6
48	54	47	47	60
49	6	75	97	53

This is the data on which we'll do the train/test split.

In [28]:
```
X = data[['f1', 'f2', 'f3']] # This is wrong, and it will throw an error.
```

```
C:\Users\Anast\.conda\envs\tfdeeplearning\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: Using a n
on-tuple sequence for multidimensional indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. I
n the future this will be interpreted as an array index, `arr[np.array(seq)]`, which will result either in an
error or a different result.
  """Entry point for launching an IPython kernel.

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-28-183d3de9e706> in <module>()
----> 1 X = data[['f1', 'f2', 'f3']] # This is wrong, and it will throw an error.

IndexError: only integers, slices (`:`), ellipsis (`...`), numpy.newaxis (`None`) and integer or boolean arra
ys are valid indices
```

In [29]:
```
# Let's do it right, with the Pandas DataFrame
```

In [30]:
```
X = df2[['f1', 'f2', 'f3']]
```

In [31]:
```
X
```

Out[31]:

	f1	f2	f3
0	35	79	98
1	82	57	77
2	3	46	29
3	21	21	81
4	94	100	71
5	27	75	5
6	86	89	63
7	77	3	56
8	49	87	52
9	47	49	24
10	64	52	60
11	29	60	53
12	40	91	45
13	24	36	38
14	52	67	43
15	79	68	68
16	61	18	51
17	28	17	87
18	52	16	70
19	84	10	62
20	57	23	86
21	26	76	66
22	17	65	57
23	2	80	50
24	88	79	93
25	92	42	22

	f1	f2	f3
26	25	97	54
27	72	80	93
28	63	80	38
29	35	25	95
30	72	11	76
31	50	22	59
32	1	34	37
33	35	42	44
34	31	79	85
35	55	73	93
36	99	40	54
37	94	86	17
38	17	18	60
39	82	7	67
40	76	94	20
41	73	59	34
42	25	78	92
43	75	33	9
44	20	82	30
45	46	29	47
46	81	71	25
47	57	21	29
48	54	47	47
49	6	75	97

In [32]:
```
y = df2['label']
```

In [33]:
```
y
```

Out[33]:
```
0      67
1      46
2      86
3      23
4      20
5      49
6      82
7      14
8      13
9      20
10     47
11     11
12     97
13      9
14      1
15    100
16     14
17     46
18     71
19     96
20     85
21     54
22     89
23     66
24      6
25     20
26     71
27     64
28     45
29     75
30     79
31     66
32     57
33     49
34      3
35     94
36     88
37     68
38     83
39     34
40     69
41     69
42     74
```

```
43     43
44      3
45     27
46     94
47      6
48     60
49     53
Name: label, dtype: int32
```

In [34]:
```
from sklearn.model_selection import train_test_split
```

Here, in a Jupyter notebook, it's very easy to simply write

```
train_test_split
```

into the next cell, then do the `Shift` + `Tab` a couple times until we find the following text to copy/paste

```
>>> X_train, X_test, y_train, y_test = train_test_split(
...     X, y, test_size=0.33, random_state=42)
...
```

We can then put it all on one line, so we don't get an error with the ellipses, and change the parameters as we'd like. Let's match Jose's lecture.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=101)
```

In [35]:
```
train_test_split # Put the cursor after the 'train_test_split', then get the docs
```

Out[35]:
```
<function sklearn.model_selection._split.train_test_split(*arrays, **options)>
```

In [36]:
```
X_train, X_test, y_train, y_test = \
    train_test_split(X, y, test_size=0.3, random_state=101)
```

In [37]:
```
X_train
```

Out[37]:

	f1	f2	f3
3	21	21	81
41	73	59	34
30	72	11	76
15	79	68	68
20	57	23	86
43	75	33	9
38	17	18	60
44	20	82	30
39	82	7	67
10	64	52	60
49	6	75	97
25	92	42	22
33	35	42	44
36	99	40	54
2	3	46	29
27	72	80	93
34	31	79	85
35	55	73	93
8	49	87	52
19	84	10	62
29	35	25	95
12	40	91	45
5	27	75	5
0	35	79	98
28	63	80	38
4	94	100	71

	f1	f2	f3
40	76	94	20
13	24	36	38
9	47	49	24
48	54	47	47
23	2	80	50
6	86	89	63
17	28	17	87
11	29	60	53
31	50	22	59

In [38]:
```
X_test
```

Out[38]:

	f1	f2	f3
37	94	86	17
14	52	67	43
21	26	76	66
32	1	34	37
22	17	65	57
1	82	57	77
26	25	97	54
46	81	71	25
42	25	78	92
47	57	21	29
16	61	18	51
24	88	79	93
7	77	3	56
45	46	29	47
18	52	16	70

In [39]:
```
y_train
```

Out[39]:
```
3      23
41     69
30     79
15    100
20     85
43     43
38     83
44      3
39     34
10     47
49     53
25     20
33     49
36     88
2      86
27     64
34      3
35     94
8      13
19     96
29     75
12     97
5      49
0      67
28     45
4      20
40     69
13      9
9      20
48     60
23     66
6      82
17     46
11     11
31     66
Name: label, dtype: int32
```

In [40]:
```
y_test
```

Out[40]:
```
37    68
14     1
21    54
32    57
22    89
1     46
26    71
46    94
42    74
47     6
16    14
24     6
7     14
45    27
18    71
Name: label, dtype: int32
```

That's not all for now, yet.

I'm going the follow the course materials, though I'm not going to go through the trouble of making things repeatable. You'll see my efforts to get it there, but that was enough. (My therapist would be so proud!)

In [41]:
```
print("Only here for reproducibility")
np.random.seed(101)
#np.random.randint(1, 1000, (1, 10))
np.random.randint(0, 1000, (11, 10))
data = np.random.randint(0, 100, (10, 2))
print("Only here for reproducibility (specifically or the random integer array)")
```

```
Only here for reproducibility
Only here for reproducibility (specifically or the random integer array)
```

In [42]:
```
import pandas as pd
```

In [43]:
```
data = pd.DataFrame(data=np.random.randint(0, 101, (50, 4)),
                    columns=['f1', 'f2', 'f3', 'label'])
```

In [44]:
```
data.head()
```

Out[44]:

	f1	f2	f3	label
0	35	79	98	67
1	82	57	77	46
2	3	46	29	86
3	21	21	81	23
4	94	100	71	20

In [45]:
```
x = data[['f1', 'f2', 'f3']] # Alternatively: x = data.drop('label', axis=1)
y = data['label']
```

In [46]:
```
from sklearn.model_selection import train_test_split
```

In [47]:
```
X_train, X_test, y_train, y_test = \
                        train_test_split(x, y,
                                         test_size=0.3,
                                         random_state=101)
```

In [48]:
```
X_train.shape
```

Out[48]: (35, 3)

In [49]:
```
X_test.shape
```

Out[49]: (15, 3)

In [50]:
```
y_train.shape
```

Out[50]: (35,)

In [51]:
```
y_test.shape
```

Out[51]: (15,)

That's all for now, folks!