# Table of Contents



In [5]:
```
# imports from custom library
import sys
sys.path.append('../')
import autograd.numpy as np
import matplotlib.pyplot as plt
import copy
datapath = '../mlrefined_datasets/nonlinear_superlearn_datasets/'

# import custom libraries
from mlrefined_libraries import nonlinear_superlearn_library as nonlib

# this is needed to compensate for %matplotlib notebook's tendancy to blow up ima
ges when plotted inline
from matplotlib import rcParams
rcParams['figure.autolayout'] = True
%matplotlib notebook
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

# Exercise 14.1. Growing deep trees by addition

Adding $2^D - 1$ one-dimensional stumps (if they do not share any split points) create $2^D - 1$ unique split points. Now take the midpoint. It creates two branches, one to its left and the other two its right. So far we have accounted for $1$ split point and a depth $1$ tree. We then take the midpoint on each branch, creating a depth $2$ tree with a total of $1 + 2$ split points. Following this pattern we will end up using all $1 + 2 + 4 + \cdots + 2^k$ split points and create a depth $k + 1$ tree.

All left to do is express $k$ in terms of $D$:

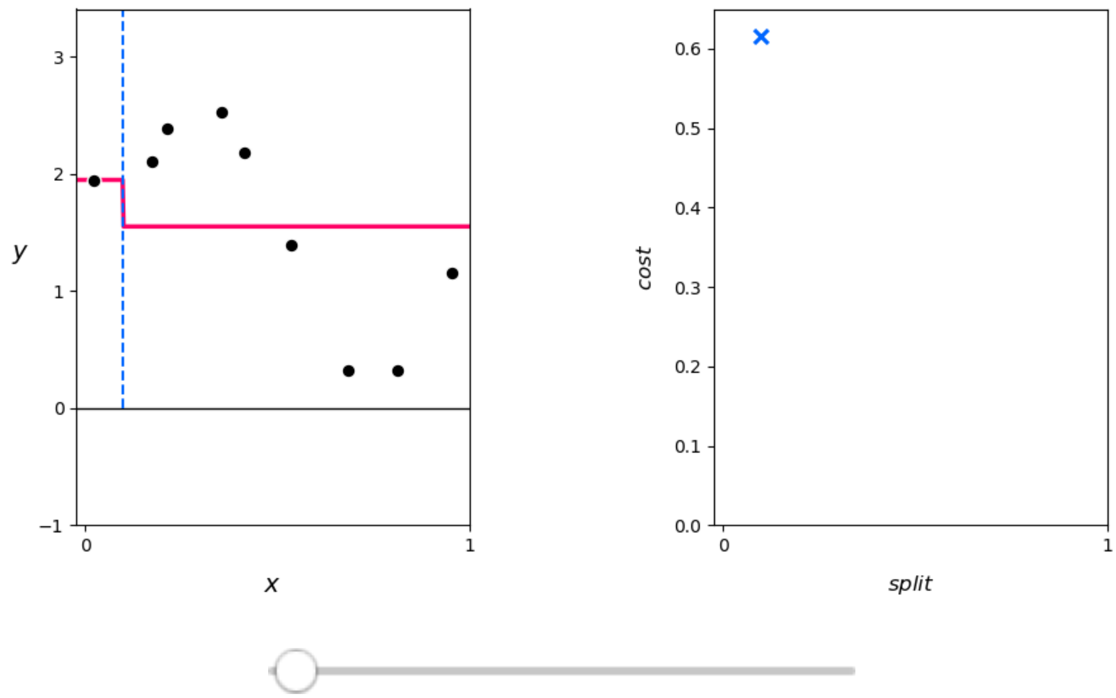
$$1 + 2 + 4 + \cdots + 2^k = 2^{k+1} - 1 = 2^D - 1 \quad \Rightarrow \quad k = D - 1$$

Therefore, adding $2^D - 1$ stumps will create a tree of depth $k + 1 = D$.

# Exercise 14.2. Fitting the parameters of a simple regression tree

In [6]:
```
## This code cell will not be shown in the HTML version of this notebook
# animate stump collection for a sample dataset
demo = nonlib.stump_visualizer_2d.Visualizer()
csvname = datapath + 'noisy_sin_subsample_raised_2.csv'
demo.load_data(csvname)
demo.browse_stumps()
```

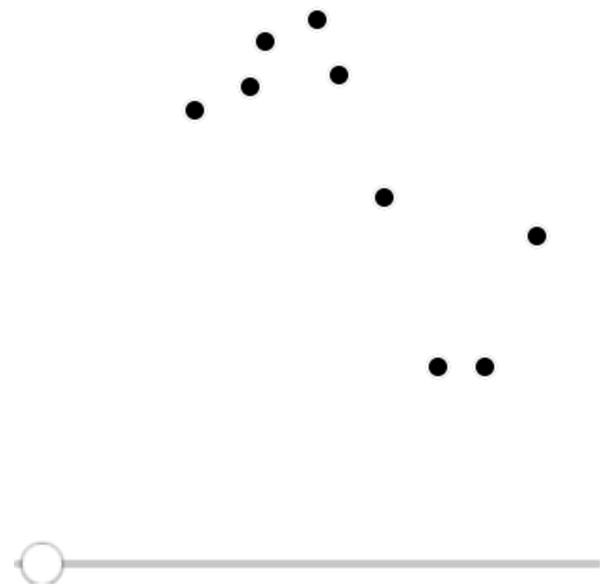
Out[6]:




## Exercise 14.3. Code up a regression tree

In [27]:
```
## This code cell will not be shown in the HTML version of this notebook
# create regression tree
csvname = datapath + 'noisy_sin_subsample_2.csv'
depth = 5
tree = nonlib.recursive_tree_lib.RegressionTree.RTree(csvname,depth)

# animate growth
demo = nonlib.recursive_tree_lib.regression_animator.Visualizer(csvname)
frames = depth
demo.animate_trees(tree)
```

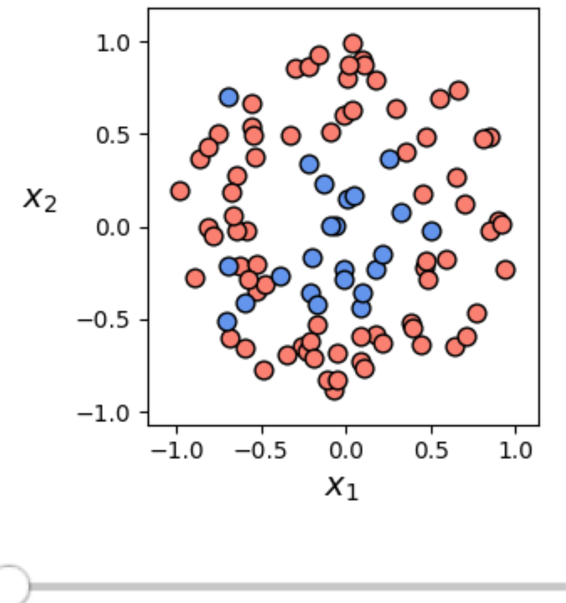
Out[27]:



# Exercise 14.4. Code up a two-class classification tree

In [17]:
```
## This code cell will not be shown in the HTML version of this notebook
# learn classification tree for input dataset
csvname = datapath + 'new_circle_data.csv'
depth = 7
tree = nonlib.recursive_tree_lib.ClassificationTree.RTree(csvname,depth)

# animate growth
demo = nonlib.recursive_tree_lib.classification_animator.Visualizer(csvname)
demo.animate_trees(tree)
```

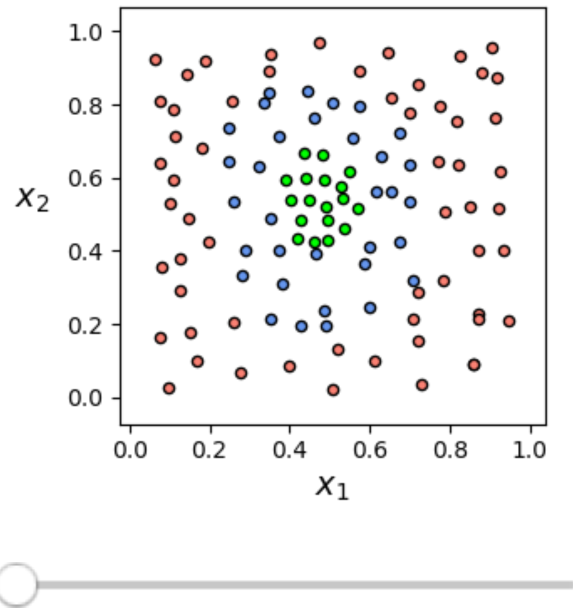
Out[17]:




# Exercise 14.5. Code up a multi-class classification tree

In [27]:
```
## This code cell will not be shown in the HTML version of this notebook
# learn classification tree for input dataset
csvname = datapath + '3_layercake_data.csv'

# learn classification tree for input dataset
depth = 7
tree = nonlib.recursive_tree_lib.ClassificationTree.RTree(csvname,depth)

# animate growth
demo = nonlib.recursive_tree_lib.classification_animator.Visualizer(csvname)
demo.animate_trees(tree,pt_size = 20)
```

Out[27]:

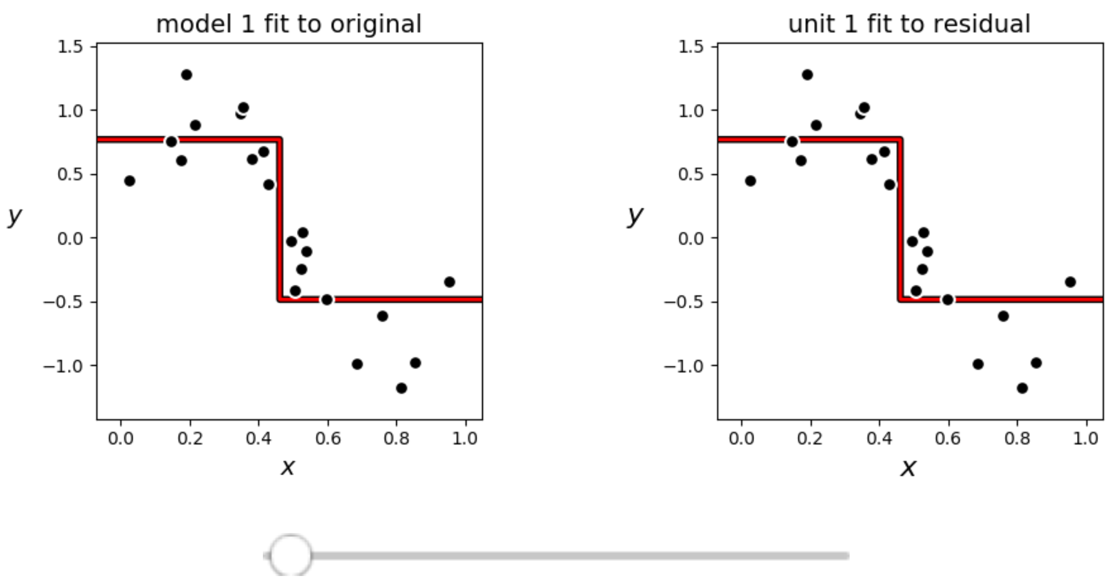



## Exercise 14.6. Gradient boosting for regression

In [ ]:
```
## This code cell will not be shown in the HTML version of this notebook
# load in dataset
csvname = datapath + 'universal_regression_samples_0.csv'
csvname = datapath + 'noisy_sin_sample.csv'

data = np.loadtxt(csvname,delimiter = ',')
x = copy.deepcopy(data[:-1,:])
y = copy.deepcopy(data[-1:,:] )

# boosting procedure
num_units = 40
runs2 = []
for j in range(num_units):
    # import the v1 library
    mylib2 = nonlib.boost_lib3.stump_booster.Setup(x,y)

    # choose normalizer
    mylib2.choose_normalizer(name = 'standard')

    # choose normalizer
    mylib2.make_train_valid_split(train_portion = 1)

    # choose cost|
    mylib2.choose_cost(name = 'least_squares')

    # choose optimizer
    mylib2.choose_optimizer('newtons_method',max_its=1)

    # run boosting
    mylib2.boost(1,verbose=False)
    mylib2.model = mylib2.models[-1]

    # add model to list
    runs2.append(copy.deepcopy(mylib2))

    # cut off output given model
    normalizer = mylib2.normalizer
    ind = np.argmin(mylib2.train_cost_vals[0])
    y_pred =  mylib2.models[-1](mylib2.normalizer(x))
    y -= y_pred

# animate the business
frames = num_units
demo2 = nonlib.boosting_regression_animators_v3.Visualizer(csvname)
demo2.animate_boosting(runs2,frames)
```

In [24]:
```
## This code cell will not be shown in the HTML version of this notebook
# load in dataset
csvname = datapath + 'universal_regression_samples_0.csv'
csvname = datapath + 'noisy_sin_sample.csv'

data = np.loadtxt(csvname,delimiter = ',')
x = copy.deepcopy(data[:-1,:])
y = copy.deepcopy(data[-1:,:] )

# boosting procedure
num_units = 40
runs2 = []
for j in range(num_units):
    # import the v1 library
    mylib2 = nonlib.boost_lib3.stump_booster.Setup(x,y)

    # choose normalizer
    mylib2.choose_normalizer(name = 'standard')

    # choose normalizer
    mylib2.make_train_valid_split(train_portion = 1)

    # choose cost|
    mylib2.choose_cost(name = 'least_squares')

    # choose optimizer
    mylib2.choose_optimizer('newtons_method',max_its=1)

    # run boosting
    mylib2.boost(1,verbose=False)
    mylib2.model = mylib2.models[-1]

    # add model to list
    runs2.append(copy.deepcopy(mylib2))

    # cut off output given model
    normalizer = mylib2.normalizer
    ind = np.argmin(mylib2.train_cost_vals[0])
    y_pred =  mylib2.models[-1](mylib2.normalizer(x))
    y -= y_pred

# animate the business
frames = num_units
demo2 = nonlib.boosting_regression_animators_v3.Visualizer(csvname)
demo2.animate_boosting(runs2,frames)
```

Out[24]:




## Exercise 14.7. Gradient boosting for classification

A signle step of Newton's method to minimize the cost function

$$g(v_L) = \frac{1}{|\Omega_L|} \sum_{p \in \Omega_L} \log\left(1 + e^{-y_p\left(\text{model}_{m-1}\left(\mathbf{x}_p, \Theta_{m-1}\right) + v_L\right)}\right)$$

takes the form

$$g(v_L) = v_L^0 - \frac{g'(v_L^0)}{g''(v_L^0)}$$

where $v_L^0$ is the starting point, and the first and second order derivative of $g$ at this point can be computed, respectively, as

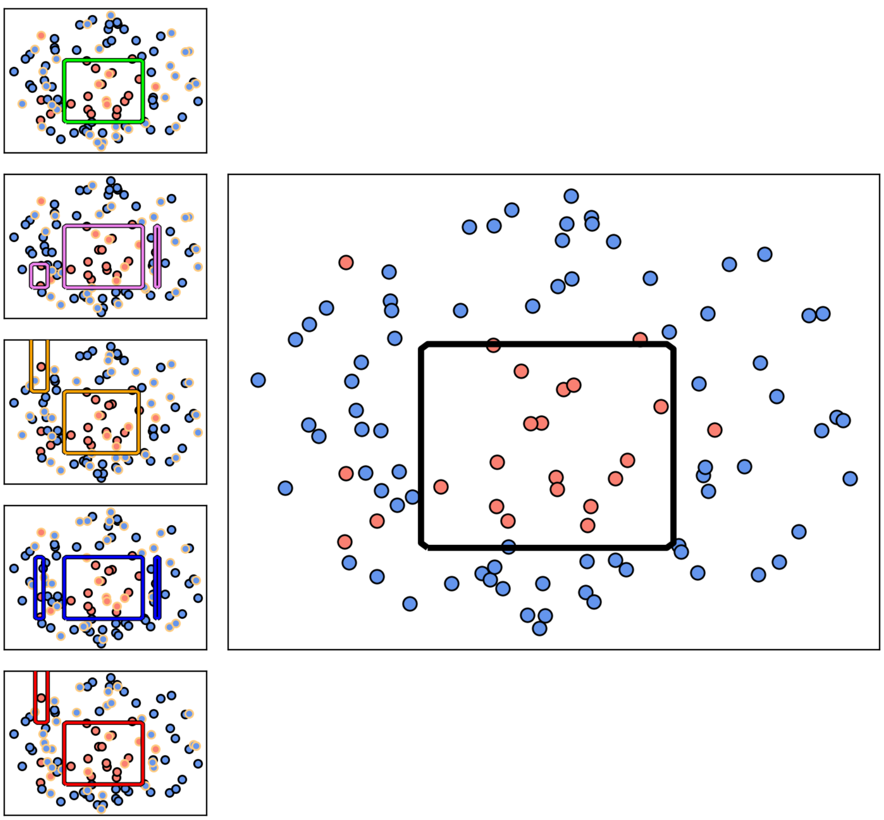
$$g'(v_L^0) = \frac{1}{|\Omega_L|} \sum_{p \in \Omega_L} \frac{-y_p\, e^{-y_p\left(\text{model}_{m-1}\left(\mathbf{x}_p, \Theta_{m-1}\right) + v_L^0\right)}}{1 + e^{-y_p\left(\text{model}_{m-1}\left(\mathbf{x}_p, \Theta_{m-1}\right) + v_L^0\right)}}$$

and

$$g''(v_L^0) = \frac{1}{|\Omega_L|} \sum_{p \in \Omega_L} \frac{e^{-y_p\left(\text{model}_{m-1}\left(\mathbf{x}_p, \Theta_{m-1}\right) + v_L^0\right)}}{\left(1 + e^{-y_p\left(\text{model}_{m-1}\left(\mathbf{x}_p, \Theta_{m-1}\right) + v_L^0\right)}\right)^2}.$$

## Exercise 14.8. Random forests

In [5]:
```
## This code cell will not be shown in the HTML version of this notebook
# path to data, container for trees
datapath = '../../mlrefined_datasets/nonlinear_superlearn_datasets/'
csvname = datapath + 'new_circle_data.csv'
trees = []
num_trees = 5
depth = 7
train_portion = 0.66

for i in range(num_trees):
    tree = nonlib.recursive_tree_lib_crossval.ClassificationTree.RTree(csvname,de
pth,train_portion=train_portion)
    trees.append(tree)

animator = nonlib.recursive_tree_lib_crossval.classification_ensembler.Visualizer
(csvname)
animator.show_runs(trees)
```



## Exercise 14.9. Limitation of trees outside their training range

Any tree, or ensemble of trees, fit to such a dataset will not allow for reliable predictions to be made *outside of the input region containing the original training data*. This is because outside of where the original training data is defined the tree predicts output *using a depth 1 tree* (a leaf in the case of a single tree). In other words, all predictiions made outside of the original region of input are *constant*.

With the student data our input is *time*, and so any tree (or ensemble of trees) will produce a *constant* prediction for all future time periods outside the region of our original input data.

# Exercise 14.10. Naive cross-validation

In [22]:
```
## This code cell will not be shown in the HTML version of this notebook
# create root stump
csvname = datapath + 'noisy_sin_sample.csv'
depth = 7
tree = nonlib.recursive_tree_lib_crossval.RegressionTree.RTree(csvname,depth,trai
n_portion = 0.66)

# animate
demo = nonlib.recursive_tree_lib_crossval.regression_animator.Visualizer(csvname)
demo.animate_trees(tree)
```

Out[22]: