Machine Learning

Lecture 1

We will be using Machine Learning tools to design and examine a specific trading strategy.

The question we want to answer is whether we can use information from a company's Earnings Release to predict whether the company's stock will out perform or under perform the SP500 index over the period between the earnings releases.

This involves collecting data, structure the data, training the models to produce predictions and test the models on unseen data to see how well they predict

We are first going to discuss *Decision Trees*

A decision tree is a model that sends a data point through a sequence of yes/no *decision nodes.* Where the data point eventually ends up is what determines the prediction of the model

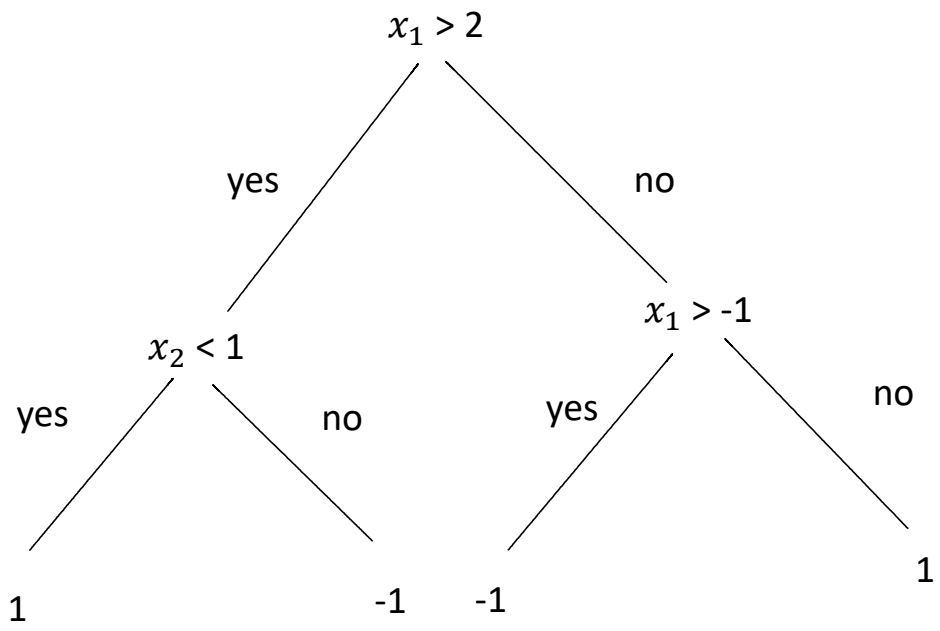
Here is a simple example:

Consider 2-dimensional data vectors (x_1, x_2)

Question 1. Is $x_1 > 2$

Question 2. if yes is $x_2 < 1$, if no is $x_1 > -1$

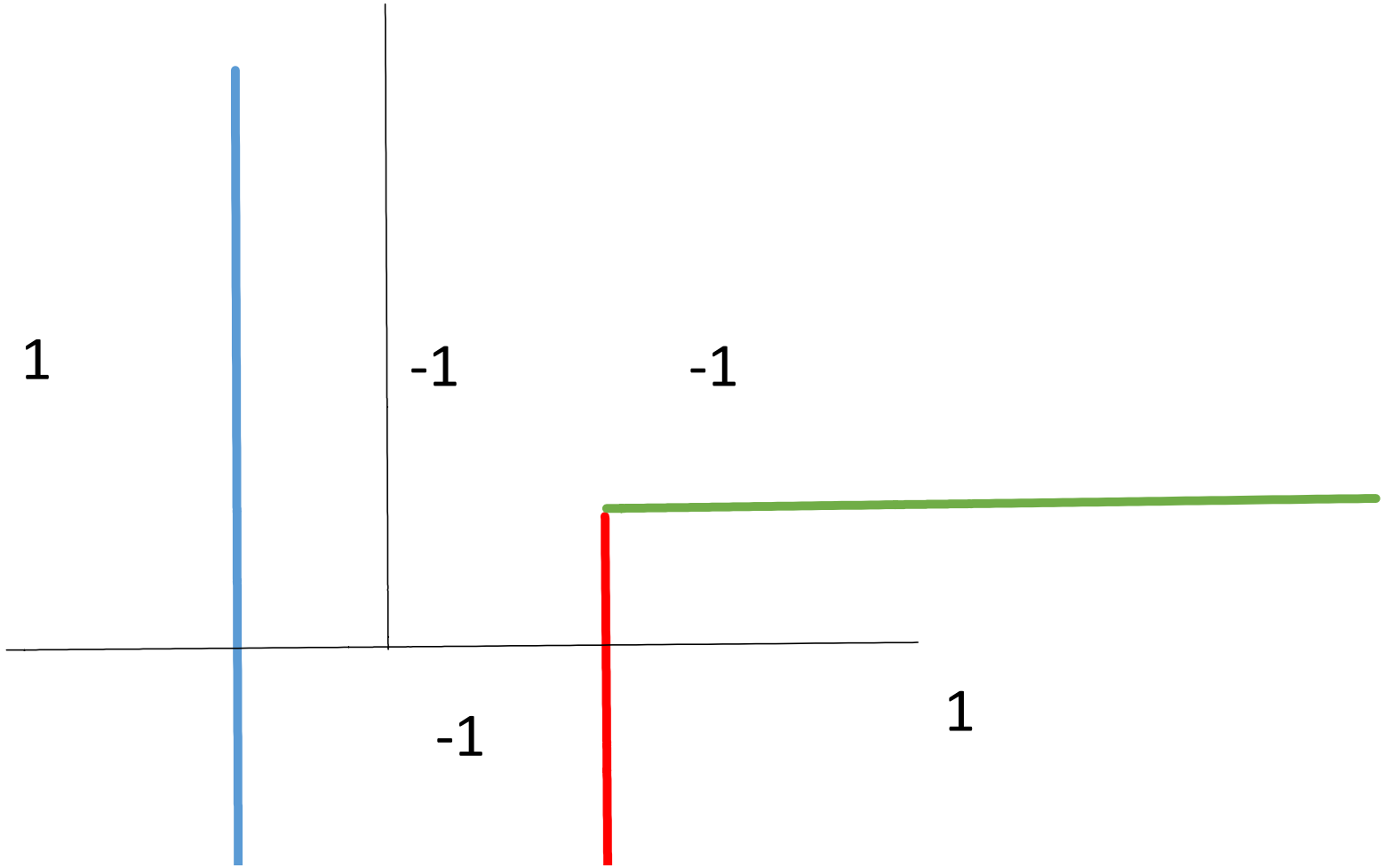
if yes, yes predict 1, if yes,no predict -1, if no,yes predict -1, if no,no predict 1



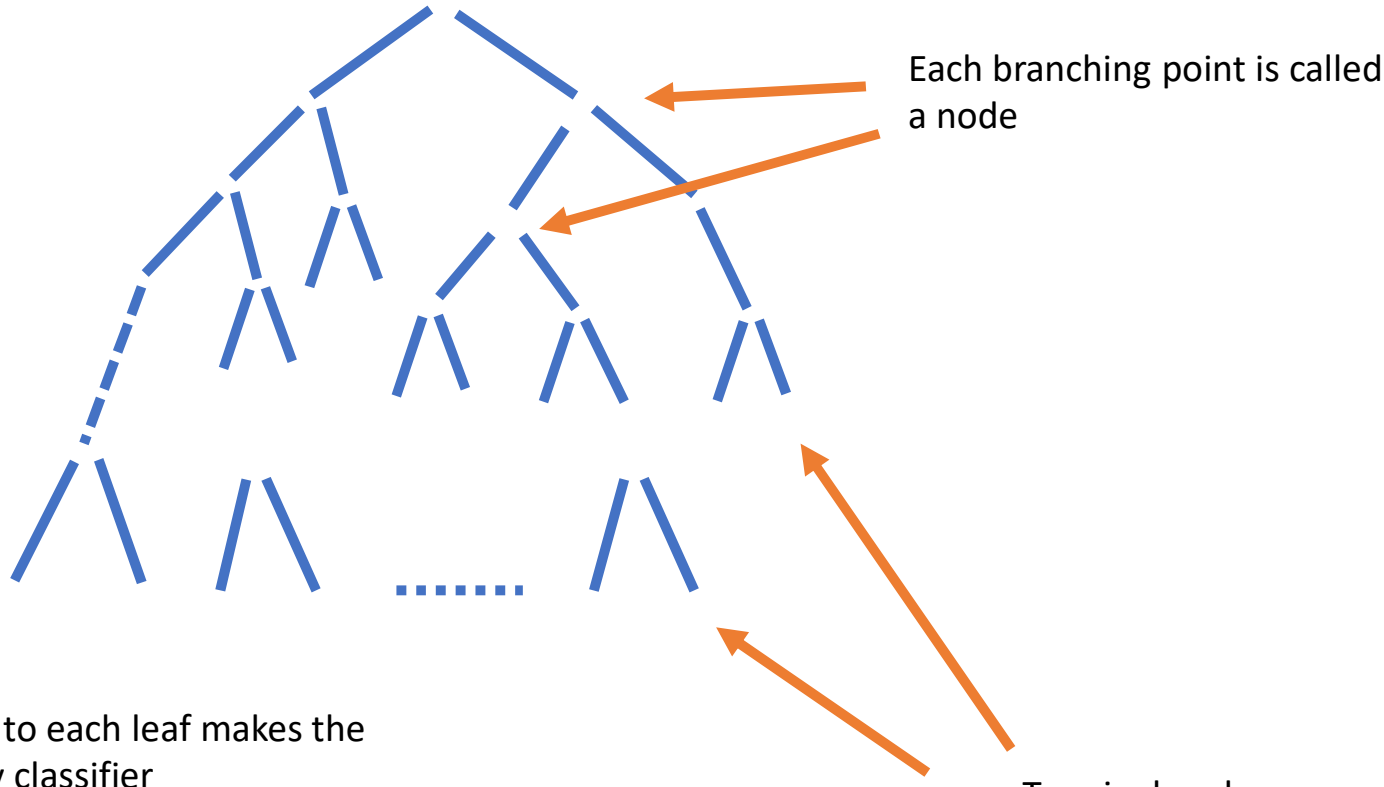
(3,-1) goes along the path yes,yes and hence the prediction is 1.

(1,0) goes along the path no,yes and so the prediction is -1

A tree determines a partition of the data space. In the example it looks like this



A tree can also be a classifier for a multi-category case



Assigning a category to each leaf makes the tree a multi-category classifier

How do we train a tree classifier given a labelled training set?

The nodes in a tree are also known as the *splits* because at each node we further split the data set into two groups.

Suppose we are given a set of labelled data vectors $\{(\underline{x}_1, y_{i_1}), (\underline{x}_2, y_{i_2}), \ldots, (\underline{x}_N, y_{i_N})\}$, where the labels lie in some set of categories $\{y_1, y_2, \ldots, y_k\}$

We shall describe how we sequentially build up a decision tree from this data set.

Assume we have partially constructed a tree and we are at a terminal node in the partially constructed tree. We have to decide how to construct a split optimally. Let $\{(\underline{x}, y)\}$ be the subset of the data in the node (for instance if we are at the root of the tree this will be the entire data set). Let m be the number of data in the node. For each category y_i let m_i be the number of data points with label y_i so $\sum_i m_i = m$

Let $p_i = \frac{m_i}{m}$, we would then classify the data points in the node to be in category y_j where $j = \underset{i}{\text{argmax}}\ p_i$ i.e. the majority category in the node. We define three *impurity* measures for the fit at the node

- Misclassification error: $\sum_{i \neq j} p_i = 1 - p_j$
- Gini index: $\sum_{i \neq i'} p_i p_{i'} = \sum_i p_i(1 - p_i)$
- Entropy: $-\sum_i p_i \log p_i$

Now we split the node into two child nodes L and R let m_L and m_R be the number of data points in L and R resp. and define the impurity of the split as

$$\frac{m_L}{m} \times \text{ impurity } L + \frac{m_R}{m} \times \text{ impurity } R$$

We then split the node into the two child nodes that minimizes the impurity of the split

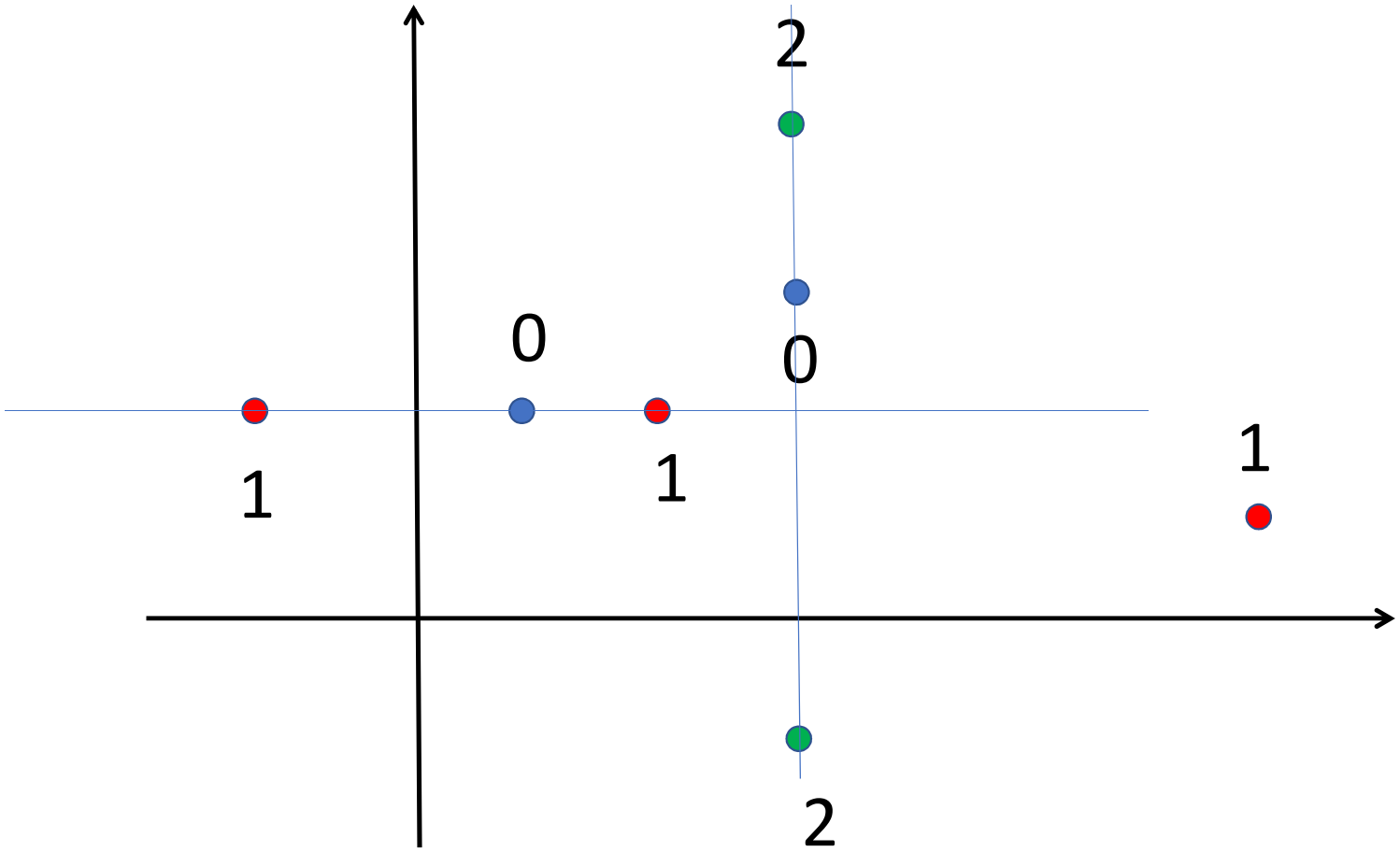
Example:

Consider the data set $((1,2),0),((3,3),0),((-1,2),1),((6,1),1),((2,2),1),((3,-1),2),((3,4),2)$ so there are 3 categories $\{0,1,2\}$ and 7 data points. The proportions are $p_0 = \frac{2}{7}, p_1 = \frac{3}{7}, p_2 = \frac{2}{7}$. The Gini index is $\frac{2}{7} \times \frac{5}{7} + \frac{3}{7} \times \frac{4}{7} + \frac{2}{7} \times \frac{5}{7} = \frac{32}{49}$.

We do not want to split at coordinates of the data points since that would put the data points on the decision boundary and we want the data points to have a reasonable margin from the decision boundary. So we instead split at the midpoints between the coordinates but excluding those that are also coordinates of data points.

Let's say we propose to split at $x_2 = 2.5$ so we split into $L = \{(x_1, x_2), x_2 \leq 2.5\}$ and $R = \{(x_1, x_2), x_2 \geq 2.5\}$. So $L = \{((1,2),0),((-1,2),1),((6,1),1),((2,1),1),(3,-1),2)\}$ and $R = \{((3,3),0),((3,4),2)\}$

The Gini index of L is $\frac{1}{5} \times \frac{4}{5} + \frac{3}{5} \times \frac{2}{5} + \frac{1}{5} \times \frac{4}{5} = \frac{14}{25}$ and the Gini index of R is $\frac{1}{2}$. Thus the Gini index of this split is $\frac{5}{7} \times \frac{14}{25} + \frac{2}{7} \times \frac{1}{2} = \frac{13}{20}$


2
0
0
1
1
1
2

We can easily write a small program that returns the optimal split

```
def compute_gini(labels):

    m = len(labels)
    c = Counter(labels)
    probs = [ c[i]/m for i in set(labels)]
    return sum([ p * (1 - p) for p in probs])
```

```
def optimal_split(data,labels):

    m = len(data)
    x_coord = np.array(data)[:,0]
    y_coord = np.array(data)[:,1]
    midpoints = [ 0.5*(data[i][0] + data[j][0]) for i in range(m) for j in range(i+1,m)],\
                [ 0.5*(data[i][1] + data[j][1]) for i in range(m) for j in range(i+1,m)]
    exclusive_midpoints = [ x for x in midpoints[0] if x not in x_coord] ,\
                          [ y for y in midpoints[1] if y not in y_coord]

    splits = [(0,exclusive_midpoints[0][i]) for i in range(len(exclusive_midpoints[0]))] +\
             [(1,exclusive_midpoints[1][i]) for i in range(len(exclusive_midpoints[1]))]
    split_indices = []

    for s in splits:
        if s[0] == 0:
            L = [labels[i] for i in range(m) if data[i][0] <= s[1]]
            R = [labels[i] for i in range(m) if data[i][0] > s[1]]

            if len(L) > 0 and len(R) > 0:
                g = len(L) * compute_gini(L) + len(R) * compute_gini(R)
                split_indices.append(((s,g/m)))

        else:

            L = [labels[i] for i in range(m) if data[i][1] <= s[1]]
            R = [labels[i] for i in range(m) if data[i][1] > s[1]]
            if len(L) > 0 and len(R) > 0:
                g = len(L) * compute_gini(L) + len(R) * compute_gini(R)
                split_indices.append(((s,g/m)))

    return sorted(split_indices,key = lambda x: x[1])[0]
```

If we run that on our data we get

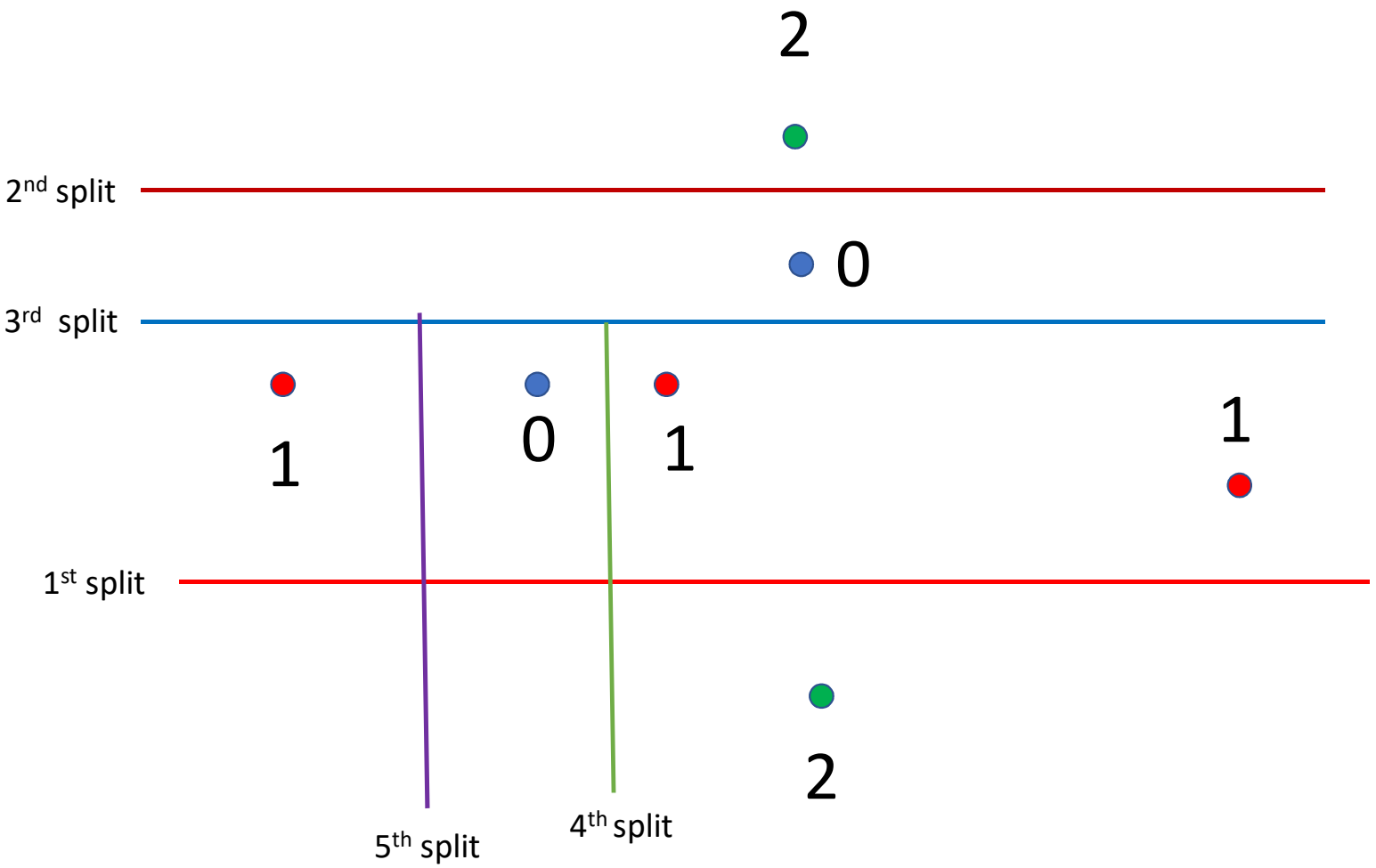
```
1 t = optimal_split(data,labels)
```

```
1 t
```

```
[((1, 0.5), 0.5238095238095238),
 ((1, 3.5), 0.5238095238095238),
 ((1, 0.5), 0.5238095238095238),
 ((1, 0.5), 0.5238095238095238),
 ((1, 0.0), 0.5238095238095238),
```

Thus we could split on $x_2 = 0.0$

This gives child nodes $L = \{((3, -1), 2)\}$ and
$R = \{((3,3),0),((6,1),1),((2,2),1),((1,2),0),((-1,2),1),((3,4),2)$


2
2nd split
0
3rd split
1
0
1
1
1st split
2
5th split
4th split

```
def compute_nodes(s,data,labels):
    L = [ i for i in range(len(data)) if data[i][s[0]] <= s[1]]
    R = [ i for i in range(len(data)) if data[i][s[0]] > s[1]]

    L = [data[i] for i in L],[labels[i] for i in L]
    R = [data[i] for i in R],[labels[i] for i in R]

    return L,R
```

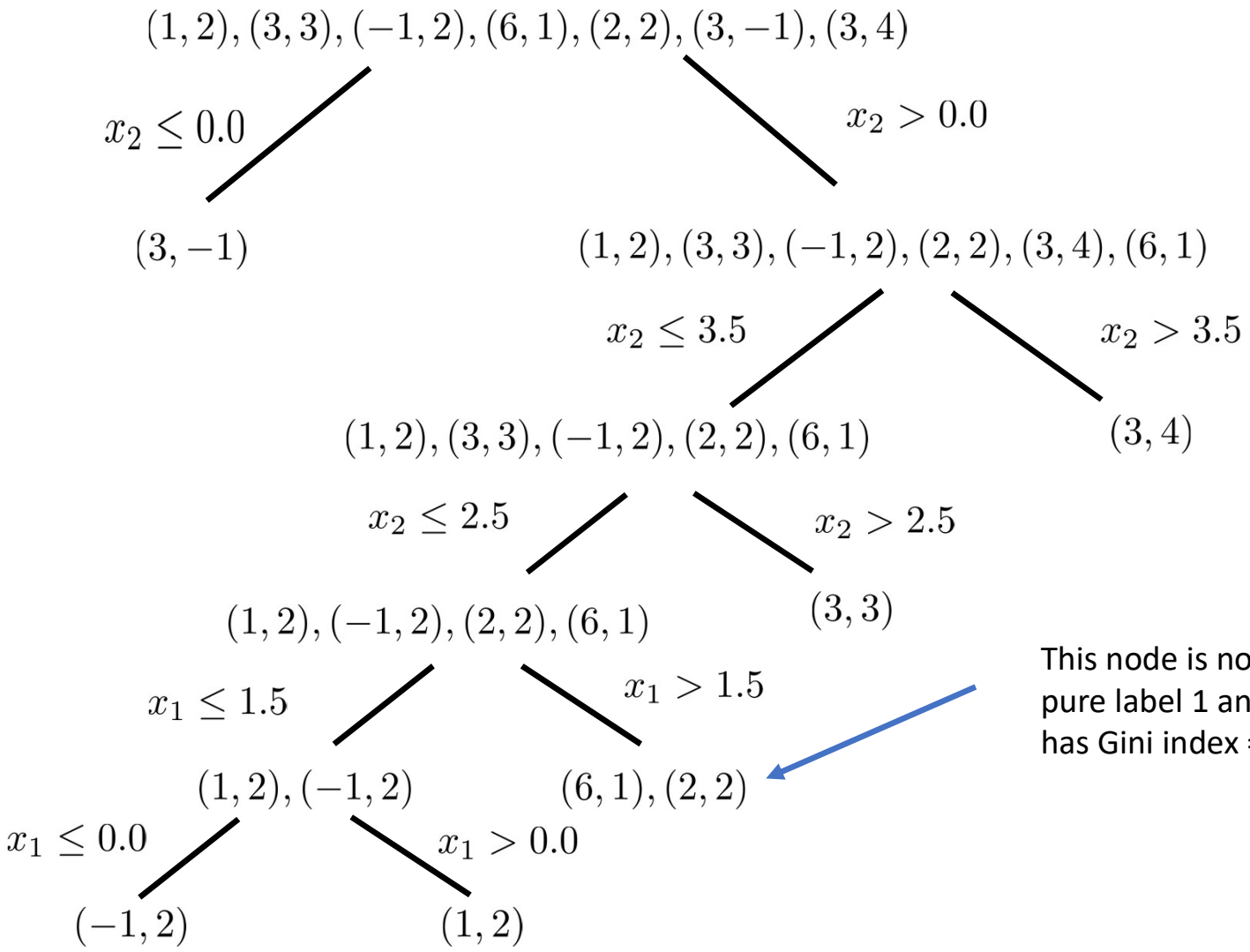
```
L,R = compute_nodes((1,0.5),data,labels)
```

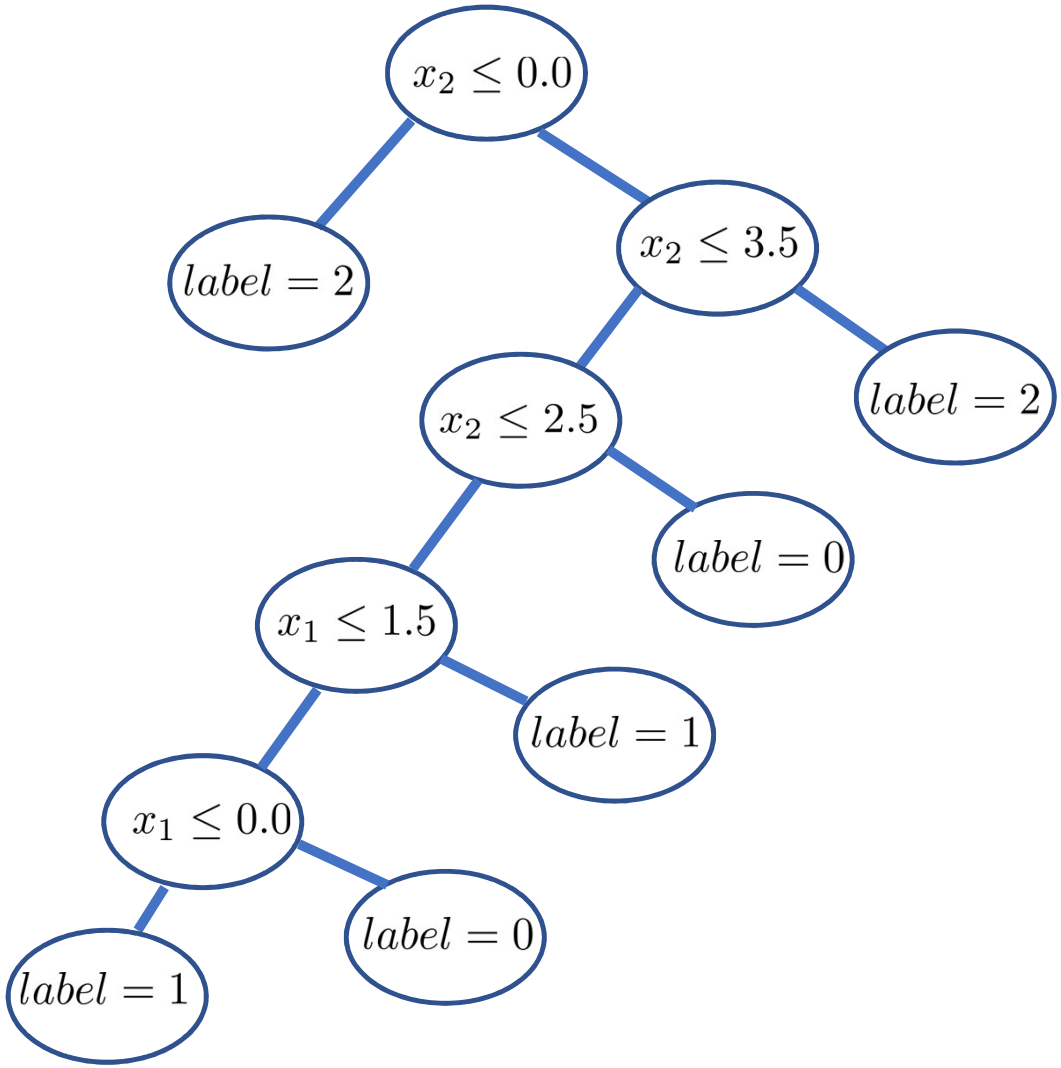
```
L
```

```
([array([ 3, -1])], [2])
```

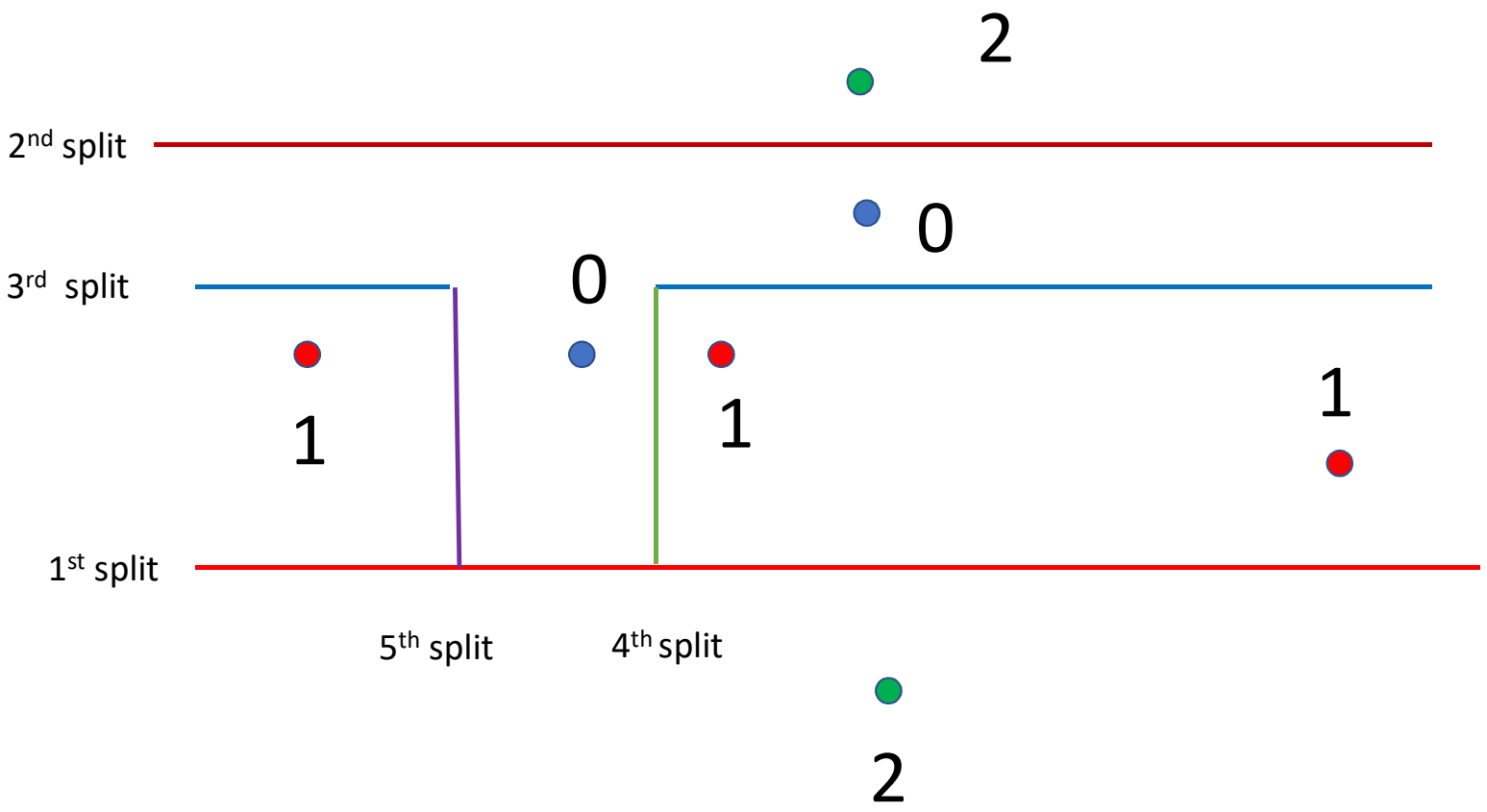
```
R
```

```
([array([1, 2]),
  array([3, 3]),
  array([-1,  2]),
  array([6, 1]),
  array([2, 2]),
  array([3, 4])],
 [0, 0, 1, 1, 1, 2])
```


$(1,2),(3,3),(-1,2),(6,1),(2,2),(3,-1),(3,4)$
$x_2 \leq 0.0$
$x_2 > 0.0$
$(3,-1)$
$(1,2),(3,3),(-1,2),(2,2),(3,4),(6,1)$
$x_2 \leq 3.5$
$x_2 > 3.5$
$(1,2),(3,3),(-1,2),(2,2),(6,1)$
$(3,4)$
$x_2 \leq 2.5$
$x_2 > 2.5$
$(1,2),(-1,2),(2,2),(6,1)$
$(3,3)$
$x_1 \leq 1.5$
$x_1 > 1.5$
$(1,2),(-1,2)$
$(6,1),(2,2)$
$x_1 \leq 0.0$
$x_1 > 0.0$
$(-1,2)$
$(1,2)$
This node is now pure label 1 and so has Gini index = 0


$x_2 \leq 0.0$
$label = 2$
$x_2 \leq 3.5$
$x_2 \leq 2.5$
$label = 2$
$label = 0$
$x_1 \leq 1.5$
$label = 1$
$x_1 \leq 0.0$
$label = 0$
$label = 1$

This corresponds to the partition of the plane



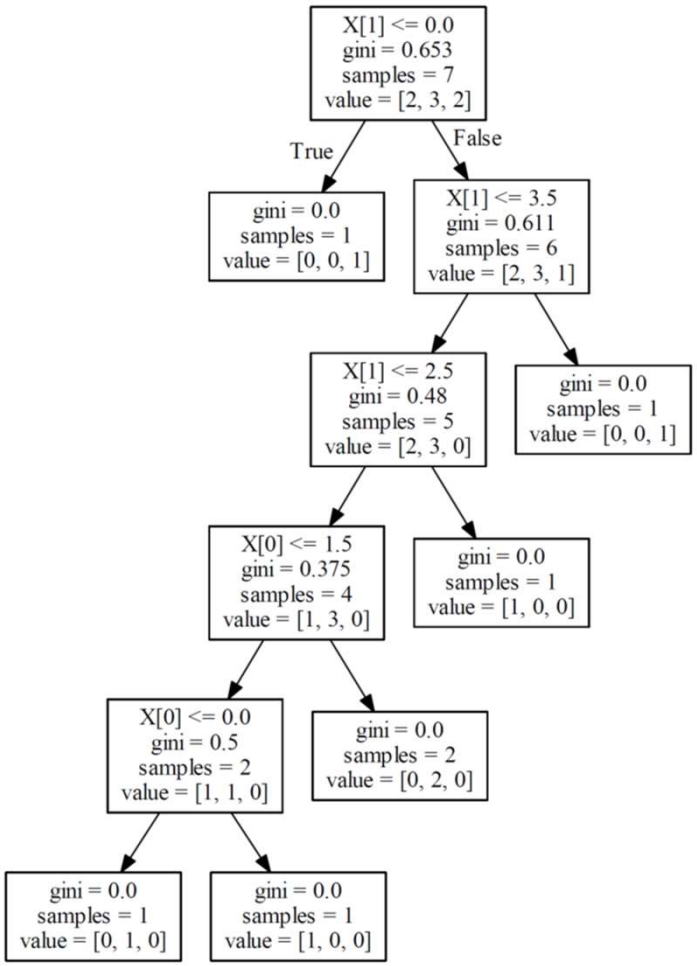
If we use the sklearn.tree. DecisionTree class we get a different tree but with the same partition of the plane so the classifier is the same

```
[3]: from sklearn.tree import DecisionTreeClassifier

     T = DecisionTreeClassifier()

     T.fit(data,labels)
```

```
t[3]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                             max_features=None, max_leaf_nodes=None,
                             min_impurity_decrease=0.0, min_impurity_split=None,
                             min_samples_leaf=1, min_samples_split=2,
                             min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                             splitter='best')
```

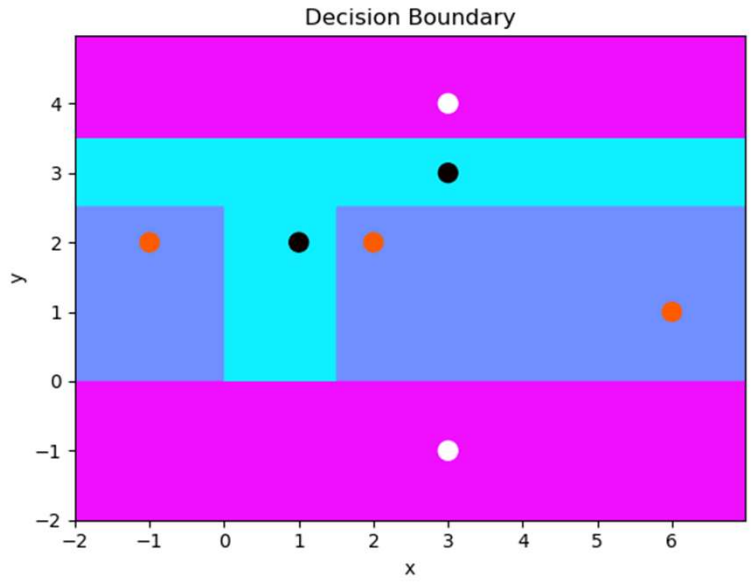
Using the software Graphviz we can get a picture of the tree

```
In [6]: import graphviz

In [7]: from sklearn import tree

In [8]: dot_data = tree.export_graphviz(T,out_file=None)
        graph = graphviz.Source(dot_data)
        graph.render('Tree1')

Out[8]: 'Tree1.pdf'
```


X[1] <= 0.0
gini = 0.653
samples = 7
value = [2, 3, 2]
True
False
gini = 0.0
samples = 1
value = [0, 0, 1]
X[1] <= 3.5
gini = 0.611
samples = 6
value = [2, 3, 1]
X[1] <= 2.5
gini = 0.48
samples = 5
value = [2, 3, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]
X[0] <= 1.5
gini = 0.375
samples = 4
value = [1, 3, 0]
gini = 0.0
samples = 1
value = [1, 0, 0]
X[0] <= 0.0
gini = 0.5
samples = 2
value = [1, 1, 0]
gini = 0.0
samples = 2
value = [0, 2, 0]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 1
value = [1, 0, 0]

In [53]:

```
import matplotlib.pyplot as plt
data = np.array(data)
plot_step = 0.02

#plt.subplot(121)
x_min, x_max = data[:, 0].min() - 1, data[:, 0].max() + 1
y_min, y_max = data[:, 1].min() - 1, data[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                     np.arange(y_min, y_max, plot_step))

Z = T.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
cs = plt.contourf(xx, yy, Z, cmap='cool')
plt.axis("tight")

plt.xlabel('x')
plt.ylabel('y')
plt.title('Decision Boundary')


plt.scatter(data[:,0],data[:,1],c=labels,cmap='hot',s=100)

plt.show()
```

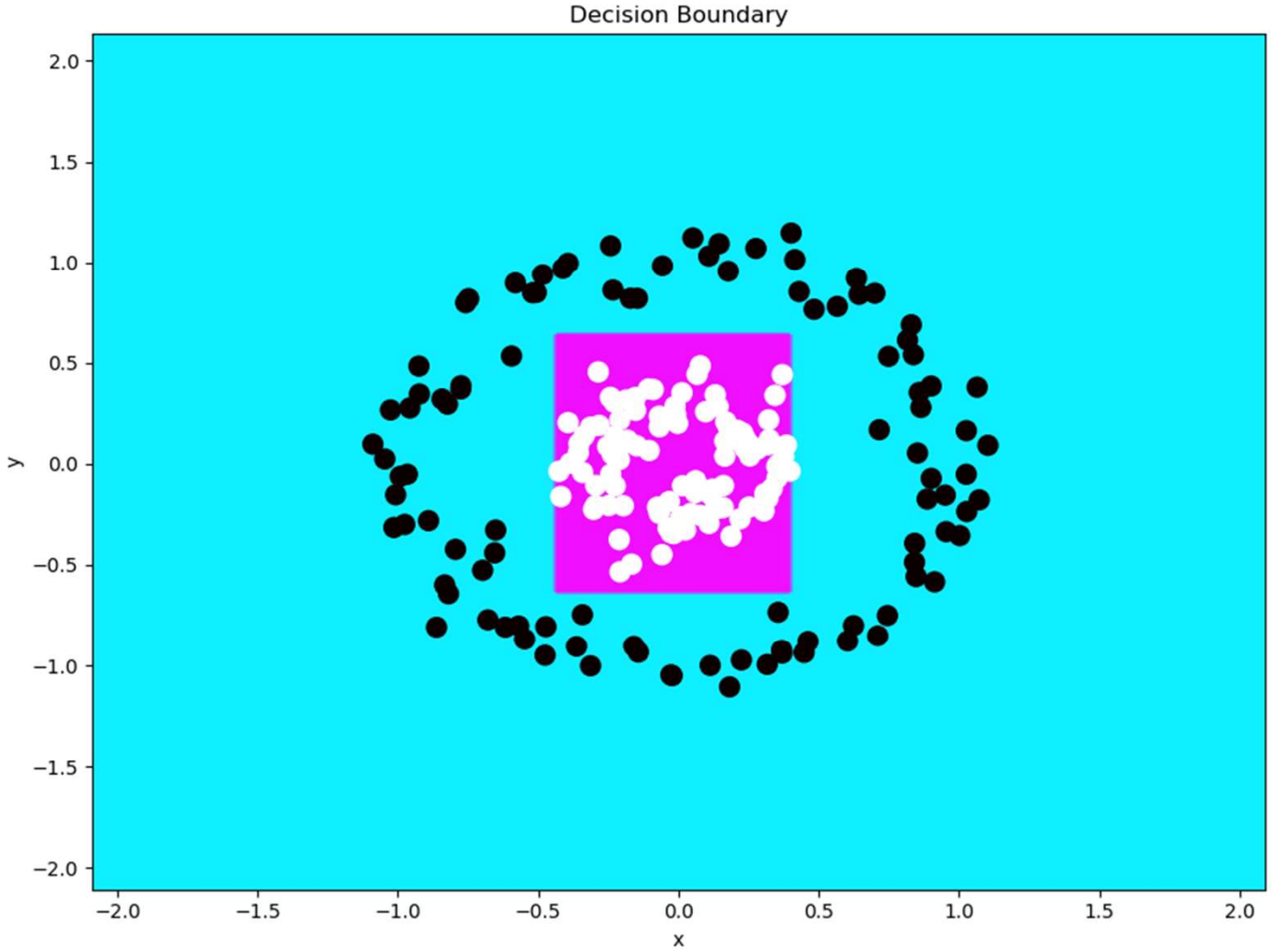


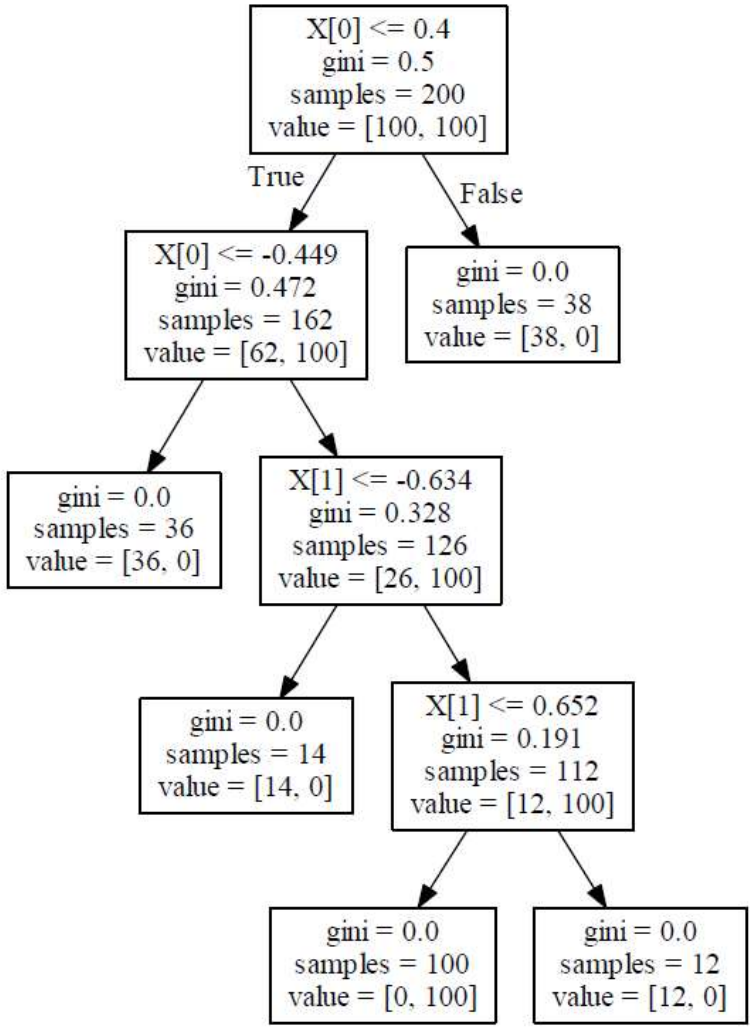
We can try the DecisionTreeClassifier on a more complicated dataset,

```
54]: from sklearn.datasets import make_circles

     training_set,labels = make_circles(200, noise=0.1,factor=0.3)

     training_labels = 2 * labels - 1
```

```
55]: T.fit(training_set,training_labels)
```

```
55]: DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
                            max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
                            splitter='best')
```


Decision Boundary
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x


X[0] <= 0.4
gini = 0.5
samples = 200
value = [100, 100]
True
False
X[0] <= -0.449
gini = 0.472
samples = 162
value = [62, 100]
gini = 0.0
samples = 38
value = [38, 0]
gini = 0.0
samples = 36
value = [36, 0]
X[1] <= -0.634
gini = 0.328
samples = 126
value = [26, 100]
gini = 0.0
samples = 14
value = [14, 0]
X[1] <= 0.652
gini = 0.191
samples = 112
value = [12, 100]
gini = 0.0
samples = 100
value = [0, 100]
gini = 0.0
samples = 12
value = [12, 0]

When we have larger data sets we would not carry the process all the way through to get terminal nodes with just one data point. To get a decision tree we associate to a terminal node with several data points, the majority label among the data points belonging to the node.

With many data points it is quite easy to get trees that are enormous and too big to fit in memory so normally we either limit the size of the tree or we use a technique called pruning.

We now want to apply a Decision Tree Classifier to construct the strategy.

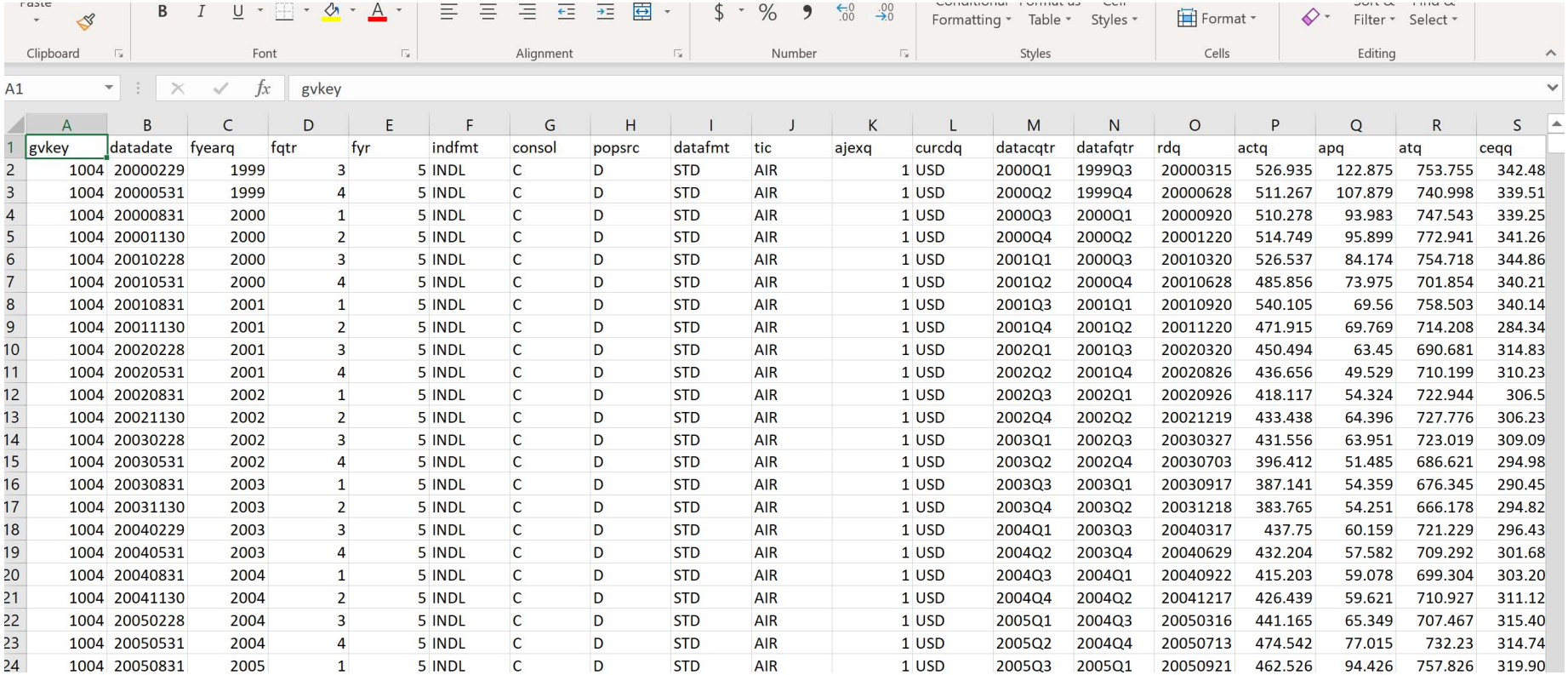
Most of the work is in constructing the data set.

We will use financial and accounting data from earnings releases and use this data to construct a number of financial ratios.

The raw accounting data are downloaded from the WRDS database.

Unfortunately the students in the program do not have access to this database but similar data can be found on the Quandl database which students do have access to

The downloaded data come in a csv file which we can visualize with Excel. The data are from 2000-01-01 t0 2018-12-31 and includes data from over 20,000 companies



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	gvkey	datadate	fyearq	fqtr	fyr	indfmt	consol	popsrc	datafmt	tic	ajexq	curcdq	datacqtr	datafqtr	rdq	actq	apq	atq	ceqq
2	1004	20000229	1999	3	5	INDL	C	D	STD	AIR	1	USD	2000Q1	1999Q3	20000315	526.935	122.875	753.755	342.48
3	1004	20000531	1999	4	5	INDL	C	D	STD	AIR	1	USD	2000Q2	1999Q4	20000628	511.267	107.879	740.998	339.51
4	1004	20000831	2000	1	5	INDL	C	D	STD	AIR	1	USD	2000Q3	2000Q1	20000920	510.278	93.983	747.543	339.25
5	1004	20001130	2000	2	5	INDL	C	D	STD	AIR	1	USD	2000Q4	2000Q2	20001220	514.749	95.899	772.941	341.26
6	1004	20010228	2000	3	5	INDL	C	D	STD	AIR	1	USD	2001Q1	2000Q3	20010320	526.537	84.174	754.718	344.86
7	1004	20010531	2000	4	5	INDL	C	D	STD	AIR	1	USD	2001Q2	2000Q4	20010628	485.856	73.975	701.854	340.21
8	1004	20010831	2001	1	5	INDL	C	D	STD	AIR	1	USD	2001Q3	2001Q1	20010920	540.105	69.56	758.503	340.14
9	1004	20011130	2001	2	5	INDL	C	D	STD	AIR	1	USD	2001Q4	2001Q2	20011220	471.915	69.769	714.208	284.34
10	1004	20020228	2001	3	5	INDL	C	D	STD	AIR	1	USD	2002Q1	2001Q3	20020320	450.494	63.45	690.681	314.83
11	1004	20020531	2001	4	5	INDL	C	D	STD	AIR	1	USD	2002Q2	2001Q4	20020826	436.656	49.529	710.199	310.23
12	1004	20020831	2002	1	5	INDL	C	D	STD	AIR	1	USD	2002Q3	2002Q1	20020926	418.117	54.324	722.944	306.5
13	1004	20021130	2002	2	5	INDL	C	D	STD	AIR	1	USD	2002Q4	2002Q2	20021219	433.438	64.396	727.776	306.23
14	1004	20030228	2002	3	5	INDL	C	D	STD	AIR	1	USD	2003Q1	2002Q3	20030327	431.556	63.951	723.019	309.09
15	1004	20030531	2002	4	5	INDL	C	D	STD	AIR	1	USD	2003Q2	2002Q4	20030703	396.412	51.485	686.621	294.98
16	1004	20030831	2003	1	5	INDL	C	D	STD	AIR	1	USD	2003Q3	2003Q1	20030917	387.141	54.359	676.345	290.45
17	1004	20031130	2003	2	5	INDL	C	D	STD	AIR	1	USD	2003Q4	2003Q2	20031218	383.765	54.251	666.178	294.82
18	1004	20040229	2003	3	5	INDL	C	D	STD	AIR	1	USD	2004Q1	2003Q3	20040317	437.75	60.159	721.229	296.43
19	1004	20040531	2003	4	5	INDL	C	D	STD	AIR	1	USD	2004Q2	2003Q4	20040629	432.204	57.582	709.292	301.68
20	1004	20040831	2004	1	5	INDL	C	D	STD	AIR	1	USD	2004Q3	2004Q1	20040922	415.203	59.078	699.304	303.20
21	1004	20041130	2004	2	5	INDL	C	D	STD	AIR	1	USD	2004Q4	2004Q2	20041217	426.439	59.621	710.927	311.12
22	1004	20050228	2004	3	5	INDL	C	D	STD	AIR	1	USD	2005Q1	2004Q3	20050316	441.165	65.349	707.467	315.40
23	1004	20050531	2004	4	5	INDL	C	D	STD	AIR	1	USD	2005Q2	2004Q4	20050713	474.542	77.015	732.23	314.74
24	1004	20050831	2005	1	5	INDL	C	D	STD	AIR	1	USD	2005Q3	2005Q1	20050921	462.526	94.426	757.826	319.90

There is a key to the abbreviations in the course material on Canvas

actq	Current Assets - Total
ajexq	Adjustment Factor (Company) - Cumulative by Ex-Date
atq	Assets - Total
capxy	Capital Expenditures
cdvcy	Cash Dividends on Common Stock (Cash Flow)
ceqq	Common/Ordinary Equity - Total
chechy	Cash and Cash Equivalents - Increase (Decrease)
cheq	Cash and Short-Term Investments
cogsq	Cost of Goods Sold
cogsy	Cost of Goods Sold
csh12q	Common Shares Used to Calculate Earnings Per Share - 12 Months Moving
cshfdq	Com Shares for Diluted EPS
cshiq	Common Shares Issued
cshopq	Total Shares Repurchased - Quarter
cshoq	Common Shares Outstanding
cshprq	Common Shares Used to Calculate Earnings Per Share - Basic
cstkq	Common/Ordinary Stock (Capital)
curcdq	Currency
datacqtr	Calendar Data Year and Quarter
datafqtr	Fiscal Data Year and Quarter

We can then download the data to a Jupyter Notebook using pandas

```
import pandas as pd
import numpy as np
```

```
qu = pd.read_csv(r'C:\Users\niels\data\CRSP-Daily\c87ebd90a260cdee_csv\c87ebd90a260cdee.csv',\
                 parse_dates=['rdq','datadate','ipodate'],low_memory=False)
```

```
qu = qu.reset_index()
qu = qu[qu['tic'].notnull()]
```

```
qu.reset_index(inplace=True)
```

```
qu = qu[qu['rdq'].notnull()]
```

We only keep the data that has a valid ticker symbol and valid earnings release dates

We get pandas to parse the dates into datetime objects

For instance for AAPL we get the dataframe

```
qu[qu['tic'] == 'AAPL']
```

:

	level_0	index	gvkey	datadate	fyearq	fqtr	fyr	indfmt	consol	popsrc	...	xinty	xrdy	xsgay	costat	dvpspq	prccq
7843	7843	7843	1690	2000-03-31	2000	2	9	INDL	C	D	...	10.0	182.0	788.0	A	0.00	135.8125
7844	7844	7844	1690	2000-06-30	2000	3	9	INDL	C	D	...	15.0	279.0	1163.0	A	0.00	52.3750
7845	7845	7845	1690	2000-09-30	2000	4	9	INDL	C	D	...	21.0	380.0	1546.0	A	0.00	25.7500
7846	7846	7846	1690	2000-12-31	2001	1	9	INDL	C	D	...	5.0	102.0	399.0	A	0.00	14.8750
7847	7847	7847	1690	2001-03-31	2001	2	9	INDL	C	D	...	10.0	203.0	792.0	A	0.00	22.0700
7848	7848	7848	1690	2001-06-30	2001	3	9	INDL	C	D	...	14.0	325.0	1184.0	A	0.00	23.2500
7849	7849	7849	1690	2001-09-30	2001	4	9	INDL	C	D	...	16.0	441.0	1568.0	A	0.00	15.5100

We can now construct a large number of quarterly ratios

Valuation Ratios

Book/Market

```
qu['beq'] = qu['seqq'] + qu['txditcq'] - qu['pstkq']
```

```
qu['bmq'] = qu['beq']/(qu['prccq'] * qu['cshoq'])
```

Book Value per Share(diluted)

```
qu['book_value_per_share'] = qu['beq']/qu['cshfdq']
```

Book Value Yield

```
qu['book_value_yield'] = qu['book_value_per_share']/qu['prccq']
```

Market Cap

```
qu['market_cap'] = qu['prccq']*qu['cshoq']
```

```
qu = qu[qu['market_cap'].notnull()]
```

Buy Back Yield

We save the finished dataframe (which now contains both the accounting data and the ratios)

```
qu.to_csv(r'C:\Users\niels\data\Quarterly-Ratios.csv')
```

rect_turnq	pay_turnq	sale_invcapq	sale_equityq	sale_nwcq	rd_saleq	at5	accrualq
11.737527	0.000000	3.803403	5.809473	5.681407	0.000000	753.75500	-5.288
6.679805	0.000000	1.907461	2.917374	2.851905	0.000000	747.37650	-3.375
8.246424	0.000000	2.237802	3.424255	3.363654	0.000000	747.43200	-14.277
6.339430	0.000000	1.677920	2.566594	2.561861	0.000000	753.80925	-0.662
6.791055	2.043102	1.676432	2.562113	2.546039	0.000000	753.99100	-8.997
6.969174	4.135103	1.676241	2.560805	2.522143	0.000000	743.61080	27.562
5.254595	6.495303	1.218541	1.869672	1.809103	0.000000	747.11180	11.847

Next we bring in the pricing data. We use the CRSP database to get daily prices and dividend and split data for the stocks in our universe.

	PERMNO	date	SICCD	TICKER	TRDSTAT	CUSIP	DCLRDT	PAYDT	RCRDDT	DIVAMT	BIDLO	ASKHI	PRC	VOL	RET	CFACP
0	10001	2000-01-03	4920	EWST	A	36720410	NaT	NaT	NaT	NaN	8.4375	8.5625	8.56250	1721.0	0.007353	1
1	10001	2000-01-04	4920	EWST	A	36720410	NaT	NaT	NaT	NaN	8.4375	8.4375	8.43750	1080.0	-0.014599	1
2	10001	2000-01-05	4920	EWST	A	36720410	NaT	NaT	NaT	NaN	8.4375	8.5625	8.56250	1711.0	0.014815	1
3	10001	2000-01-06	4920	EWST	A	36720410	NaT	NaT	NaT	NaN	8.5000	8.5000	8.50000	580.0	-0.007299	1
4	10001	2000-01-07	4920	EWST	A	36720410	NaT	NaT	NaT	NaN	8.4375	8.5625	8.43750	1406.0	-0.007353	1
5	10001	2000-01-10	4920	EWST	A	36720410	NaT	NaT	NaT	NaN	8.4375	8.5000	8.43750	3390.0	0.000000	1
6	10001	2000-01-11	4920	EWST	A	36720410	NaT	NaT	NaT	NaN	8.4375	8.5000	-8.46875	0.0	0.003704	1
7	10001	2000-01-12	4920	EWST	A	36720410	NaT	NaT	NaT	NaN	7.0000	8.4375	8.37500	14025.0	-0.011070	1
8	10001	2000-01-13	4920	EWST	A	36720410	NaT	NaT	NaT	NaN	8.1250	8.1250	8.12500	1700.0	-0.029851	1
9	10001	2000-01-14	4920	EWST	A	36720410	NaT	NaT	NaT	NaN	8.3750	8.3750	8.37500	850.0	0.030769	1

We rename the column headers to the same format as our fundamental data frame

We drop a column and we fill in any missing data with 0's

We reorder the way the columns are displayed

```
data.rename(columns={'TICKER':'ticker','DIVAMT':'dividend','SICCD':'sic','RCRDDT':'ex_dividend',
                     'CFACPR':'split','OPENPRC':'open','ASKHI':'high','BIDLO':'low',
                     'PRC':'close','VOL':'volume','CUSIP':'cusip','TRDSTST':'trading_status','PAYDT':'payment_date',
                     'RET':'return'},inplace=True)
```

```
data.drop('PERMNO',axis=1,inplace=True)
```

```
data['dividend'].fillna(0.0,inplace=True)
data['open'].fillna(0.0,inplace=True)
data['high'].fillna(0.0,inplace=True)
data['low'].fillna(0.0,inplace=True)
data['close'].fillna(0.0,inplace=True)
data['split'].fillna(1.0,inplace=True)
data['volume'] = data['volume'].fillna(0).map(lambda x: int(x)).values
data['close'] = np.abs(data.close.values)
```

```
data = data.reindex(columns=['date', 'ticker', 'cusip','open','high', 'low','close','volume','return','dividend','d
```

The data frame now looks like this

	date	ticker	cusip	open	high	low	close	volume	return	dividend	declaration_date	ex
0	2000-01-03	EWST	36720410	8.4375	8.5625	8.4375	8.56250	1721	0.007353	0.0	NaT	
1	2000-01-04	EWST	36720410	8.4375	8.4375	8.4375	8.43750	1080	-0.014599	0.0	NaT	
2	2000-01-05	EWST	36720410	8.4375	8.5625	8.4375	8.56250	1711	0.014815	0.0	NaT	
3	2000-01-06	EWST	36720410	8.5000	8.5000	8.5000	8.50000	580	-0.007299	0.0	NaT	
4	2000-01-07	EWST	36720410	8.5625	8.5625	8.4375	8.43750	1406	-0.007353	0.0	NaT	
5	2000-01-10	EWST	36720410	8.5000	8.5000	8.4375	8.43750	3390	0.000000	0.0	NaT	
6	2000-01-11	EWST	36720410	0.0000	8.5000	8.4375	8.46875	0	0.003704	0.0	NaT	
7	2000-01-12	EWST	36720410	8.4375	8.4375	7.0000	8.37500	14025	-0.011070	0.0	NaT	
8	2000-01-13	EWST	36720410	8.1250	8.1250	8.1250	8.12500	1700	-0.029851	0.0	NaT	
9	2000-01-14	EWST	36720410	8.3750	8.3750	8.3750	8.37500	850	0.030769	0.0	NaT	

The *return* column contains the daily return (from the previous day) adjusted for dividends and splits. The dividend adjustment is made on the ex-dividend date

We set date and ticker as a multi-index and sort the data frame by the index

```
data = data.set_index(['date','ticker'])
```

```
data.sort_index(inplace=True)
```

Remark the i*nplace* keyword here, without it we would be making a new copy of the frame. Since it is about 3 GB this would not be a good idea

date	ticker	cusip	open	high	low	close	volume	return	dividend	declaration_date
2000-01-03	A	00846U10	78.75000	78.9375	67.37500	72.00000	3343299	-0.068715	0.0	NaT
	AA	03965L10	82.00000	83.5625	80.37500	80.93750	1551299	-0.024849	0.0	NaT
	AAABB	00723110	8.43750	8.4375	7.87500	8.43750	3350	-0.007353	0.0	NaT
	AABC	00431F10	0.00000	7.9375	7.68750	7.81250	0	-0.015748	0.0	NaT
	AAC	03910110	4.43750	4.6250	4.43750	4.56250	121400	0.028169	0.0	NaT
	AACB	01853E20	0.00000	8.0000	7.81250	7.90625	0	-0.041667	0.0	NaT
	AACE	00440310	18.00000	18.1250	16.62500	17.00000	43934	-0.081081	0.0	NaT
	AAG	38991510	17.93750	17.9375	17.50000	17.50000	2000	-0.027778	0.0	NaT
	AAGP	30035510	2.39063	2.4375	2.28125	2.34375	13475	0.013514	0.0	NaT
	AAII	00252W10	8.62500	8.6250	7.75000	7.87500	9990	-0.136986	0.0	NaT

We can look at the data for AAPL in the quarter 2017/01/01 – 2017/04/01

```
idx = pd.IndexSlice
data.loc[(idx['2017-01-01':'2017-04-01'],'AAPL'),:]
```

2017-02-03	AAPL	03783310	128.31000	129.19000	128.16000	129.08000	24707301	0.004279	0.00	NaT	NaT	NaT
2017-02-06	AAPL	03783310	129.13000	130.50000	128.89999	130.28999	27095924	0.009374	0.00	NaT	NaT	NaT
2017-02-07	AAPL	03783310	130.53999	132.09000	130.45000	131.53000	38183916	0.009517	0.00	NaT	NaT	NaT
2017-02-08	AAPL	03783310	131.35001	132.22000	131.22000	132.03999	23863263	0.003877	0.00	NaT	NaT	NaT
2017-02-09	AAPL	03783310	131.64999	132.44501	131.12000	132.12000	28340859	0.007185	0.57	2017-01-31	2017-02-13	2017-02-16
2017-02-10	AAPL	03783310	132.46001	132.94000	132.05000	132.12000	20066343	-0.002266	0.00	NaT	NaT	NaT
2017-02-13	AAPL	03783310	133.08000	133.82001	132.75000	133.28999	23035671	0.008856	0.00	NaT	NaT	NaT
2017-02-14	AAPL	03783310	133.47000	135.09000	133.25000	135.02000	33311248	0.012979	0.00	NaT	NaT	NaT

Dividend

We reset the index and turn the *date* and *ticker* indices back to ordinary columns.

```
data.reset_index(inplace=True)
data = data[data['ticker'].notnull()]
data['return'] = pd.to_numeric(data['return'], errors='coerce')
```

We delete the rows that do not have a valid ticker symbol

The *return* column unfortunately has the wrong type so we turn it into numbers

```
data['return'] = data['return'].fillna(0.0)
data.set_index(['ticker','date'],inplace=True)
data.head()
```

Fix missing values (turn NaN into 0.0). Set the index to *ticker,date*

		cusip	open	high	low	close	volume	
ticker	date							
A	2000-01-03	00846U10	78.7500	78.9375	67.3750	72.0000	3343299	-0
	2000-01-04	00846U10	68.1250	68.8750	64.7500	65.5625	3408199	-0

We want to compute the cumulative returns for each ticker, this way the return between two dates is simply the difference between the cumulative returns at the two dates

```
data['cum_ret'] = \
data.groupby('ticker')['return'].transform(pd.Series.cumsum)
```

We first use the *groupby* command to divide the data frame into chunks, one for each ticker symbol. For each of these chunks we take the *return* column. This gives us pandas Series objects. Use the *transform* command and the pd.Series command *cumsum* to compute the series of cumulative sums. We make a new column in the data frame to hold these values

The *groupby* command does not return a new DataFrame or Series object and it can be somewhat complicated to use correctly. Applying a function using *transform* gives us back a Series object

Next we want to get the returns on the index. We use the tradable ETF with ticker symbol SPY, which tracks the SP500 index as a proxy for the actual index.

There are tradable ETFs that track many factors, like the Fama-French factors for instance

We first get the cumulative returns for SPY. This is a pandas Series object. We turn the Series into a DataFrame

```
SPY_cum_ret = data.loc['SPY']['cum_ret']
```

```
SPY_cum_ret = SPY_cum_ret.to_frame('spy_cum_ret')
```

We re-index the data DataFrame with *date (*this makes *ticker* an ordinary column in data)

```
data.reset_index(inplace=True)
data.set_index('date',inplace=True)
```

Pandas Series

```
date
2000-01-03   -0.009787
2000-01-04   -0.048893
2000-01-05   -0.047104
2000-01-06   -0.063175
2000-01-07   -0.005099
2000-01-10   -0.001668
2000-01-11   -0.014916
2000-01-12   -0.023578
2000-01-13   -0.010035
2000-01-14    0.003543
2000-01-18   -0.004324
2000-01-19   -0.001324
2000-01-20   -0.011580
2000-01-21   -0.013739
2000-01-24   -0.042082
2000-01-25   -0.030726
2000-01-26   -0.038652
2000-01-27   -0.042647
```

Pandas DataFrame

	spy_cum_ret
date	
2000-01-03	-0.009787
2000-01-04	-0.048893
2000-01-05	-0.047104
2000-01-06	-0.063175
2000-01-07	-0.005099
2000-01-10	-0.001668
2000-01-11	-0.014916
2000-01-12	-0.023578
2000-01-13	-0.010035
2000-01-14	0.003543
2000-01-18	-0.004324
2000-01-19	-0.001324
2000-01-20	-0.011580
2000-01-21	-0.013739
2000-01-24	-0.042082

Next we are going to merge the data and the SPY_cum_ret DataFrames along the date index. This will create a DataFrame with a new column which for each date has the cumulative return for SPY

```
data = data.merge(SPY_cum_ret,left_index=True,right_index=True)
```

date	ticker	cusip	open	high	low	close	volume	return	c
2000-01-03	A	00846U10	78.7500	78.9375	67.3750	72.0000	3343299	-0.068715	
2000-01-03	AA	03965L10	82.0000	83.5625	80.3750	80.9375	1551299	-0.024849	
2000-01-03	AAABB	00723110	8.4375	8.4375	7.8750	8.4375	3350	-0.007353	
2000-01-03	AABC	00431F10	0.0000	7.9375	7.6875	7.8125	0	-0.015748	
2000-	AAC	03910110	4.4375	4.6250	4.4375	4.5625	121400	0.028169	

ent_date	split	cum_ret	spy_cum_ret
NaT	1.464715	-0.068715	-0.009787
NaT	0.888889	-0.024849	-0.009787
NaT	1.050000	-0.007353	-0.009787
NaT	1.000000	-0.015748	-0.009787
NaT	1.000000	0.028169	-0.009787

A different view of the data DataFrame

ticker	date	cusip	open	high	low	close	volume	return	dividend	declaratio	ate	split	cum_ret	spy_cum_ret
A	2000-01-03	00846U10	78.7500	78.9375	67.3750	72.0000	3343299	-0.068715	0.0		aT	1.464715	-0.068715	-0.009787
	2000-01-04	00846U10	68.1250	68.8750	64.7500	65.5625	3408199	-0.089410	0.0		aT	1.464715	-0.158125	-0.048893
	2000-01-05	00846U10	66.2500	66.2500	60.3125	62.3750	4119199	-0.048618	0.0		aT	1.464715	-0.206743	-0.047104
	2000-01-06	00846U10	61.6250	62.0000	58.1250	59.0000	1812799	-0.054108	0.0		aT	1.464715	-0.260851	-0.063175
	2000-01-07	00846U10	59.0625	65.9375	59.0000	65.0000	2016899	0.101695	0.0		aT	1.464715	-0.159156	-0.005099

We drop the columns we don’t need

```
data.drop(['cusip',
           'open',
           'high',
           'low',
           'close',
           'volume',
           'dividend',
           'declaration_date',
           'ex_dividend',
           'payment_date',
           'split'],axis=1,inplace=True)
```

```
data.head()
```

		return	cum_ret	spy_cum_ret
ticker	date			
A	2000-01-03	-0.068715	-0.068715	-0.009787
	2000-01-04	-0.089410	-0.158125	-0.048893
	2000-01-05	-0.048618	-0.206743	-0.047104
	2000-01-06	-0.054108	-0.260851	-0.063175
	2000-01-07	0.101695	-0.159156	-0.005099

Finally we save it as a csv file

```
data.to_csv(r'C:\Users\niels\data\crsp_prices.csv')
```

If we want to preserve the indexing we can use pickle to serialize the DataFrame i.e. save it as a binary file with the structure encoded in the file

```
data.to_pickle(r'C:\Users\niels\data\crsp_prices.pkl')
```

Now we are ready to combine the dataframe of fundamentals with the dataframe of returns

```
import pandas as pd
import numpy as np
```

```
crsp_prices = pd.read_pickle(r'C:\Users\niels\data\crsp_prices.pkl')
```

```
crsp_prices.head(10)
```

4]:

ticker	date	return	cum_ret	spy_cum_ret
A	2000-01-03	-0.068715	-0.068715	-0.009787
	2000-01-04	-0.089410	-0.158125	-0.048893
	2000-01-05	-0.048618	-0.206743	-0.047104
	2000-01-06	-0.054108	-0.260851	-0.063175
	2000-01-07	0.101695	-0.159156	-0.005099
	2000-01-10	0.060577	-0.098579	-0.001668
	2000-01-11	-0.013599	-0.112178	-0.014916
	2000-01-12	-0.020221	-0.132399	-0.023578
	2000-01-13	0.015009	-0.117390	-0.010035
	2000-01-14	0.011091	-0.106299	0.003543

```
crsp_prices = crsp_prices.swaplevel()
```

```
crsp_prices.sort_index(inplace=True)
```

```
crsp_prices.head()
```

:

		return	cum_ret	spy_cum_ret
date	**ticker**			
2000-01-03	**A**	-0.068715	-0.068715	-0.009787
	AA	-0.024849	-0.024849	-0.009787
	AAABB	-0.007353	-0.007353	-0.009787
	AABC	-0.015748	-0.015748	-0.009787
	AAC	0.028169	0.028169	-0.009787

```
crsp_prices = crsp_prices[~crsp_prices.index.duplicated()]
```

Load the fundamentals DataFrame

```
ratios = pd.read_pickle(r'C:\Users\niels\data\Quarterly-Ratios.pkl')
```

Rename the rdq (=Earnings Release Date) and the tic columns so they vae the same names as in the crsp-prices

```
ratios.rename(columns={'rdq':'date','tic':'ticker'},inplace=True)
```

```
ratios.set_index(['date'],inplace=True)
```

To avoid any kind of look-ahead-bias we only trade on the day after the earnings release, so we add 1 business day to the date

```
from pandas.tseries.offsets import BDay
ratios.index = ratios.index.map(lambda x : x + BDay())
```

```
ratios.reset_index(inplace=True)
```

```
ratios.set_index(['date','ticker'],inplace=True)
```

```
ratios.sort_index(inplace=True)
```

We get rid of rows with invalid ticker symbols and we only keep US traded securities

```
ratios = ratios.loc[ratios.index.get_level_values(1).notnull()]
```

```
ratios = ratios[ratios['curcdq']=='USD']
```

```
merged = ratios.merge(crsp_prices,left_index=True,right_index=True)
```

We merge the two dataframes on the date and ticker

Companies that have released earnings on the previous day

date	ticker	gvkey	fyr	fyearq	fqtr	level_0	index	datadate	indfmt	consol	popsrc	...	pay_	alq	return	cum_ret	spy_cum_ret
2000-02-01	LLB	3354	1	1999	4	26246	26246	2000-01-31	INDL	C	D	...		laN	-0.055556	1.363473	-0.036842
2000-02-03	MYR	13866	1	1999	4	146033	146033	2000-01-31	INDL	C	D	...		laN	0.006356	0.010728	-0.020891
2000-02-08	LZB	6543	4	1999	3	61730	61730	2000-01-31	INDL	C	D	...		laN	0.057269	-0.104214	-0.012964
	SJM	9777	4	1999	3	97130	97130	2000-01-31	INDL	C	D	...		laN	0.003448	-0.112228	-0.012964
2000-02-09	CSCO	20779	7	2000	2	231584	231584	2000-01-31	INDL	C	D	...		laN	0.023845	0.200574	-0.033969
	ROP	24925	10	2000	1	275534	275534	2000-01-31	INDL	C	D	...		laN	-0.058201	-0.111988	-0.033969

The first business day after the earnings release, these are the dates we can trade

The cumulative returns on those dates

Set the index to *date*

```
merged.reset_index(inplace=True)
```

```
merged.set_index('date',inplace=True)
```

We compute the returns between the trading dates by grouping into tickers and then take successive differences

```
merged['next_period_return'] = \
merged.groupby('ticker')['cum_ret'].transform(pd.Series.diff)
```

```
merged['spy_next_period_return'] = \
merged.groupby('ticker')['spy_cum_ret'].transform(pd.Series.diff)
```

date	ticker	gvkey	fyr	fyearq	fqtr	return	cum_ret	spy_cum_ret	next_period_return	spy_next_period_return
2000-02-01	LLB	3354	1	1999	4	055556	1.363473	-0.036842	NaN	NaN
2000-02-03	MYR	13866	1	1999	4	006356	0.010728	-0.020891	NaN	NaN
2000-02-08	LZB	6543	4	1999	3	057269	-0.104214	-0.012964	NaN	NaN
2000-02-08	SJM	9777	4	1999	3	003448	-0.112228	-0.012964	NaN	NaN
2000-02-09	CSCO	20779	7	2000	2	023845	0.200574	-0.033969	NaN	NaN
2000-02-09	ROP	24925	10	2000	1	058201	-0.111988	-0.033969	NaN	NaN

The first entries in the successive differences are all NaN because we do not have the data from the previous dates. But we want to predict the return in the next earnings period so we want to shift these columns upwards

```
merged['next_period_return'] = merged.groupby('ticker')['next_period_return'].shift(-1)
```

```
merged['spy_next_period_return'] = merged.groupby('ticker')['spy_next_period_return'].shift(-1)
```

cum_ret	spy_cum_ret	next_period_return	spy_next_period_return
1.363473	-0.036842	-0.464979	0.055973
0.010728	-0.020891	0.165367	0.055698
-0.104214	-0.012964	0.112182	0.020457
-0.112228	-0.012964	0.153279	0.032095
0.200574	-0.033969	-0.037098	-0.011175
-0.111988	-0.033969	0.077255	0.011080

We set the index to date and ticker and drop the columns we don't need

```
merged.reset_index(inplace=True)
```

```
merged.set_index(['date','ticker'],inplace=True)
```

```
merged.sort_index(inplace=True)
```

```
merged.drop(['gvkey',
             'fyr',
             'fyearq',
             'level_0',
             'index',
             'indfmt',
             'consol',
             'datafmt',
             'popsrc',
             'costat',
             'ipodate',
             'datadate',
             'curcdq',
             'datacqtr',
             'datafqtr',
             'ajexq'
            ],axis=1,inplace=True)
```

```
merged.head()
```

]:

date	ticker	fqtr	actq	apq	atq	ceqq	cheq	cogsq	csh12q	cshfdq	cshiq	...	s
2000-02-01	LLB	4	3.540	0.143	7.668	6.732	2.553	0.458	6.3910	6.391	6.989	...	
2000-02-03	MYR	4	107.661	24.387	220.463	136.555	1.049	36.883	25.5360	22.082	28.458	...	
2000-02-08	LZB	3	447.719	57.893	740.905	460.612	16.531	274.525	52.2660	52.274	52.544	...	
	SJM	3	234.415	33.821	488.136	322.432	26.054	91.172	28.8808	28.603	32.425	...	
2000-02-09	CSCO	2	7722.000	482.000	21391.000	16523.000	3968.000	1422.000	3374.1250	3648.000	3445.000	...	

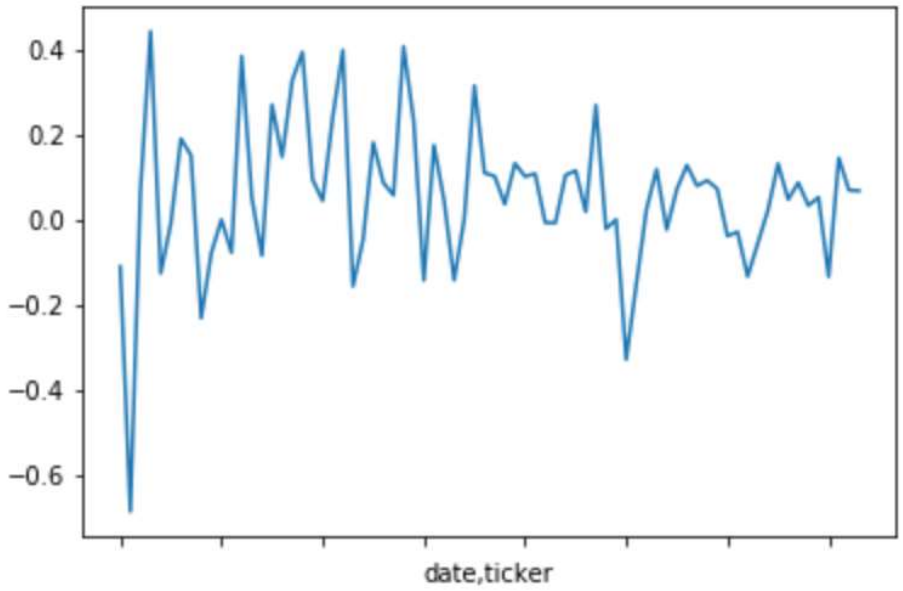
5 rows × 188 columns

We insert a column with the differences between the stock return and the index return

```
merged['pred_rel_return'] = merged['next_period_return'] - merged['spy_next_period_return']
```

ıext_period_return	spy_next_period_return	pred_rel_return
-0.464979	0.055973	-0.520952
0.165367	0.055698	0.109669
0.112182	0.020457	0.091725
0.153279	0.032095	0.121184
-0.037098	-0.011175	-0.025923
0.077255	0.011080	0.066175
0.287974	0.046629	0.241345
-0.334574	0.040832	-0.375406
-0.724923	0.059780	-0.784703
0.346867	0.040832	0.306035

```
import matplotlib.pyplot as plt
%matplotlib inline

merged.loc[(slice(None),'AAPL'),:]['pred_rel_return'].plot();
```



```
merged = pd.get_dummies(merged,prefix='fqtr',columns=['fqtr'])
```

```
merged['cshopq'].fillna(0.0,inplace=True)
```

```
merged = pd.get_dummies(merged,prefix='sic',columns=['sic'])
```

```
merged = pd.get_dummies(merged,columns=['spcsrc'])
```

```
merged = pd.get_dummies(merged,prefix='sector_code',columns=['spcindcd'])
```

The fqrt = fiscal quarter, sic = industry code, spcsrc = S&P grade, spcindcd = S&P industry code are all categorical values and we turn the into one-hot-encoded columns

Our date set is essentially finished so we save it as a csv file and a pickle file

```
[64]: merged.to_csv(r'C:\Users\niels\data\dataset.csv')
```

```
[65]: merged.to_pickle(r'C:\Users\niels\data\dataset.pkl')
```

Home Work

1. Insert a column in the data set where the entries are 1 if the stock outperforms SPY in the earnings period and -1 if it underperforms or has the same return
2. Insert a column in the data set with entries: 2 if the stock return is more then 5% higher than the SPY return, 1 if it is more than 1% but less than 5% higher, 0 if it is between -1% and 1%, -2 if the stock underperforms the SPY by more than -5% and -1 if the performance is between -1% and -5%

3. A regression tree is used when the labels are real numbers instead of categories. Think of a linear regression situation when we have data points $\{x_i\}$ and response variables $\{y_i\}$ that are real numbers.
   A regression tree uses the variance of the response variables instead of the Gini index. If n_j is a node with data $\{x_{ij}\}$ and $\{y_{ij}\}$ the variance of the response variables is

$$Var(\{y_{ij}\}) = \frac{1}{\#\{y_{ij}\}} \sum_i (y_{ij} - \overline{y}_{ij})^2$$

   where $\overline{y}_{ij}$ is the average of the $\{y_{ij}\}$

To split the node into n_{j1} and n_{j2} we look for the split that minimizes

$$\frac{\#n_{j1}}{\#n_j} Var(\{y_{ij1}\}) + \frac{\#n_{j2}}{\#n_j} Var(\{y_{ij2}\})$$

In the notebook “Visualizing Trees” use a DecisionTreeRegressor instead of the DecisionTreeClassifier, directly on the data1 and the target (so do not transform the target into labels).

Instead of taking max in each rectangle take the average and generate the image.

Experiment with different color schemes (cm.?)