Scaling

Sometimes it is necessary to restrict the *scale* of our data

- Features and targets

There are several reasons

- Some models are sensitive to scale
    - Features with large differences in scale may affect the model
- Some Loss functions are sensitive to scale

Some scalers

Scaling transformations are relatively simple mathematically.

Let's discuss the *how* of scaling before delving into the *why*.

Note Scaling may be applied to a variable regardless of whether it is a feature of target.

MinMax

Convert to $[0, 1]$ range.

$$\tilde{\mathbf{x}}_j^{\mathbf{(i)}} = \frac{\mathbf{x}_j^{\mathbf{(i)}} - \min\limits_{1 \le i \le m} (\mathbf{x}_j^{\mathbf{(i)}})}{\min\limits_{1 \le i \le m} (\mathbf{x}_j^{\mathbf{(i)}}) - \min\limits_{1 \le i \le m} (\mathbf{x}_j^{\mathbf{(i)}}}$$

New feature is measured in units of "fraction of range".

Standardize

$$\tilde{\mathbf{x}}_j^{(\mathbf{i})} = \frac{\mathbf{x}_j^{(\mathbf{i})} - \bar{\mathbf{x}}_j}{\sigma_{\mathbf{x}_j}}$$

The new feature has a mean of 0, i.e., is *zero centered*.

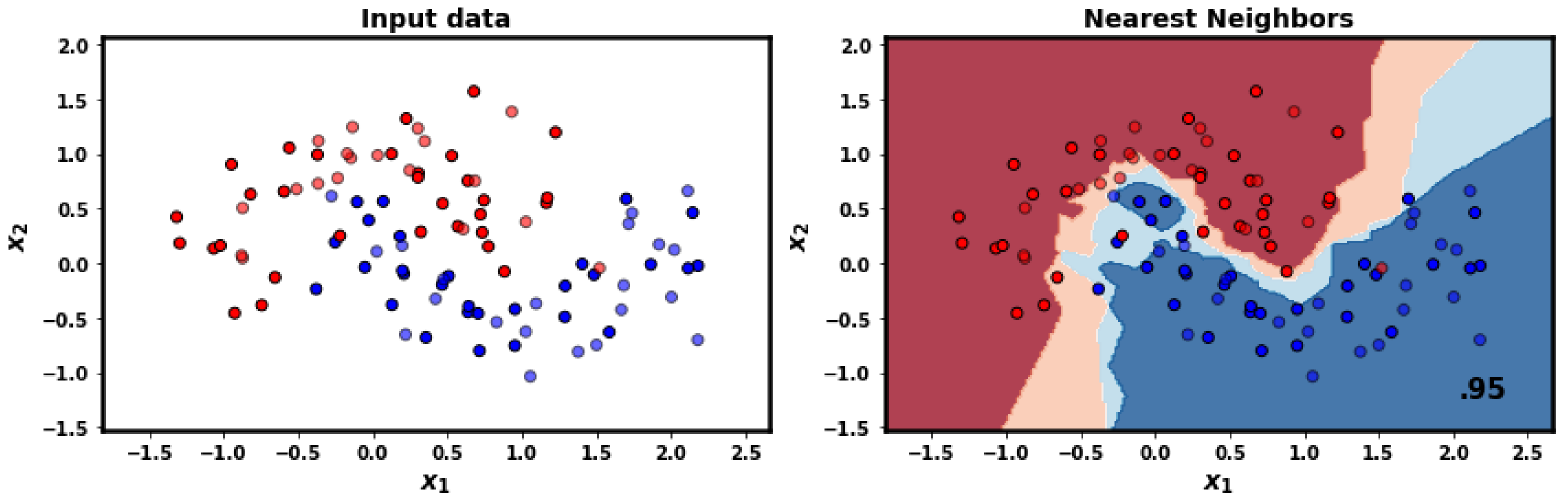
New feature is measured in units of "number of standard deviations".

Loss functions sensitive to scale

One reason for scaling features is the mathematics of the model.

Consider the behavior of the KNN Classification model on a set of examples in which the two features $\mathbf{x}_1, \mathbf{x}_2$ are roughly the same scale.

```
In [4]:  _=  kn.plot_classifiers(scale=False)
```



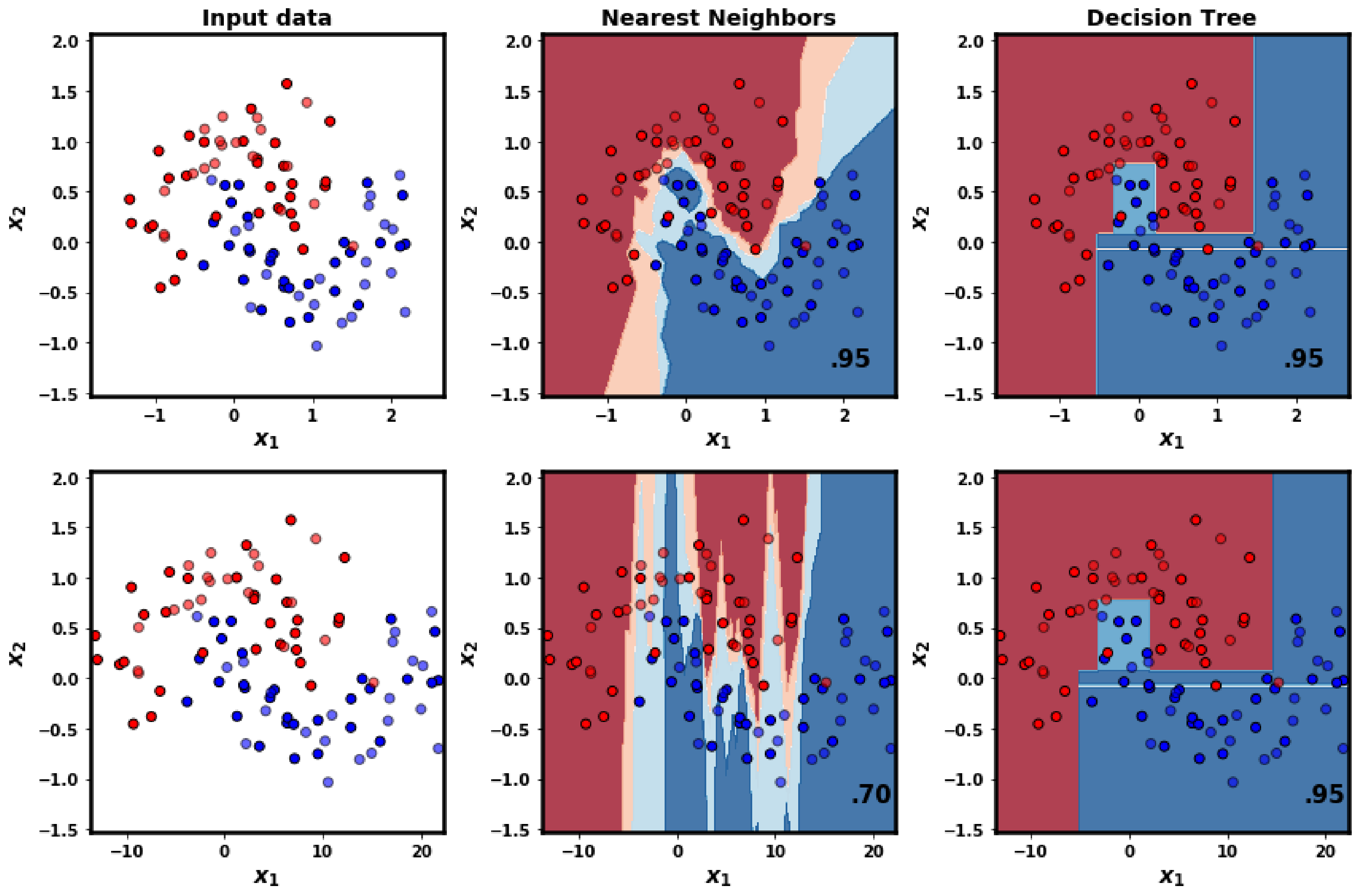
What would happen if the scales were very different for the two features ?

Let's "stretch" the first feature by multiplying it by a factor of 10, leaving the second feature unchanged.

(n.b., accuracy is the number in the lower right of the prediction plot)

In [5]:
```python
kn = tmh.KNN_Helper()

_ = kn.plot_classifiers(scale=False, num_ds=2)
```



The first row shows the original dataset; the second the "stretched" data set (i.e., scale of $\mathbf{x}_1$)

Note the horizontal scale of the lower plot is 10 times the upper.

- The plots look the same because the plotting routine keeps the plot sizes the same.

As you can see, the KNN classifier produces drastically different results on the two datasets.

Arguably KNN's predictions on the stretched dataset feel overfit.

- Even with overfitting: accuracy drops from 95% to 70%

What is the reason for this ?

If you recall our brief introduction to KNN:

- The classifier measures the distance between features in a test example and feature in (each) training example
- The distance measure (L2, sum of squared feature-wise differences) is sensitive to scale

$(\mathbf{x}_1^{(\mathbf{i})}, \mathbf{x}_2^{(\mathbf{i})})$ training example i

$(10 * \mathbf{x}_1^{(\mathbf{i})}, \mathbf{x}_2^{(\mathbf{i})})$ stretched training example i

$(\mathbf{x}_1^{(test)}, \mathbf{x}_2^{(test)})$ test example

$(\mathbf{x}_1^{(test)} - \mathbf{x}_1^{(\mathbf{i})})^2 \quad +(\mathbf{x}_2^{(test)} - \mathbf{x}_2^{(\mathbf{i})})^2$ distance from test

$(\mathbf{x}_1^{(test)} - 10 * \mathbf{x}_1^{(\mathbf{i})})^2 \quad +(\mathbf{x}_2^{(test)} - \mathbf{x}_2^{(\mathbf{i})})^2$ stretched distance from test

Differences in $\mathbf{x}_1$ are much more important than differences in $\mathbf{x}_2$ in the stretched data.

The issue is that the distance metric used by KNN is *sensitive to feature scale.*

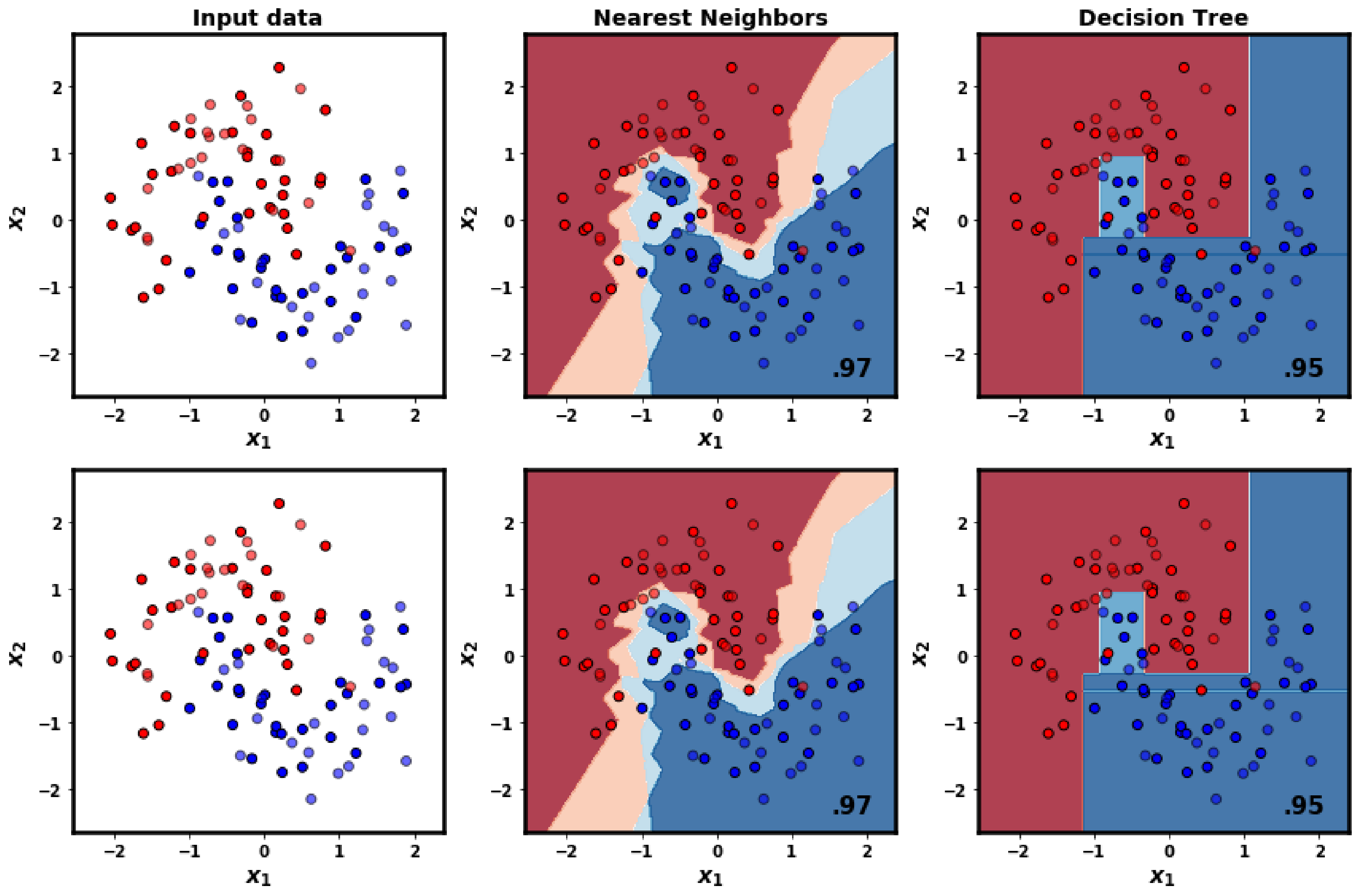
By rescaling both features to be on a similar scale (as in the raw input)

- We balance the contribution to distance from each feature
- Removing the sensitivity

Let's see how the KNN Classifier and the Decision Tree classifier (topic of a subsequent lecture) behave if we re-scale by Standardization

In [6]:
```
kn = tmh.KNN_Helper()

_ = kn.plot_classifiers(scale=True, num_ds=2)
```



- Both features now on *identical* scale (Input plot is now of Standardized features)
- The fit of the KNN Classifier is more sensible
    - And Accuracy improves relative to the raw, unscaled input
- The fit and Accuracy of the Decision Tree Classifier is unaffected

So

- The KNN model **is** sensitive to scale
- The Decision Tree model **is not** sensitive to scale. In contrast, the Decision Tree classifier (which will be the topic of a subsequent lecture)

Differences in scale might inadvertently creep into the data.

- Millimeters rather than meters
- Whole dollars rather than cents (hundredths of a dollar)

Bottom line

- Know the math behind your model ! Don't blindly "use an API".

We will encounter scale sensitivity once more

- Upcoming lecture on Principal Components
- Results change when we use
    - The Correlation matrix (i.e., scaled data)
    - The Covariance matrix (i.e., unscaled data)

Parameters sensitive to scale

Consider the linear model

$$\mathbf{y} = \Theta_1 * \mathbf{x}_1$$

If we inflate $\mathbf{x}_1$ by a factor of 10 then Θ_1 will mathematically need to decrease by the same factor

$$\begin{aligned} \mathbf{y} &= \frac{\Theta_1}{10} * (10 * \mathbf{x}_1) \\ &= \Theta'_1 * \mathbf{x}'_1 \end{aligned}$$

Changing the scale of the feature *also* changes the scale of the corresponding parameter.

Why might this matter ?

Again: there are Loss functions where the scale of **parameters** (rather than features) matter.

Loss functions sensitive to scale: regularization penalty

When a model has too many parameters (always count them!)

- Many elements of the Θ vector may be near 0

Recall too the symptom of the Dummy Variable trap

- Pairs of parameters with similar absolute magnitude but opposite sign

Regularization is a technique

- That modifies the Loss function
- By adding a *penalty*
- With the effect of encouraging small, near zero parameter values

That is, the Loss function becomes a *Regularized Loss*
$$\mathcal{L} + (\alpha * Q)$$
where

- Q is a penalty that is a function of Θ
- α is the *strength* of the penalty
- $\mathcal{L}$ is the unregularized Loss function

(See our Deep Dive notebook section [Regularization (Bias_and_Variance.ipynb#Regularization:-reducing-overfitting)](Bias_and_Variance.ipynb#Regularization:-reducing-overfitting) for more detail.

Because scaling a feature may rescale the *corresponding parameter*

- The penalty Q will be affected
- And so too the regularized Loss

Ridge Regression (external/PythonDataScienceHandbook/notebooks/05.06-Linear-Regression.ipynb#Ridge-regression-%28L_2-Regularization%29) is basically Linear Regression with a penalty that is the sum of squared parameters values

$$Q = \sum_{n=1}^{N} \Theta_n^2$$

In our example

- Inflating the sole parameter $\mathbf{x}_1$ by a factor of 10
- Decreases the corresponding parameter Θ_1 by a factor of 10
- Decreases the corresponding penalty by a factor of 100

Note that even something as "benign" as changing units from meters to millimeters has a big effect.

Lasso Regression (external/PythonDataScienceHandbook/notebooks/05.06-Linear-Regression.ipynb#Lasso-regression-%28L_1-regularization%29) is Linear Regression with a penalty that is the sum of the absolute value of parameters

$$Q = \alpha \sum_{n=1}^{N} |\Theta_n|$$

In our example

- Inflating the sole $\mathbf{x}_1$ parameter by a factor of 10
- Decreases the corresponding parameter Θ_1 by a factor of 10
- Decreases the corresponding penalty by a factor of 10

It is easy to forget (and get burned !) by the Scaling transformation when you innocently add a Regularization penalty to the loss.

Feature scaling: summary

- A very simple operation
- With subtle but significant impact

Preview:

- In the Deep Learning part of the course
- We will see the need to scale features
    - Small magnitude

```
In [12]: print("Done")
```

```
Done
```