Machine Learning

Lecture 9

Adaboost = Adaptive Boosting is an iterative algorithm to build up a classifier by successively adding simpler classifiers such as DecisionTrees or even RandomForestClassifiers

We are going to explain the principle behind Adaboosting by looking at binary classifiers but it can also be applied to multi-class problems.

A weak classifier is one whose accuracy rate is only slightly better than 50%. The idea of boosting is to sequentially apply weak classifiers G to modified versions of the data thereby producing a sequence of weak classifiers G_m . The predictions from all of them are then combined through a weighted sum

$$G(\underline{x}) = sign(\sum \beta_m G_m(\underline{x}))$$

If we were dealing with more than two labels we would use the softmax function rather than just the sign

The training data set is modified by applying weights to the data in such a way that data points that are misclassified by the classifier G_{m-1} receive a higher weight while data points that are correctly classified do not change weights.

For a weighted data set where the data point $\underline{x}_i$ with label y_i has weight w_i, the weighted error of the classifier G_m is defined by

$$err_m = \frac{\sum w_i I(y_i \neq G_m(\underline{x}_i))}{\sum w_i}$$

where $I(y_i \neq G_m(\underline{x})) = 0$ if $y_i = G_m(\underline{x}_i)$ i.e. if G_m correctly classifies $\underline{x}_i$ and 1 if G_m misclassifies it.

Given the weights we fit the weak classifier G_m to minimize the weighted error

The update process works by setting

$$\beta_m = \frac{1}{2} \log \frac{1 - err_m}{err_m}$$

and then updating

$$w_i \leftarrow w_i \exp(2\beta_m I(y_i \neq G_m(\underline{x}_i))$$

After M steps the AdaBoost classifier is given by

$$G = \sum_m^M \beta_m G_m$$

and it labels a point $\underline{x}$ as $sign(G(\underline{x}))$

To see how we get these weights consider a data set $\{\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_N\}$ and labels $y_{j_i} = \pm 1$. Let G be a classifier i.e. a function such that the predicted label for a data point $\underline{x}$ is $sign(G(\underline{x}))$. Define the *exponential loss function* by

$$\frac{1}{N}\sum_i \exp(-y_i G(\underline{x}_i))$$

Remark that if G correctly classifies $\underline{x}$ then $yG(\underline{x}) > 0$ so $\exp(-yG(\underline{x})) < 1$, while if G misclassifies $\underline{x}$, $yG(\underline{x}) < 0$ so $\exp(-yG(\underline{x})) > 1$.

We want to construct a classifier of the form

$$G = \sum \beta_k G_k$$

where each G_k is some simple classifier like a tree with just two brances (a tree stump).

We construct the G_j's step by step.

Assume we have constructed a classifier f and we want to update f to $G = f + \beta G_k$ where G_k is a tree stump i.e. $G_k(\underline{x}) = \pm 1$.

Using the exponential loss function we want to minimize

$$\mathcal{L}(\beta, G) = \frac{1}{N}\sum_i \exp(-y_{j_i}(f(\underline{x}_i) + \beta G_k(\underline{x}_i))$$

Let $w_i = \exp(-y_{j_i} f(\underline{x}_i))$ so $\mathcal{L}(\beta, G) = \dfrac{1}{N}\sum_i w_i \exp(-y_{j_i}\beta G_k(\underline{x}_i))$

Now $\exp(-y_{j_i}\beta G_k(\underline{x}_i)) = \exp(-\beta)$ if $y_{j_i} = G_k(\underline{x}_i)$ and $\exp(\beta)$ if $y_{j_i} \neq G_k(\underline{x}_i)$ so we can write

$$\begin{aligned}\mathcal{L}(\beta, G) &= \exp(-\beta)\sum_{G_k(\underline{x}_i)=y_{j_i}} w_i + \exp(\beta)\sum_{G_k(\underline{x}_i)\neq y_{j_i}} w_i \\ &= (\exp(\beta) - \exp(-\beta))\sum_{G_k(\underline{x}_i)\neq y_{j_i}} w_i + \exp(-\beta)\sum_i w_i\end{aligned}$$

Let

$$err_{G_k} = \frac{\sum\limits_{G_k(\underline{x}_i) \neq y_{j_i}} w_i}{\sum\limits_i w_i}$$

So we want to minimize

$$(\exp(\beta) - \exp(-\beta))err_{G_k} + \exp(-\beta) = \exp(\beta)err_{G_k} + \exp(-\beta)(1 - err_{G_k})$$

Minimizing with respect to β we differentiate and set the derivative $= 0$. This gives

$$\exp(\beta)err_{G_k} - \exp(-\beta)(1 - err_{G_k}) = 0$$

so $\beta = \dfrac{1}{2}\log\dfrac{1 - err_{G_k}}{err_{G_k}}$.

If we want $\beta > 0$ we have $(\exp(\beta) - \exp(-\beta)) > 0$ so we want to minimize err_{G_k} and so we choose G_k to minimize err_{G_k}, the weighted error. If $err_{G_k} < \dfrac{1}{2}$ we have $\beta = \dfrac{1}{2}\log\dfrac{1 - err_{G_k}}{err_{G_k}} > 0$

The updated classifier is then

$$G = f + \beta G_k$$

and the new weights

$$w_i^{(new)} = \exp(-y_{j_i}(f(\underline{x}_i) + \beta G_k(\underline{x}_i)) = w_i \exp(-y_{j_i}\beta G_k(\underline{x}_i))$$

Now $-y_{j_i}G_k(\underline{x}_i) = -1$ if $G_k(\underline{x}_i) = y_{j_i}$ i.e. if G_k correctly classifies $\underline{x}_i$. Otherwise it is $+1$.

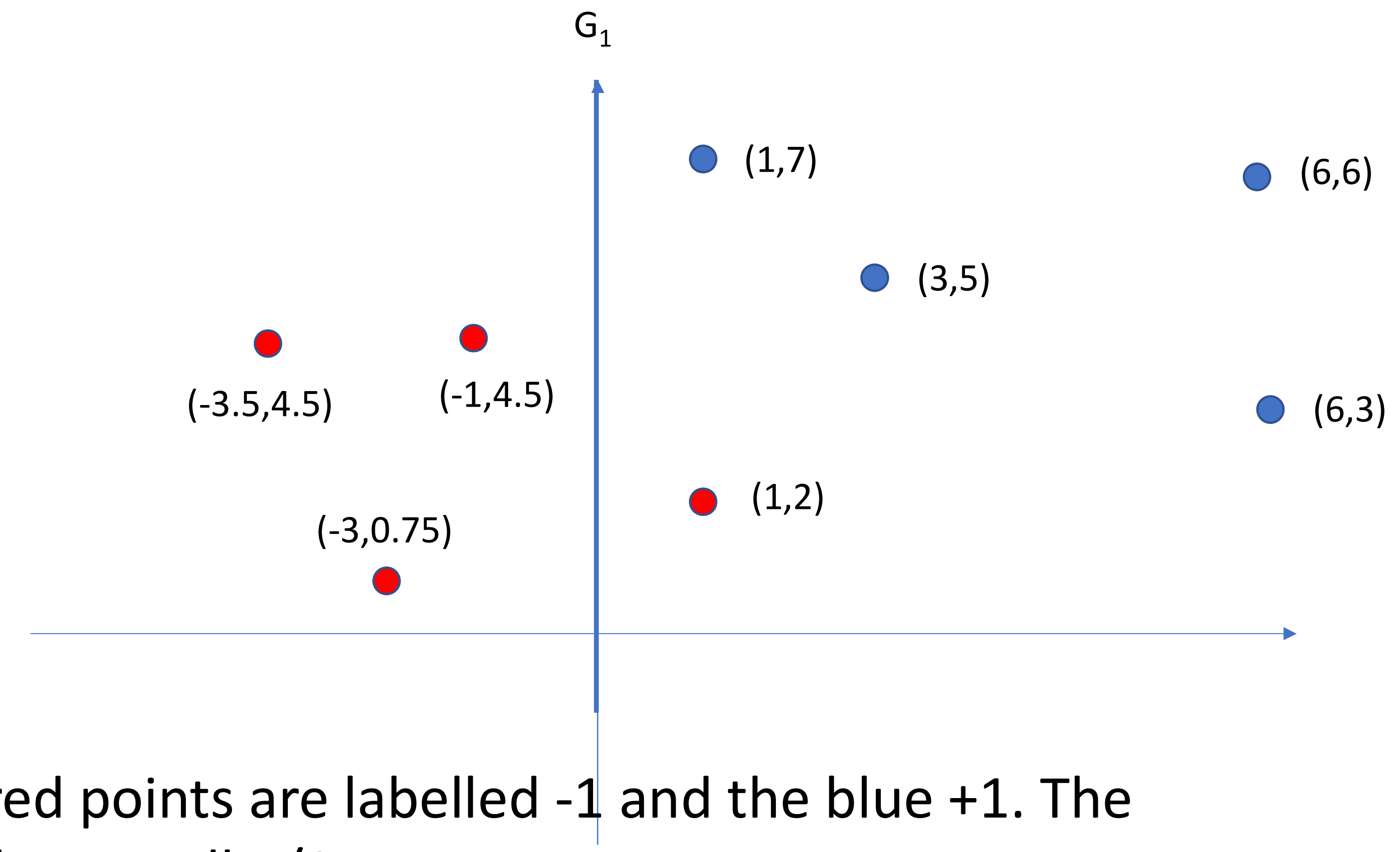
Thus

$$w_i^{(new)} = \begin{cases} w_i \exp(-\beta) \text{ if } G_k(\underline{x}_i) = y_{j_i} \\ w_i \exp(\beta) = w_i \exp(2\beta)\exp(-\beta) \text{ if } G_k(\underline{x}_i) \neq y_{j_i} \end{cases}$$

Scaling all the weights by the same factor has no effect on the weighted error so we have the update rule for the weights

$$w_i^{(new)} = \begin{cases} w_i \text{ if } G_k(\underline{x}_i) = y_{j_i} \\ w_i \exp(2\beta) = w_i \dfrac{1 - err_{G_k}}{err_{G_k}} \text{ if } G_k(\underline{x}_i) \neq y_{j_i} \end{cases}$$

We try a toy experiment

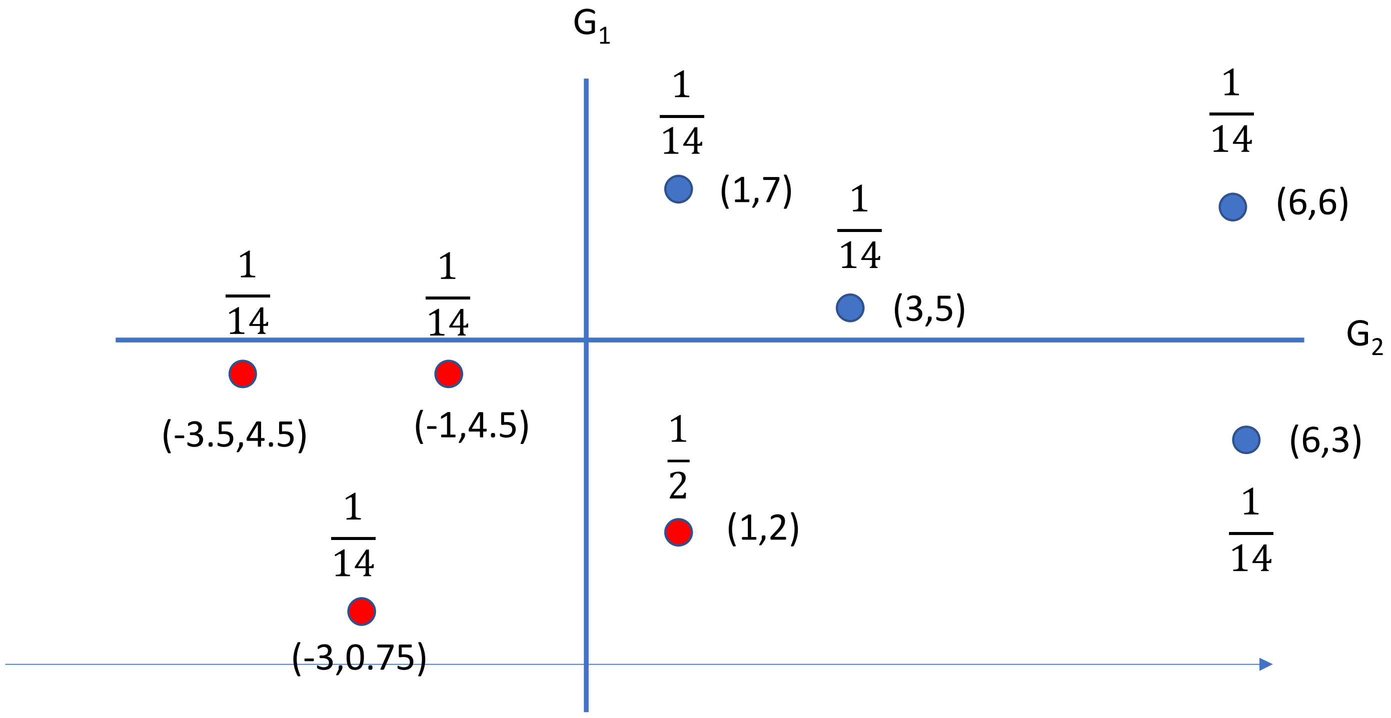


The red points are labelled -1 and the blue +1. The weights are all 1/8

Here the tree stump G_1 is the split at $x_1 = 0$. This tree misclassifies $(1,2)$ and so $err_{G_1} = \frac{1}{8}$. The β_1 coefficient is then $\frac{1}{2}\log\frac{1-err}{err} = \frac{1}{2}\log 7$ and $2\beta = \log 7$. So the 1'st classifier is $sign(\log(7)G_1)$ (since we are taking $sign$ it is harmless to get rid of the factor $\frac{1}{2}$). Now we change the weights. The point that is misclassified is $(1,2)$. The weight is updated to

$$w_1 = \frac{1}{8}\exp(2\beta_1) = \frac{1}{8}\frac{1-err}{err} = \frac{1}{8}7 = \frac{7}{8}$$

and the other weights are unchanged. To get the weights to sum to 1 we divide by the sum of the new weights $= \frac{7}{8} + 7 \times \frac{1}{8} = \frac{14}{8}$ and hence the new weights are $w_{(1,2)} = \frac{1}{2}$ and all the other weights are $w = \frac{1}{14}$


G_1
G_2
$\frac{1}{14}$
(1,7)
$\frac{1}{14}$
(6,6)
$\frac{1}{14}$
(3,5)
$\frac{1}{14}$
(-3.5,4.5)
$\frac{1}{14}$
(-1,4.5)
$\frac{1}{2}$
(1,2)
(6,3)
$\frac{1}{14}$
$\frac{1}{14}$
(-3,0.75)

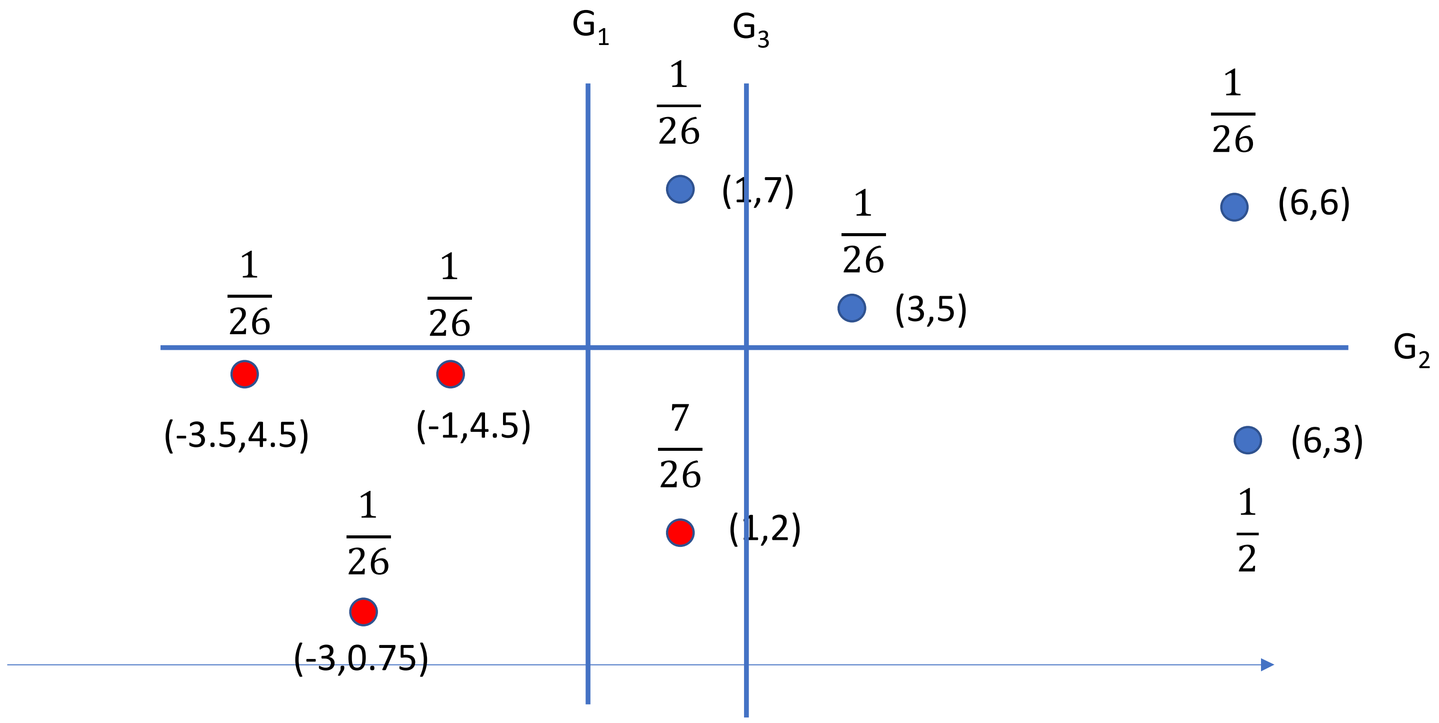
Let G_2 be the split at $x_2 = 4.75$ so $G_2(x_1, x_2) = \begin{cases} -1 \text{ if } x_2 \leq 4.75 \\ +1 \text{ if } x_2 > 4.75 \end{cases}$.

This misclassifies the point $(6, 3)$ and the weighted error $err_2 = \frac{1}{14}$. Hence $\beta_2 = \frac{1}{2} \log 13$. The classifier at this stage is

$$sign(\log(7)G_1 + \log(13)G_2)$$

The new weights are $w_{(6,3)} = \frac{13}{14}$ and the others unchanged. The sum of the new weights is $\frac{13}{14} + \frac{1}{2} + 6\frac{1}{14} = \frac{26}{14}$ so after normalizing we get

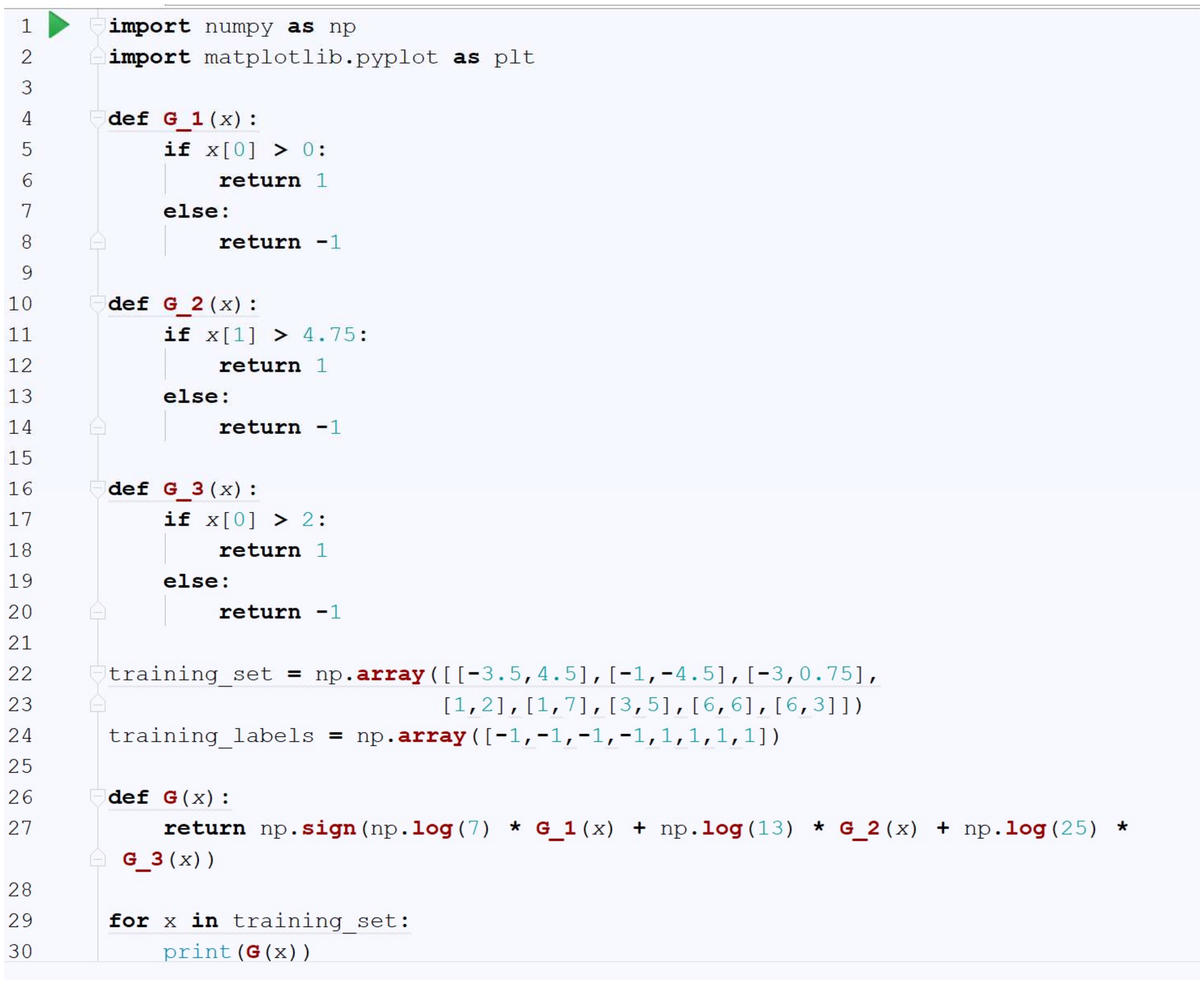
$$w_{(6,3)} = \frac{1}{2}, w_{(1,2)} = \frac{7}{26}, \text{ all others } = \frac{1}{26}$$


G_1
G_3
G_2
1/26
(1,7)
1/26
(6,6)
1/26
(3,5)
1/26
(-3.5,4.5)
1/26
(-1,4.5)
7/26
(1,2)
(6,3)
1/2
1/26
(-3,0.75)

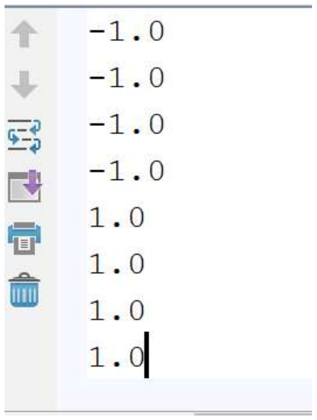
Now let G_3 be the split at $x_1 = 2$. Then $(1,7)$ is misclassified and the weighted error is $err_3 = \frac{1}{26}$. Thus $\beta_3 = \frac{1}{2}\log 25$ and the new classifier is

$$sign\,((\log 7)G_1 + (\log 13)G_2 + \log(25)G_3)$$

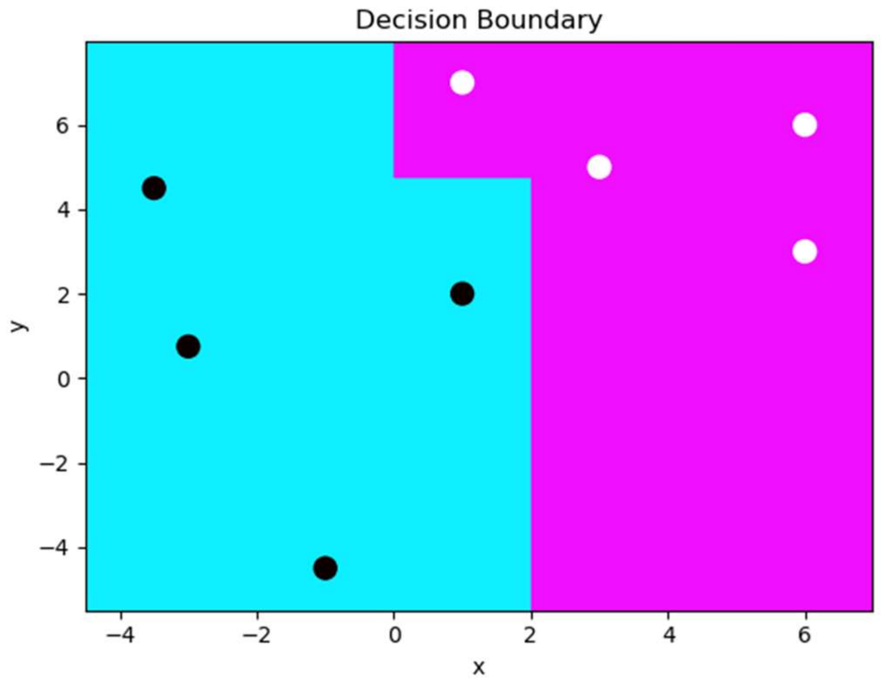
This classifier actually correctly classifies all the points



```python
import numpy as np
import matplotlib.pyplot as plt

def G_1(x):
    if x[0] > 0:
        return 1
    else:
        return -1

def G_2(x):
    if x[1] > 4.75:
        return 1
    else:
        return -1

def G_3(x):
    if x[0] > 2:
        return 1
    else:
        return -1

training_set = np.array([[-3.5,4.5],[-1,-4.5],[-3,0.75],
                         [1,2],[1,7],[3,5],[6,6],[6,3]])
training_labels = np.array([-1,-1,-1,-1,1,1,1,1])

def G(x):
    return np.sign(np.log(7) * G_1(x) + np.log(13) * G_2(x) + np.log(25) * G_3(x))

for x in training_set:
    print(G(x))
```



```
-1.0
-1.0
-1.0
-1.0
1.0
1.0
1.0
1.0
```


Decision Boundary
y
x