Tree Strategy HW1 Q1, Q2

January 21, 2024

0.1 Imports

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.metrics import accuracy_score, confusion_matrix
pd.set_option('use_inf_as_na', True)
from collections import Counter
```

```
/var/folders/sp/wlr6xm297918vx6kjh2z1dk00000gn/T/ipykernel_58828/2166773765.py:6
: FutureWarning: use_inf_as_na option is deprecated and will be removed in a
future version. Convert inf values to NaN before operating instead.
  pd.set_option('use_inf_as_na', True)
```

0.2 Loading the Data Set (you need to put in the file where you have stored the data)

[2]:
```
raw_data = pd.read_pickle('dataset.pkl')
```

[4]:
```
raw_data = raw_data.drop([x  for x in raw_data.columns if 'fqtr' in x],axis=1)
```

0.3 Restricting to Companies with Market Cap > 1 Billion

[5]:
```
data = raw_data[raw_data['market_cap'] > 1000.0]
```

0.4 The Total Number of Companies w/ Market Cap > 1 Billion that appear during our time horizon

[6]:
```
len(data.index.get_level_values(1).unique())
```

[6]: 4076

0.5 Filling in Missing Values

[8]:
```
data = data.copy()
data.replace([np.inf,-np.inf],np.nan,inplace=True)
data = data.fillna(method='ffill')
```

```
/var/folders/sp/wlr6xm297918vx6kjh2z1dk00000gn/T/ipykernel_58828/970161762.py:3:
FutureWarning: DataFrame.fillna with 'method' is deprecated and will raise in a
future version. Use obj.ffill() or obj.bfill() instead.
  data = data.fillna(method='ffill')
```

[9]:
```
data = data.fillna(0)
```

[10]:
```
data['pred_rel_return']
```

[10]:
```
date        ticker
2000-02-09  CSCO     -0.025923
            ROP       0.066175
2000-02-10  CMOS      0.241345
2000-02-11  DELL      0.306035
2000-02-15  VAL       0.043852
                        …
2018-12-21  NKE      -0.100100
            SAFM     -0.100100
            SCHL     -0.100100
            WBA      -0.100100
2018-12-24  KMX      -0.100100
Name: pred_rel_return, Length: 111468, dtype: float64
```

0.6 HW Question 1

Inserting a column in the dataset where entries are 1 if stock outperforms SPY in the earnings period, and -1 otherwise:

[11]:
```
# function to return appropriate values based on performance
def f_1(x):
    if x > 0:
        return 1
    else:
        return -1
```

[12]:
```
# apply the function to the column of relative returns
data = data.copy()
data['rel_performance_1'] = data['pred_rel_return'].apply(f_1)
```

This is the column of labels next to the original relative returns:

[13]:
```
data[['pred_rel_return', 'rel_performance_1']]
```

[13]:

```
                       pred_rel_return  rel_performance_1
date       ticker
2000-02-09 CSCO              -0.025923                 -1
           ROP                0.066175                  1
2000-02-10 CMOS               0.241345                  1
2000-02-11 DELL               0.306035                  1
2000-02-15 VAL                0.043852                  1
…                                  …                 …
2018-12-21 NKE               -0.100100                 -1
           SAFM              -0.100100                 -1
           SCHL              -0.100100                 -1
           WBA               -0.100100                 -1
2018-12-24 KMX               -0.100100                 -1

[111468 rows x 2 columns]
```

Thus we can observe that the labels for the stocks whose relative returns are postive (i.e. indiciating it outperformed the SPY) have label 1 (e.g CMOS has label 1).
Otherwise if they are negative or zero, the label is -1 (e.g. NKE has label -1).

0.7 HW Question 2

Inserting a column in the dataset where entries are:
- 2 if the stock return is more than 5% higher than the SPY return
- 1 if it is between 1% and 5% higher than the SPY return
- 0 if it is between -1% and 1% relative to the SPY return
- -1 if it is between -1% and -5% relative to the SPY return

[14]:

```python
# function to return appropriate values based on performance as detailed above
def f_2(x):
    if x > 0.05:
        return 2
    elif x > 0.01:
        return 1
    elif x > -0.01:
        return 0
    elif x > -0.05:
        return -1
```

[15]:

```python
# apply the function to the column of relative returns
data = data.copy()
data['rel_performance_2'] = data['pred_rel_return'].apply(f_2)
```

This is the column of labels next to the original relative returns:

[16]:

```python
data[['pred_rel_return', 'rel_performance_2']]
```

[16]:

```
                        pred_rel_return  rel_performance_2
date       ticker
2000-02-09 CSCO              -0.025923               -1.0
           ROP                0.066175                2.0
2000-02-10 CMOS               0.241345                2.0
2000-02-11 DELL               0.306035                2.0
2000-02-15 VAL                0.043852                1.0
…                                  …                 …
2018-12-21 NKE               -0.100100                NaN
           SAFM              -0.100100                NaN
           SCHL              -0.100100                NaN
           WBA               -0.100100                NaN
2018-12-24 KMX               -0.100100                NaN

[111468 rows x 2 columns]
```

Thus we can observe that the labels were applied correctly:
- CMOS had a relative return of 0.24135 (i.e. ~ 24.1% higher than the SPY) so its label is 2
- VAL had a relative return of 0.043852 (i.e. ~ 4.3% higher than the SPY) so its label is 1
- CSCO had a relative return of -0.025923 (i.e. ~ -2.5% relative to the SPY) so its label is -1
- NKE had a relative return of -0.100100 (i.e. ~ -10% relative to the SPY) so its label is NaN (as we have not assigned a label for those stocks with relative performance less than 5% compared to the SPY.

[17]:

```python
# Find those stocks whose relative performance 2 label is 0
data[['pred_rel_return', 'rel_performance_2']].loc[data['rel_performance_2'] ==
 ↪0]
```

[17]:

```
                        pred_rel_return  rel_performance_2
date       ticker
2000-02-24 MDT                0.005511                0.0
           ORTL               0.005511                0.0
2000-03-08 DDS               -0.004425                0.0
2000-04-13 DJ                -0.009937                0.0
2000-04-14 CYN               -0.003053                0.0
…                                  …                 …
2018-09-20 CPRT              -0.004948                0.0
2018-09-27 KMX               -0.008830                0.0
2018-10-01 MTN                0.004993                0.0
2018-10-02 CALM               0.004993                0.0
           KMG                0.004993                0.0

[7647 rows x 2 columns]
```

It remains to check that if we can observe that the label 0 was applied correctly:
- MDT had a relative return of 0.005511 (i.e. ~ 0.55% higher than the SPY) so its label is 0 as this is between -1% and 1%

Thus all labels have been applied appropriately.

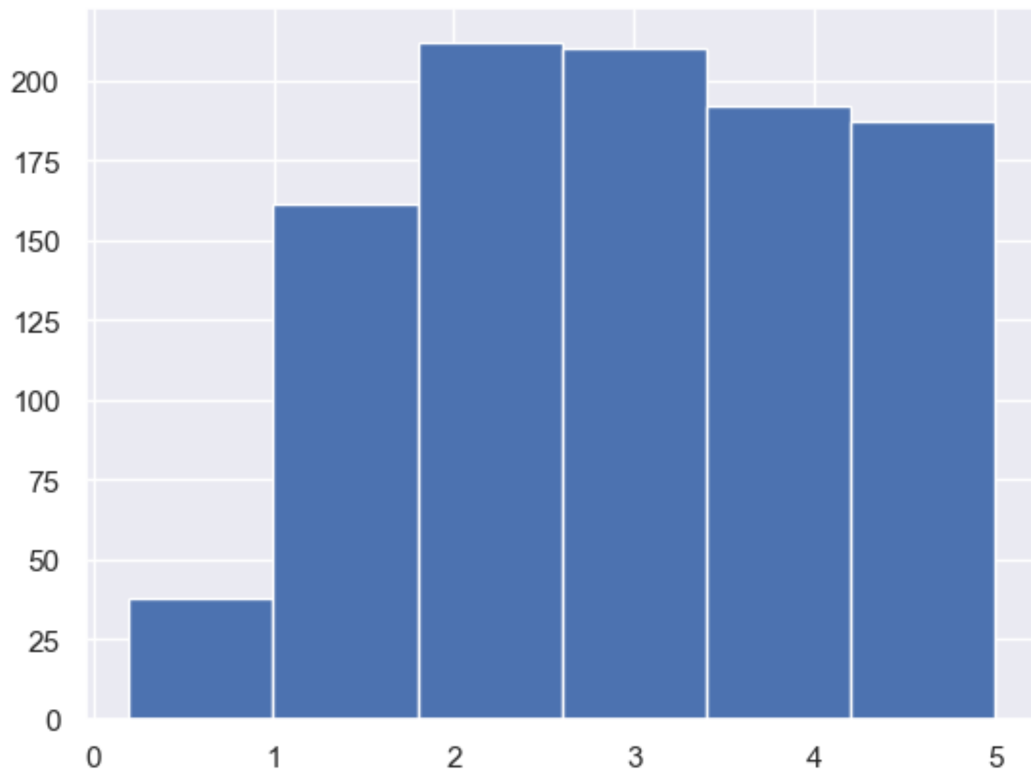
Visualizing Trees HW1 Q3

January 21, 2024

[1]:
```python
%matplotlib inline

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn import tree
from matplotlib.patches import Rectangle
from matplotlib.collections import PatchCollection
from matplotlib import cm
from collections import Counter

sns.set()
```

[2]:
```python
n = 1000
x = np.random.uniform(0, 1, n)
y = np.random.uniform(0, 1, n)
target = np.random.uniform(x+y,5)
# norm.pdf((x - 0.75) / 0.1) + norm.pdf((y - 0.75) / 0.1) \
#         + norm.pdf((x - 0.25) / 0.1) + norm.pdf((y - 0.25) / 0.1) \
#         + np.array(np.round(np.random.normal(-0.1,0.1, n), 2))
```

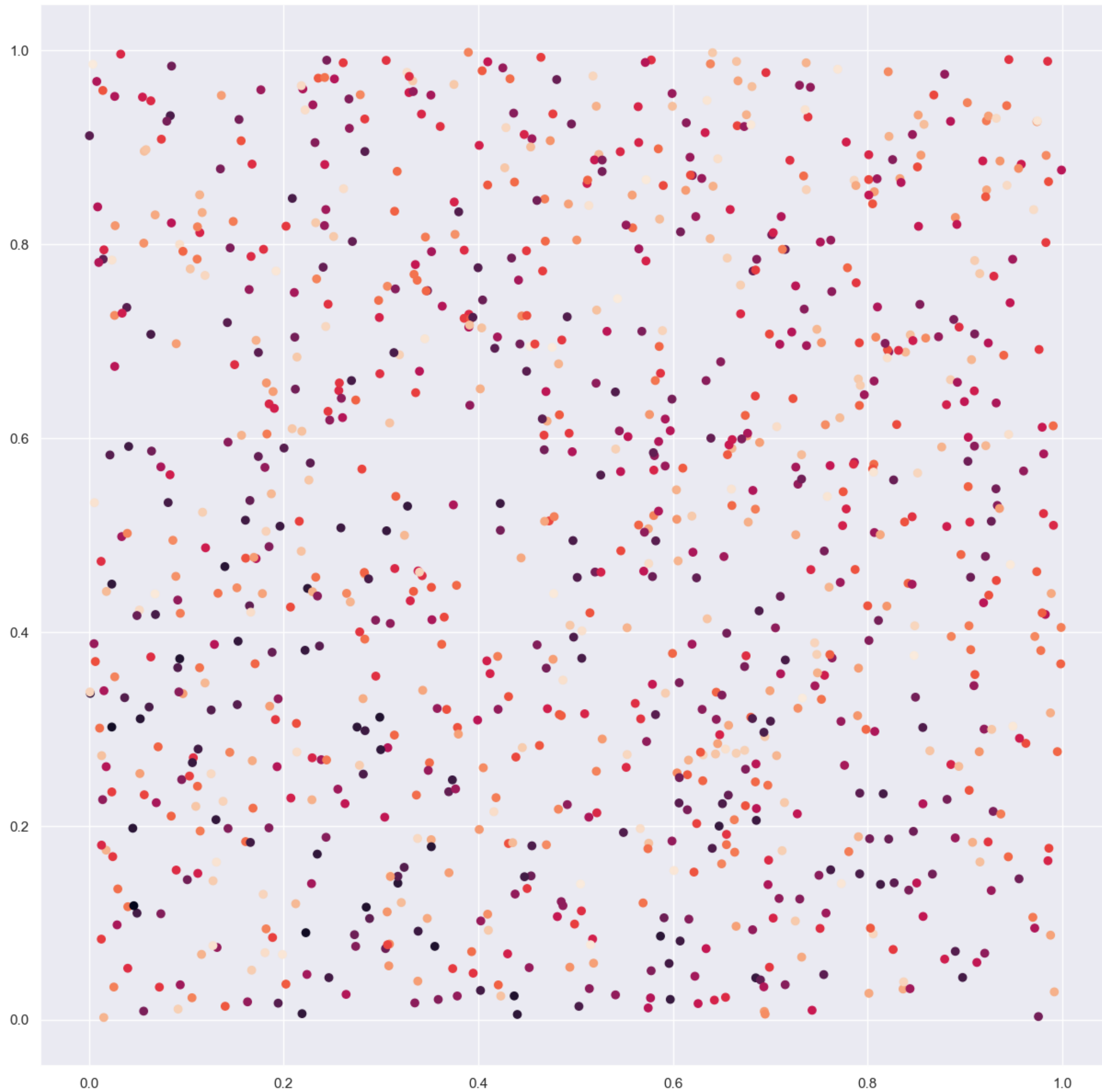
[3]:
```python
a = plt.hist(target,bins=6)[1]
```



[4]:
```
a
```

[4]:
```
array([0.19834388, 0.99850434, 1.7986648 , 2.59882526, 3.39898572,
       4.19914618, 4.99930664])
```

[5]:
```
# Plot all points
fig, ax = plt.subplots(figsize = (15,15))
ax.scatter(x, y, c = target);
```



Note: Here I use a DecisionTreeRegressor instead of a DecisionTreeClassifier.
And I no longer transform the target into labels as we want to create a regression tree.

[6]:
```
data1 = pd.DataFrame({'x' : x, 'y' : y})
tree_1 = DecisionTreeRegressor(max_depth=5,min_samples_leaf = 50,max_features=0.
 ↪5)
tree_1.fit(data1,target)
```

[6]:
```
DecisionTreeRegressor(max_depth=5, max_features=0.5, min_samples_leaf=50)
```

[7]:
```
data1
```

[7]:

```
            x         y
0    0.623607  0.828632
1    0.032622  0.996253
2    0.929928  0.767134
3    0.761656  0.571742
4    0.665185  0.274939
..        …         …
995  0.379618  0.294699
996  0.737405  0.695517
997  0.434263  0.785849
998  0.034064  0.728959
999  0.762236  0.154673

[1000 rows x 2 columns]
```

In the following function, I adapted line 83 onwards so that now:
- The function works with target (continuous data) instead of labels (categorical data)
- It takes the average of the target in each rectangle instead of the max
- A colormap is added and a normalizing function to map the continuous values to a color

After experimenting with different colormaps, I decided to use the matplotlib "turbo" map as it is also a rainbow colormap and has some advantages for visualization purposes: https://blog.research.google/2019/08/turbo-improved-rainbow-colormap-for.html

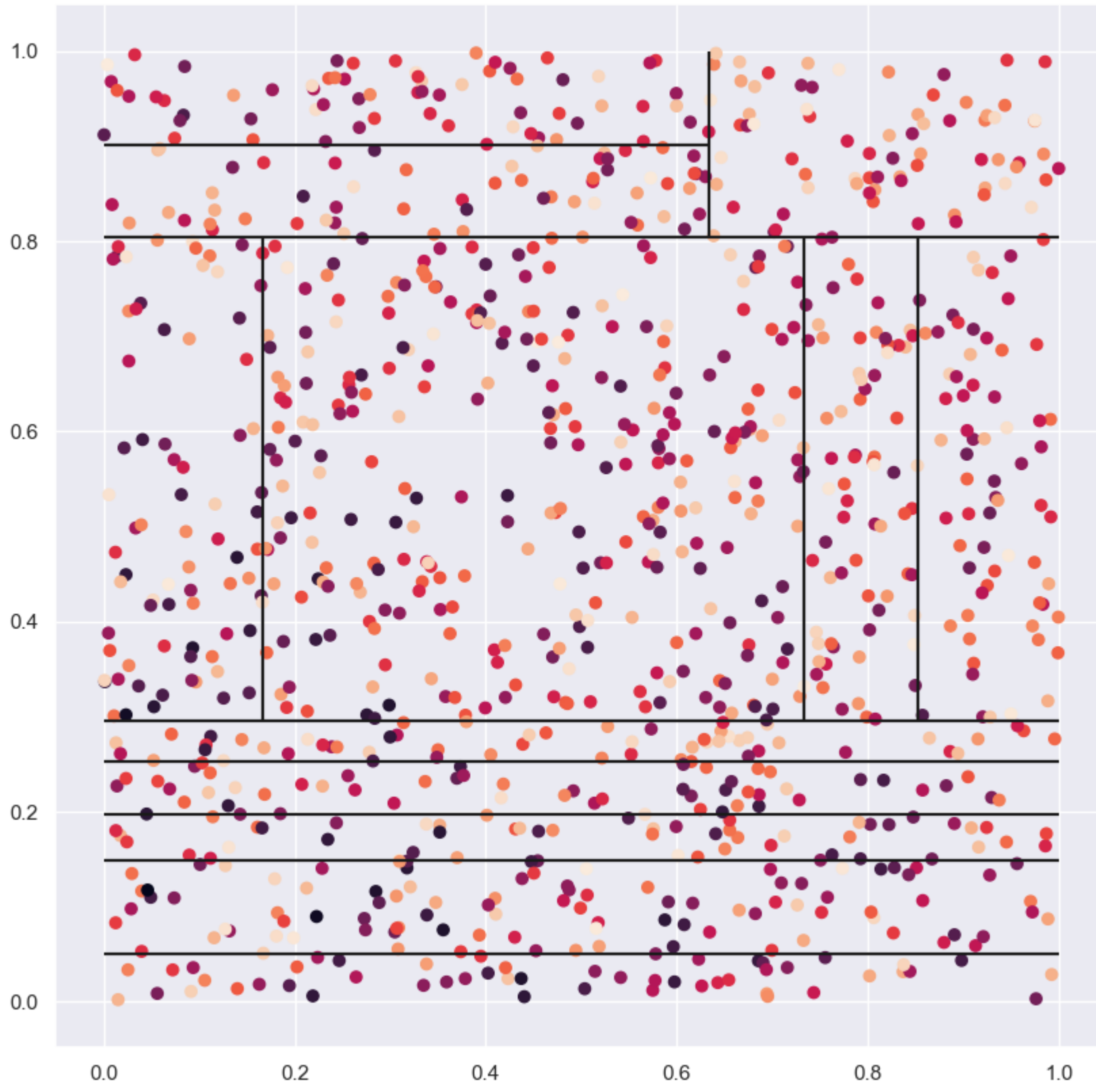
[8]:

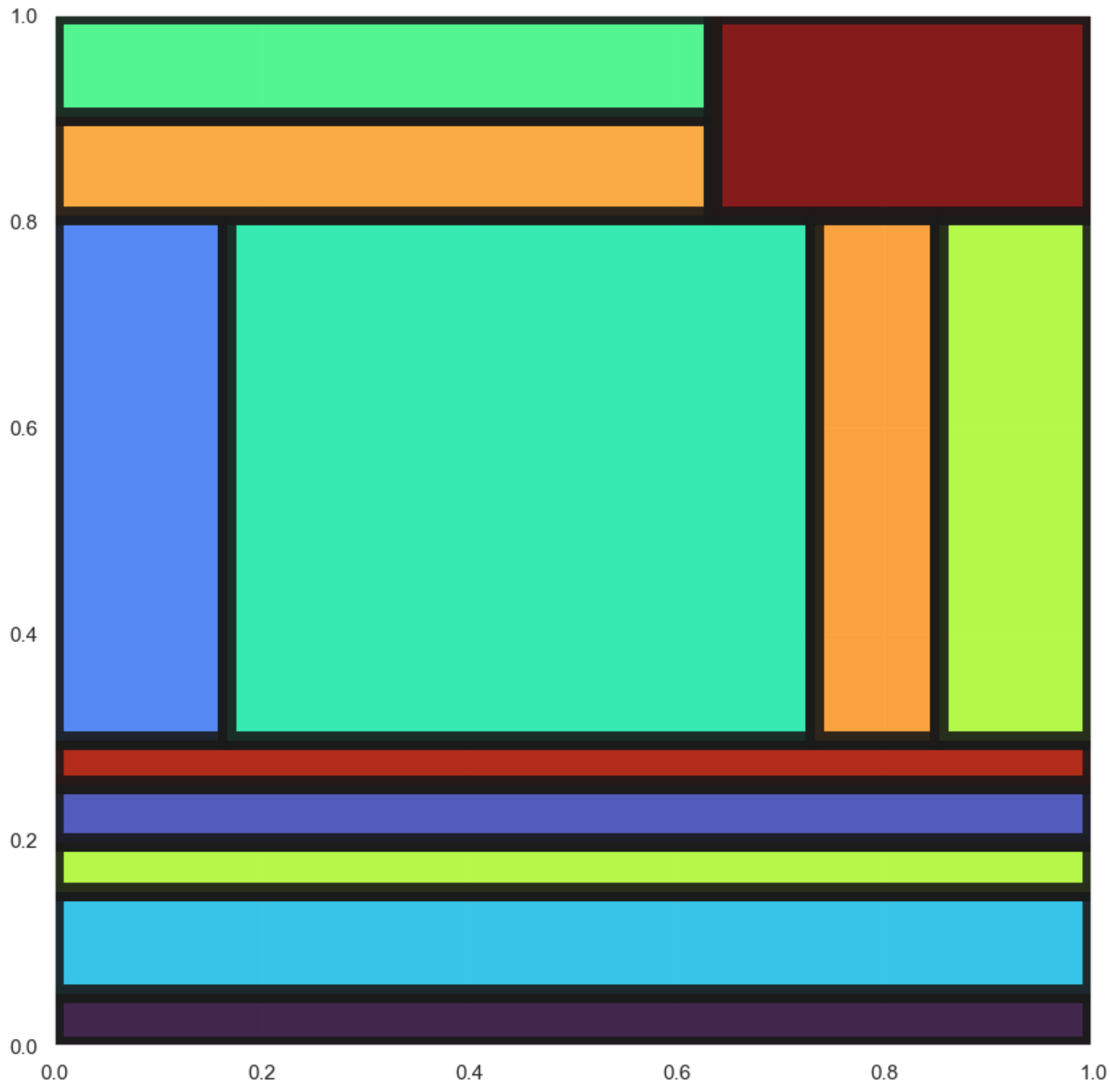
```
def boxes(tree,data,target):

    n_nodes = tree.tree_.node_count
    children_left = tree.tree_.children_left
    children_right = tree.tree_.children_right
    feature = tree.tree_.feature
    threshold = tree.tree_.threshold

    def split(i):

        left = children_left[i]
        right = children_right[i]

        return (left,right)


    def parent(i):
        splits = enumerate([split(i) for i in range(n_nodes)])
        for a,b in splits:
            if (b[0] == i) or (b[1] == i):
                return a
            else: continue
```

```
def box(i):

    (a,b),(c,d) = (0,0),(0,0)

    if i == 0:
        (a,b) = (0,0)
        (c,d) = (1,1)
    else:
        j = parent(i)
        t = threshold[j]
        (a,b),(c,d) = box(j)

        if feature[j] == 0:
            if i == split(j)[0]:
                (a,b) = (a,b)
                (c,d) = (t,d)
            else:
                (a,b) = (t,b)
                (c,d) = (c,d)

        if feature[j] == 1:
            if i == split(j)[0]:
                (a,b) = (a,b)
                (c,d) = (c,t)
            else:
                (a,b) = (a,t)
                (c,d) = (c,d)

    return (a,b),(c,d)

boxes = []
for i in range(n_nodes):
    boxes.append(box(i))

fig, ax = plt.subplots(figsize = (10,10))
ax.scatter(x, y, c = target);



for i in range(1,n_nodes):

    j = parent(i)
    t = threshold[j]
    ((a,b),(c,d)) = boxes[j]
    if feature[j] == 0:
        ax.vlines(t, b, d, colors='k')
```

```
        else:
            ax.hlines(t,a,c,colors='k')

    leaves = [x for x in range(n_nodes) if split(x) == (-1,-1)]

    leaf_rects = []
    for leaf in leaves:
        ((a,b),(c,d)) = box(leaf)
        rect = Rectangle((a,b), c - a,d - b )
        leaf_rects.append(rect)

    rect_averages = []
    for leaf in leaves:
        data_points_in_rect = []
        for i in range(len(data1)):
            p = data1.iloc[i]
            ((a,b),(c,d)) = boxes[leaf]
            if (p['x'] > a) and (p['x'] <= c) and (p['y'] > b) and (p['y'] <=␣
↪d):
                data_points_in_rect.append(i)

        rect_averages.append(np.average(target[data_points_in_rect]))

    # Normalize range of values to colormap
    cmap = plt.get_cmap('turbo')
    norm = plt.Normalize(np.min(rect_averages), np.max(rect_averages))

    facecolor = []
    for i in range(len(leaves)):
        color = cmap(norm(rect_averages[i]))
        facecolor.append(color)

    pc = PatchCollection(leaf_rects, facecolor=facecolor, alpha=0.9,
                         edgecolor='k',linewidths = (10,))

    fig,ax = plt.subplots(figsize = (10,10))
    ax.add_collection(pc);
```

[9]:
```
boxes(tree_1,data1,target)
```


1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



Note here we use a BaggingRegressor instead of a BaggingClassifier, and use a DecisionTreeRegressor as the base model.

[10]:
```
from sklearn.ensemble import BaggingRegressor
```

[11]:
```
bg_rgr = BaggingRegressor(DecisionTreeRegressor(min_samples_leaf=32),n_estimators=10,max_samples=0.4)
```

[12]:
```
bg_rgr.fit(data1,target)
```

[12]:
```
BaggingRegressor(estimator=DecisionTreeRegressor(min_samples_leaf=32),
                 max_samples=0.4)
```

[13]:
```python
trees = bg_rgr.estimators_
```

Again, in following function, I adapted line 68 onwards so that now:
- The function works with target (continuous data) instead of labels (categorical data)
- It takes the average of the target in each rectangle instead of the max
- A colormap is added and a normalizing function to map the continuous values to a color

[14]:
```python
def bagging_boxes(tree,data,labels):

    n_nodes = tree.tree_.node_count
    children_left = tree.tree_.children_left
    children_right = tree.tree_.children_right
    feature = tree.tree_.feature
    threshold = tree.tree_.threshold

    def split(i):

        left = children_left[i]
        right = children_right[i]

        return (left,right)


    def parent(i):
        splits = enumerate([split(i) for i in range(n_nodes)])
        for a,b in splits:
            if (b[0] == i) or (b[1] == i):
                return a
            else: continue



    def box(i):

        (a,b),(c,d) = (0,0),(0,0)

        if i == 0:
            (a,b) = (0,0)
            (c,d) = (1,1)
        else:
            j = parent(i)
            t = threshold[j]
            (a,b),(c,d) = box(j)

            if feature[j] == 0:
                if i == split(j)[0]:
                    (a,b) = (a,b)
                    (c,d) = (t,d)
```

```
                else:
                    (a,b) = (t,b)
                    (c,d) = (c,d)

            if feature[j] == 1:
                if i == split(j)[0]:
                    (a,b) = (a,b)
                    (c,d) = (c,t)
                else:
                    (a,b) = (a,t)
                    (c,d) = (c,d)

        return (a,b),(c,d)

    boxes = []
    for i in range(n_nodes):
        boxes.append(box(i))

    leaves = [x for x in range(n_nodes) if split(x) == (-1,-1)]

    leaf_rects = []
    for leaf in leaves:
        ((a,b),(c,d)) = box(leaf)
        rect = Rectangle((a,b), c - a,d - b )
        leaf_rects.append(rect)

    rect_averages = []
    for leaf in leaves:
        points_in_rect = []
        for i in range(len(data1)):
            p = data1.iloc[i]
            ((a,b),(c,d)) = boxes[leaf]
            if (p['x'] > a) and (p['x'] <= c) and (p['y'] > b) and (p['y'] <=␣
↪d):
                points_in_rect.append(i)
        rect_averages.append(np.average(target[points_in_rect]))

     # Normalize range of values to colormap
    cmap = plt.get_cmap('turbo')
    norm = plt.Normalize(np.min(rect_averages), np.max(rect_averages))

    facecolor = []
    for i in range(len(leaves)):
        color = cmap(norm(rect_averages[i]))
        facecolor.append(color)
```

```python
    pc = PatchCollection(leaf_rects, facecolor=facecolor, alpha=0.1,
                         edgecolor='k',linewidths = (2,))

    ax.add_collection(pc);
```

[15]:
```python
fig, ax = plt.subplots(figsize = (10,10))

for tree in trees:
    bagging_boxes(tree,data1,target)
```