Homework2

Kai Wang

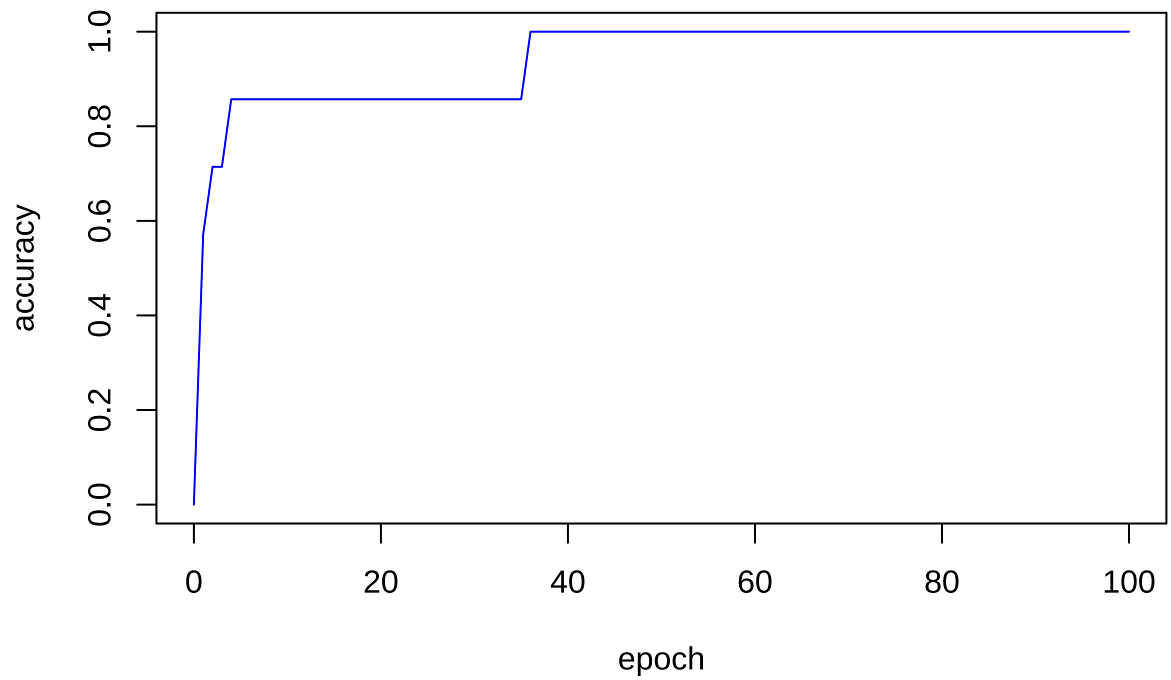
2018/1/30

1 Classifiers for Basketball Courts

a

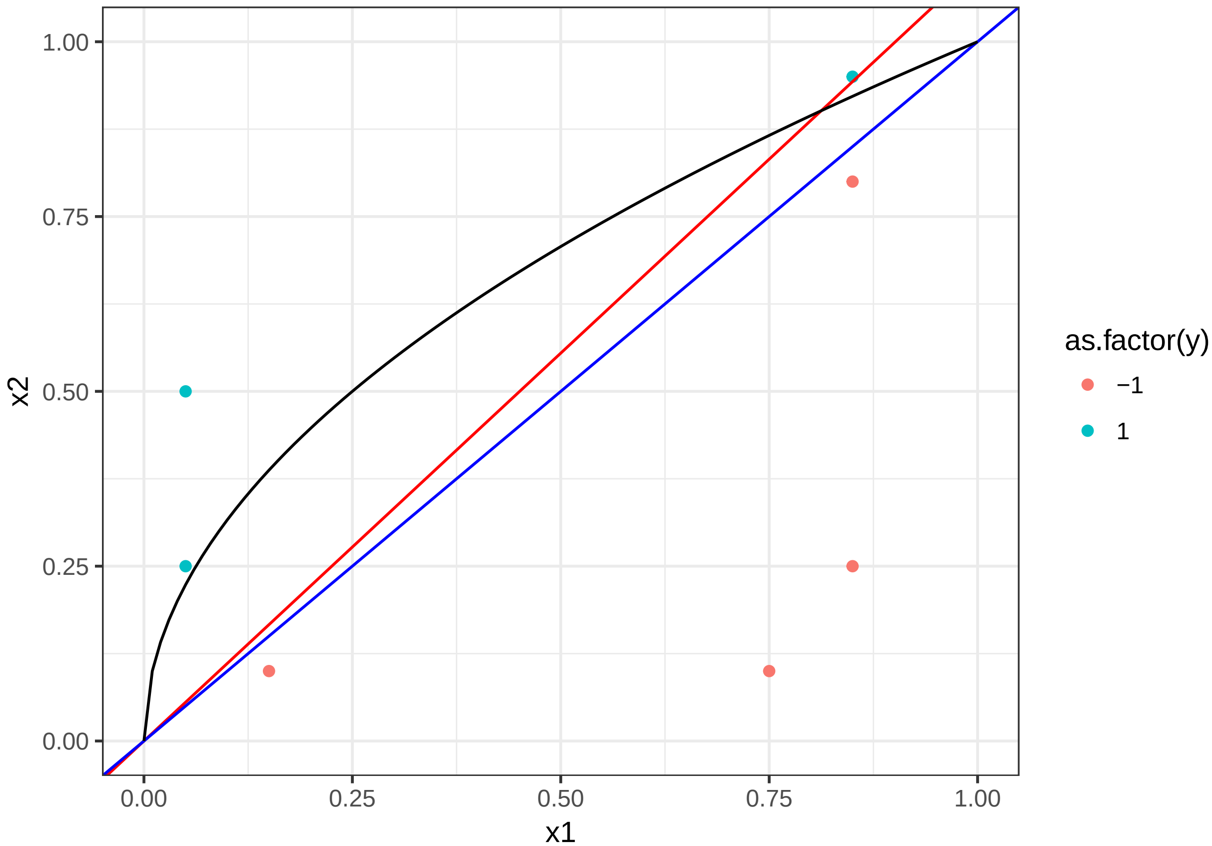
```
#oberserved shots
shots = data.frame(x1=c(.75,.85,.85,.15,.05,.05,.85),
                   x2=c(.10,.80,.95,.10,.25,.50,.25),
                   y=c(-1,-1,1,-1,1,1,-1))
shots.data = shots[,c(1,2)] %>% as.matrix()
shots.label = shots[,3] %>% as.matrix()

#perceptron algorithm from HW1
perceptron = function(x, y, epoch) {
        # initialize weight vector
        weight = rep(0, dim(x)[2])
        result = matrix(0, nrow = epoch, ncol = dim(x)[2])
        for (i in 1:epoch) {
                for (j in 1:length(y)) {
                        z = sum(weight*x[j, ])
                        if(z <= 0) {
                                ypred = -1
                        } else {
                                ypred = 1
                        }

                        # Update weight
                        if (y[j] != ypred) {
                          weight = weight + y[j] * x[j,]
                        }
                }
          #save weight vectors for each step
          result[i,] = weight
        }
        return(result)
}
```

```
weight = perceptron(shots.data, shots.label,100)
pred.train = t(shots.data %*% t(weight))
pred.train[pred.train>0] = 1
pred.train[pred.train<=0] = -1
accuracy = apply(pred.train, 1, function(x) {return(sum(x==shots.label)/length(shots.label))})
accuracy = c(0,accuracy)
#plot accuracy vs. epoch
plot(x = 0:100, y=accuracy, ylim = range(0,1), type = "l", col = "blue", xlab = "epoch", ylab = "accura
```



From the epoch vs. accuracy plot, we can see the perceptron converges at 37th iteration. Since the accuracy is 100%, there's no empirical error for this classifier. We can come up with other linear classifiers which will give the same error(0). For example, an boolean function $y = f(x_1, x_2) = \mathbb{I}_{-x_1+x_2>0}$. More generally, as long as the slope of the linear classifier passing through origin is between $(\frac{16}{17}, \frac{19}{17})$.

```
#plot observed data and another seperation line
ggplot(shots) + geom_point(aes(x = x1, y = x2, col = as.factor(y)))+
  theme_bw() + geom_abline(slope = -weight[100,1]/weight[100,2], color = 'red') +
  geom_abline(slope = 1, intercept = 0, color = 'blue') +
  stat_function(fun=function(x) sqrt(x), xlim = c(0,1))
```



The above plot shows data and the decision boundary of the perceptron (as red line). In addition, the boolean function $y = f(x_1, x_2) = \mathbb{I}_{-x_1+x_2+0.08>0}$ is also plotted as a black line, which clearly seperates our data with no empirical error.

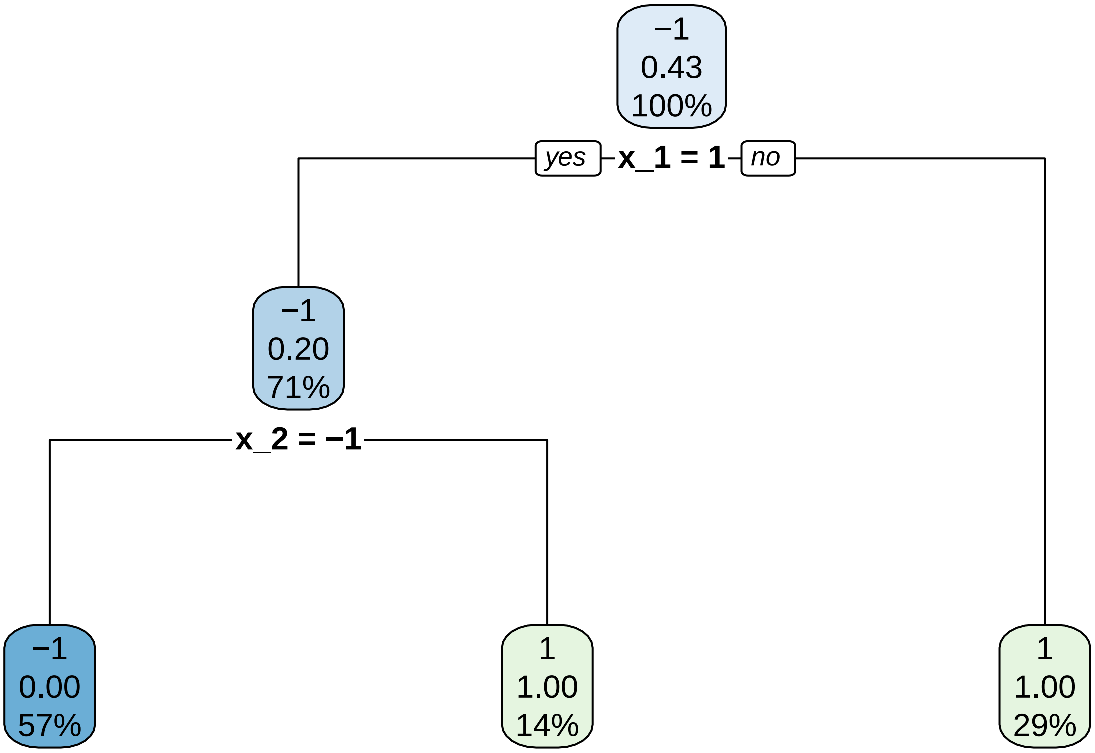
b

```r
#split the decision tree based on gini index
shots = shots %>%
  mutate(x_1 = ifelse(x1>0.1, 1, -1) %>% as.factor(),
         x_2 = ifelse(x2>0.85, 1, -1) %>% as.factor())
fit1 = rpart(as.factor(y)~x_1+x_2, data = shots, method = "class", parms = list(split = "gini"),
             control=rpart.control(minsplit=1))
summary(fit1)
```

```
## Call:
## rpart(formula = as.factor(y) ~ x_1 + x_2, data = shots, method = "class",
##     parms = list(split = "gini"), control = rpart.control(minsplit = 1))
##   n= 7
##
##          CP nsplit rel error    xerror      xstd
## 1 0.6666667      0 1.0000000 1.0000000 0.4364358
## 2 0.3333333      1 0.3333333 1.0000000 0.4364358
## 3 0.0100000      2 0.0000000 0.3333333 0.3086067
##
## Variable importance
## x_1 x_2
```

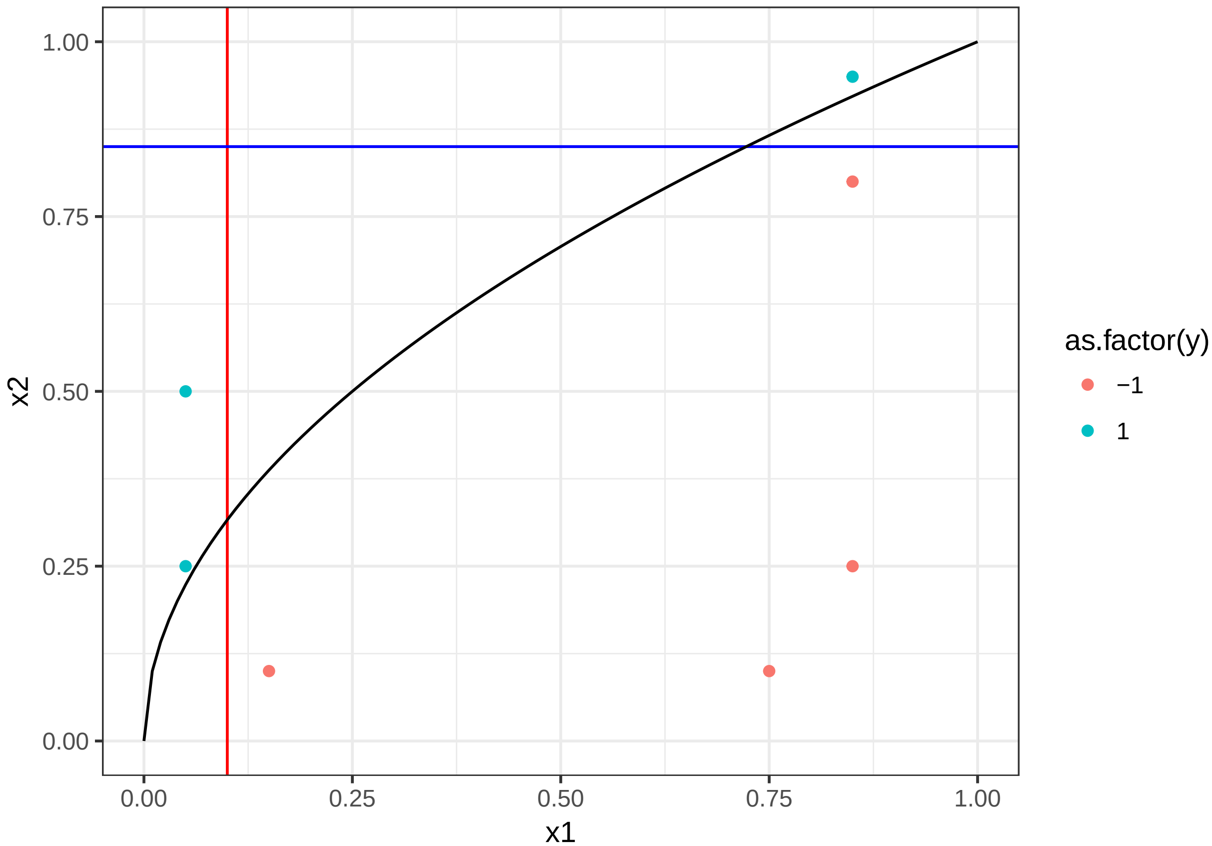
```
##  53  47
##
## Node number 1: 7 observations,    complexity param=0.6666667
##   predicted class=-1  expected loss=0.4285714  P(node) =1
##     class counts:     4     3
##    probabilities: 0.571 0.429
##   left son=2 (5 obs) right son=3 (2 obs)
##   Primary splits:
##       x_1 splits as  RL, improve=1.8285710, (0 missing)
##       x_2 splits as  LR, improve=0.7619048, (0 missing)
##
## Node number 2: 5 observations,    complexity param=0.3333333
##   predicted class=-1  expected loss=0.2  P(node) =0.7142857
##     class counts:     4     1
##    probabilities: 0.800 0.200
##   left son=4 (4 obs) right son=5 (1 obs)
##   Primary splits:
##       x_2 splits as  LR, improve=1.6, (0 missing)
##
## Node number 3: 2 observations
##   predicted class=1   expected loss=0  P(node) =0.2857143
##     class counts:     0     2
##    probabilities: 0.000 1.000
##
## Node number 4: 4 observations
##   predicted class=-1  expected loss=0  P(node) =0.5714286
##     class counts:     4     0
##    probabilities: 1.000 0.000
##
## Node number 5: 1 observations
##   predicted class=1   expected loss=0  P(node) =0.1428571
##     class counts:     0     1
##    probabilities: 0.000 1.000
```

```r
rpart.plot(fit1)
```



We can use "rpart" package to split our decision tree based on gini index. And this decision tree has all data points correctly classified.

```
ggplot(shots) + geom_point(aes(x = x1, y = x2, col = as.factor(y)))+
  theme_bw() + geom_vline(xintercept = 0.1, color = "red") + geom_hline(yintercept = 0.85, color = "blu
  stat_function(fun=function(x) sqrt(x), xlim = c(0,1))
```



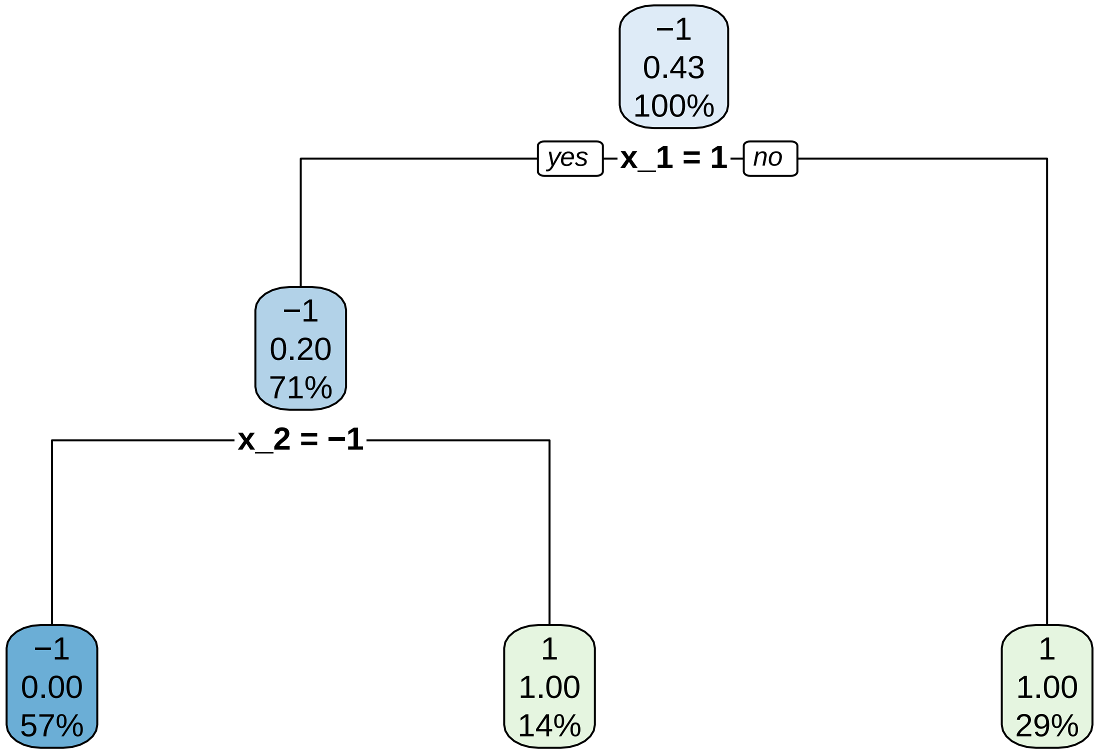
We can choose $x_1 > 0.1$ and $x_2 > 0.85$ as our threshold for x_1, x_2, then split the decision tree based on transformed data. Using the reduction in the Gini index as the splitting criterion, the error is 0. We can adjust the threshold for x_1 between $[0.05, 0.15)$ and for x_2 bewtween $[0.80, 0.95)$, which will give us the same split on the data. The decision tree of such a threshold is plotted as follow.

```
shots = shots %>%
  mutate(x_1 = ifelse(x1>0.08, 1, -1) %>% as.factor(),
         x_2 = ifelse(x2>0.90, 1, -1) %>% as.factor())
fit2 = rpart(as.factor(y)~x_1+x_2, data = shots, method = "class", parms = list(split = "gini"),
             control=rpart.control(minsplit=1))
summary(fit2)
```

```
## Call:
## rpart(formula = as.factor(y) ~ x_1 + x_2, data = shots, method = "class",
##     parms = list(split = "gini"), control = rpart.control(minsplit = 1))
##   n= 7
##
##          CP nsplit rel error    xerror      xstd
## 1 0.6666667      0 1.0000000 1.0000000 0.4364358
## 2 0.3333333      1 0.3333333 1.0000000 0.4364358
## 3 0.0100000      2 0.0000000 0.3333333 0.3086067
##
## Variable importance
## x_1 x_2
##  53  47
##
## Node number 1: 7 observations,    complexity param=0.6666667
##   predicted class=-1  expected loss=0.4285714  P(node) =1
```

```
##     class counts:     4     3
##    probabilities: 0.571 0.429
##   left son=2 (5 obs) right son=3 (2 obs)
##   Primary splits:
##       x_1 splits as  RL, improve=1.8285710, (0 missing)
##       x_2 splits as  LR, improve=0.7619048, (0 missing)
##
## Node number 2: 5 observations,    complexity param=0.3333333
##   predicted class=-1  expected loss=0.2  P(node) =0.7142857
##     class counts:     4     1
##    probabilities: 0.800 0.200
##   left son=4 (4 obs) right son=5 (1 obs)
##   Primary splits:
##       x_2 splits as  LR, improve=1.6, (0 missing)
##
## Node number 3: 2 observations
##   predicted class=1   expected loss=0  P(node) =0.2857143
##     class counts:     0     2
##    probabilities: 0.000 1.000
##
## Node number 4: 4 observations
##   predicted class=-1  expected loss=0  P(node) =0.5714286
##     class counts:     4     0
##    probabilities: 1.000 0.000
##
## Node number 5: 1 observations
##   predicted class=1   expected loss=0  P(node) =0.1428571
##     class counts:     0     1
##    probabilities: 0.000 1.000
```

```r
rpart.plot(fit2)
```



c

Since the three-point line is our approximation is $x_2 = \sqrt{x_1}$, we know $f(x)^{\text{true}} = \text{sign}(x_2 - \sqrt{x_1})$. We need to use a linear classifier that goes through origin, say $f(\mathbf{x}) = \text{sign}(x_2 - ax_1)$.

The true risk is $R^{\text{true}}(f) = \mathbb{E}_{(x,y)\sim D} l(f(\mathbf{x}), y)$, with the misclassification function $l(f(\mathbf{x}), y) = 1_{\text{sign}(f(\mathbf{x}) \neq y)}$.

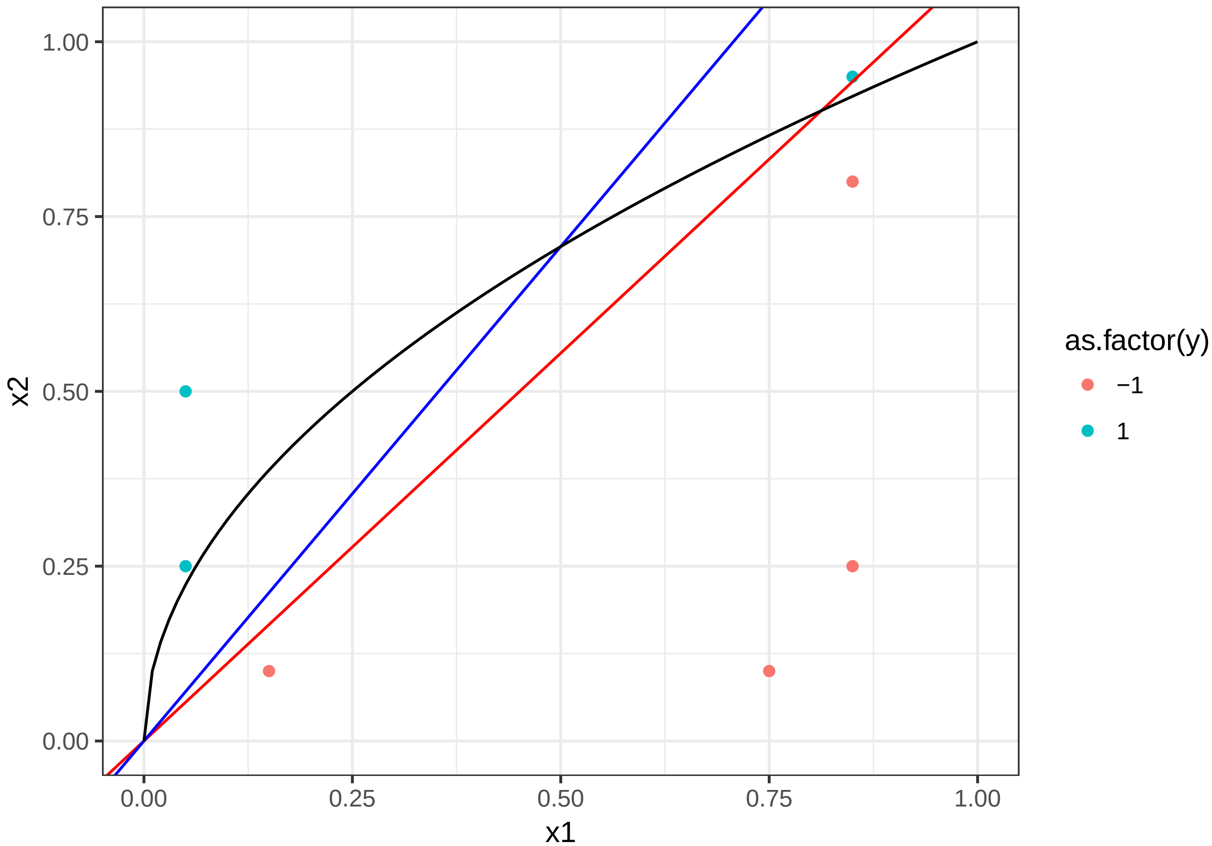
Since $x_1, x_2 \sim \text{Uniform}[0, 1]$, we transform this problem into finding the definite integral of area between $x_2 = \sqrt{x_1}$ and $x_2 = ax_1$, and it's intuitive that $a \geq 1$.

$$
\begin{aligned}
R^{\text{true}}(f) &= \int_0^{\frac{1}{a^2}} (\sqrt{x_1} - ax_1)dx_1 + \int_{\frac{1}{a^2}}^{\frac{1}{a}} (ax_1 - \sqrt{x_1})dx_1 + \int_{\frac{1}{a}}^{1} (1 - \sqrt{x_1})dx_1 \\
&= \frac{2}{3}a^{-3} - \frac{a}{2}a^{-4} + \frac{a}{2}a^{-2} - \frac{a}{2}a^{-4} - \frac{2}{3}a^{-\frac{3}{2}} + \frac{2}{3}a^{-3} + 1 - \frac{1}{a} - \frac{2}{3} + \frac{2}{3}a^{-\frac{3}{2}} \\
&= \frac{1}{3}a^{-3} - \frac{1}{2}a^{-1} + \frac{1}{3}
\end{aligned}
$$

To minimize the true risk, we need to choose a value of a so that $R^{\text{true}}(f)$ is minimized.

We have $\frac{dR^{\text{true}}(f)}{da} = \frac{1}{2a^2} - \frac{1}{a^4}$, which equals to 0 when $a = \sqrt{2}$. The second derivative shows that $R^{\text{true}}(f)$ is increasing after $a = \sqrt{2}$. Hence, $R^{\text{true}}(f)_{\min} = \frac{1}{3} - \frac{\sqrt{2}}{6}$.

```r
#plot observed data and another seperation line
ggplot(shots) + geom_point(aes(x = x1, y = x2, col = as.factor(y)))+
  theme_bw() + geom_abline(slope = -weight[100,1]/weight[100,2], color = 'red') +
  geom_abline(slope = sqrt(2), intercept = 0, color = 'blue') +
  stat_function(fun=function(x) sqrt(x), xlim = c(0,1))
```



The blue line shows the optimal linear classifier, but empirically it classified 1 point wrong. The empirical error $R(f) = \frac{1}{8}$. Any linear classifier with slope between $(\frac{2}{3}, 5)$ would result in the same empirical error.

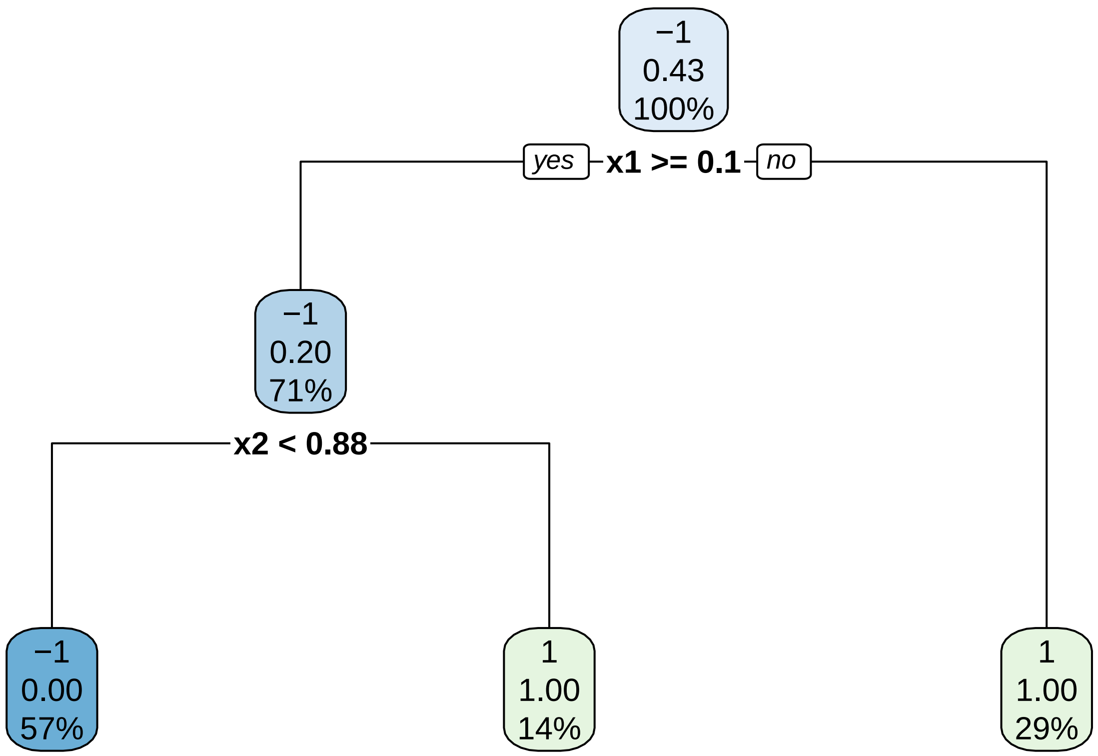
d

```
fit3 = rpart(as.factor(y)~x1+x2, data = shots, method = "class", parms = list(split = "gini"),
             control=rpart.control(minsplit=1))
summary(fit3)
```

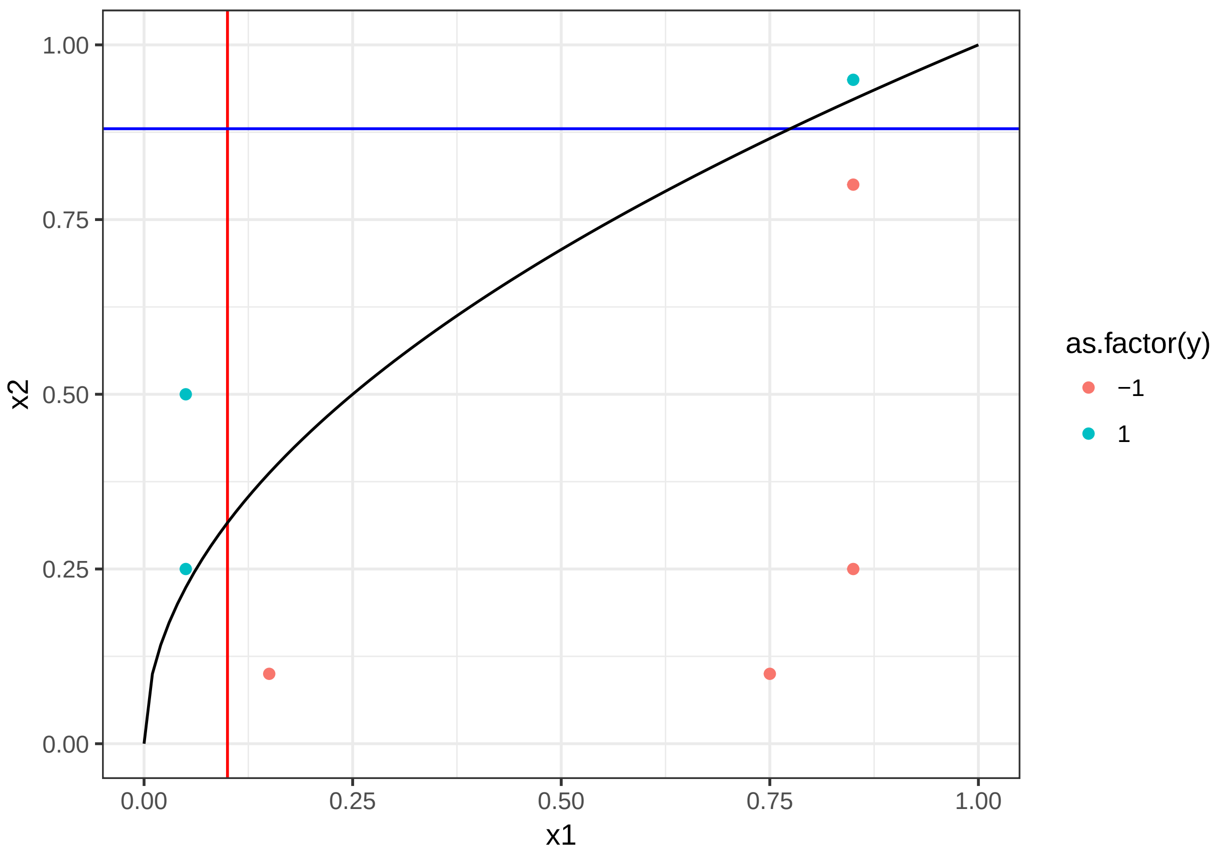
```
## Call:
## rpart(formula = as.factor(y) ~ x1 + x2, data = shots, method = "class",
##     parms = list(split = "gini"), control = rpart.control(minsplit = 1))
##   n= 7
##
##          CP nsplit rel error   xerror      xstd
## 1 0.6666667      0 1.0000000 1.000000 0.4364358
## 2 0.3333333      1 0.3333333 1.333333 0.4364358
## 3 0.0100000      2 0.0000000 1.333333 0.4364358
##
## Variable importance
## x1 x2
## 53 47
##
## Node number 1: 7 observations,    complexity param=0.6666667
##   predicted class=-1  expected loss=0.4285714  P(node) =1
##     class counts:     4     3
```

```
##    probabilities: 0.571 0.429
##   left son=2 (5 obs) right son=3 (2 obs)
##   Primary splits:
##       x1 < 0.1   to the right, improve=1.828571, (0 missing)
##       x2 < 0.175 to the left,  improve=1.028571, (0 missing)
##
## Node number 2: 5 observations,    complexity param=0.3333333
##   predicted class=-1  expected loss=0.2  P(node) =0.7142857
##     class counts:     4     1
##    probabilities: 0.800 0.200
##   left son=4 (4 obs) right son=5 (1 obs)
##   Primary splits:
##       x2 < 0.875 to the left,  improve=1.6000000, (0 missing)
##       x1 < 0.8   to the left,  improve=0.2666667, (0 missing)
##
## Node number 3: 2 observations
##   predicted class=1   expected loss=0  P(node) =0.2857143
##     class counts:     0     2
##    probabilities: 0.000 1.000
##
## Node number 4: 4 observations
##   predicted class=-1  expected loss=0  P(node) =0.5714286
##     class counts:     4     0
##    probabilities: 1.000 0.000
##
## Node number 5: 1 observations
##   predicted class=1   expected loss=0  P(node) =0.1428571
##     class counts:     0     1
##    probabilities: 0.000 1.000
```

```
rpart.plot(fit3)
```




```r
ggplot(shots) + geom_point(aes(x = x1, y = x2, col = as.factor(y)))+
  theme_bw() + geom_vline(xintercept = 0.1, color = "red") + geom_hline(yintercept = 0.88, color = "blu
  stat_function(fun=function(x) sqrt(x), xlim = c(0,1))
```



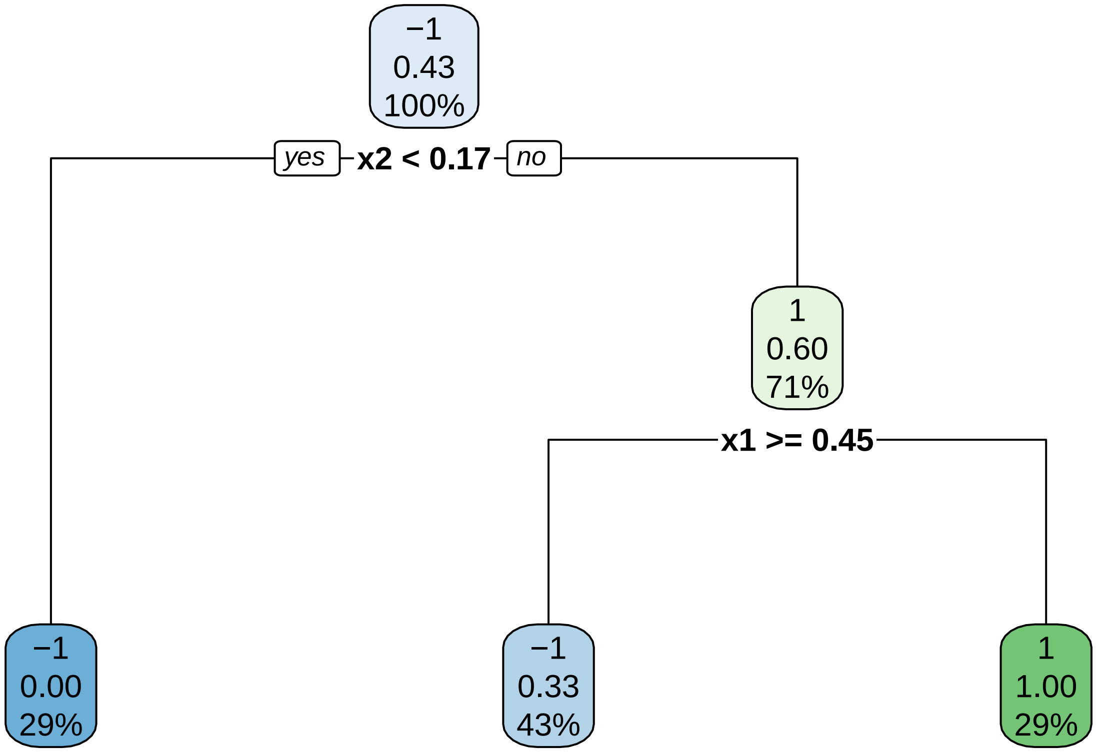
First decision tree splits on x_1 first and the x_2, we can see $s_1 = 0.1$ and $s_3 = 0.88$. s_2 does not exist since there is no split with $x_1 <= 0.1$. This is among the solutions that achieved the minimum empirical error.

```
#split on x2 first by assigning higher cost to x1
fit4 = rpart(as.factor(y)~x1+x2, data = shots, method = "class", parms = list(split = "gini"),
             control=rpart.control(minsplit=1, maxdepth = 2), cost = c(2,1))
summary(fit4)
```

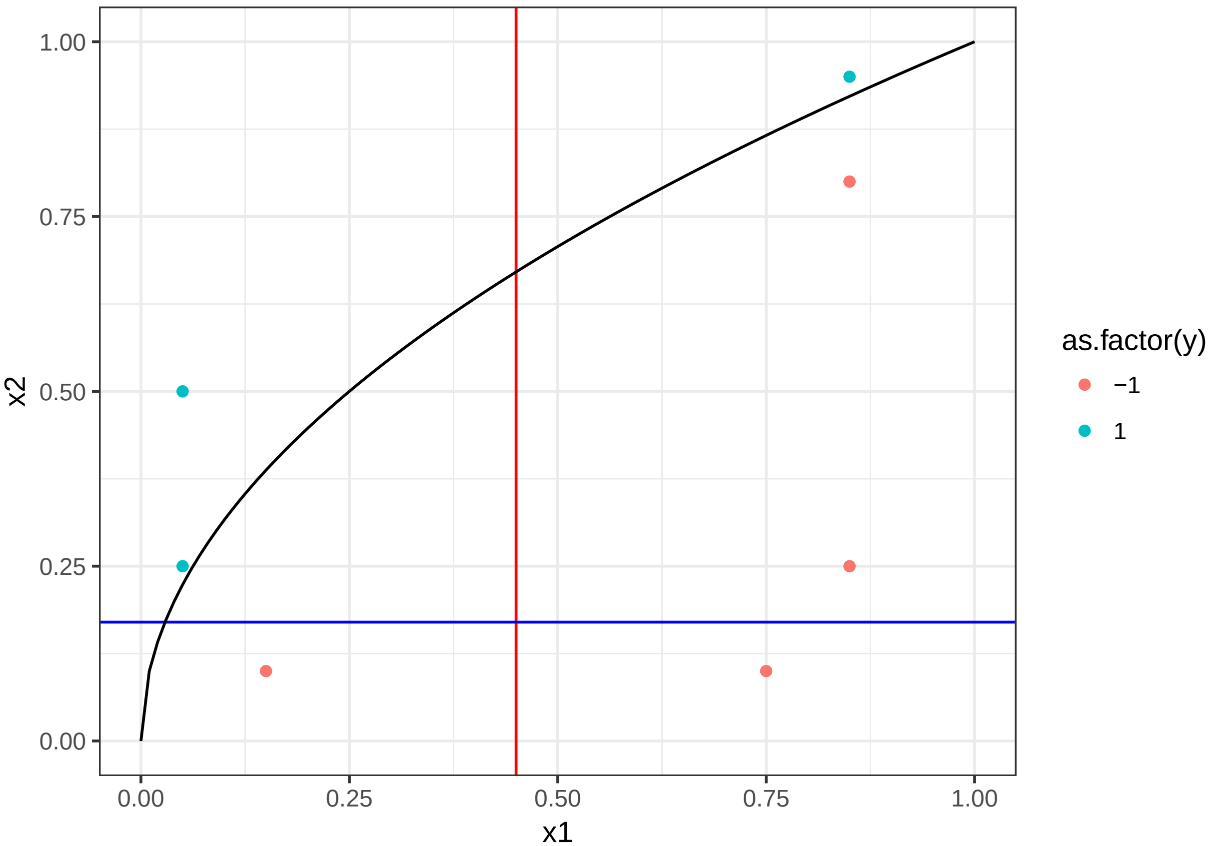
```
## Call:
## rpart(formula = as.factor(y) ~ x1 + x2, data = shots, method = "class",
##     parms = list(split = "gini"), control = rpart.control(minsplit = 1,
##         maxdepth = 2), cost = c(2, 1))
##   n= 7
##
##          CP nsplit rel error   xerror      xstd
## 1 0.3333333      0 1.0000000 1.000000 0.4364358
## 2 0.0100000      2 0.3333333 1.666667 0.3984095
##
## Variable importance
## x2 x1
## 71 29
##
## Node number 1: 7 observations,    complexity param=0.3333333
##   predicted class=-1  expected loss=0.4285714  P(node) =1
##     class counts:     4     3
##    probabilities: 0.571 0.429
##   left son=2 (2 obs) right son=3 (5 obs)
##   Primary splits:
##       x2 < 0.175 to the left,  improve=1.0285710, (0 missing)
##       x1 < 0.1   to the right, improve=0.9142857, (0 missing)
##
## Node number 2: 2 observations
##   predicted class=-1  expected loss=0  P(node) =0.2857143
##     class counts:     2     0
##    probabilities: 1.000 0.000
##
## Node number 3: 5 observations,    complexity param=0.3333333
##   predicted class=1   expected loss=0.4  P(node) =0.7142857
##     class counts:     2     3
##    probabilities: 0.400 0.600
##   left son=6 (3 obs) right son=7 (2 obs)
##   Primary splits:
##       x1 < 0.45  to the right, improve=0.5333333, (0 missing)
##       x2 < 0.875 to the left,  improve=0.4000000, (0 missing)
##   Surrogate splits:
##       x2 < 0.65  to the right, agree=0.8, adj=0.5, (0 split)
##
## Node number 6: 3 observations
##   predicted class=-1  expected loss=0.3333333  P(node) =0.4285714
##     class counts:     2     1
##    probabilities: 0.667 0.333
##
## Node number 7: 2 observations
##   predicted class=1   expected loss=0  P(node) =0.2857143
##     class counts:     0     2
```

```
##    probabilities: 0.000 1.000
```

```r
rpart.plot(fit4)
```




```r
#plot second tree
ggplot(shots) + geom_point(aes(x = x1, y = x2, col = as.factor(y)))+
  theme_bw() + geom_vline(xintercept = 0.45, color = "red") + geom_hline(yintercept = 0.17, color = "blu
  stat_function(fun=function(x) sqrt(x), xlim = c(0,1))
```

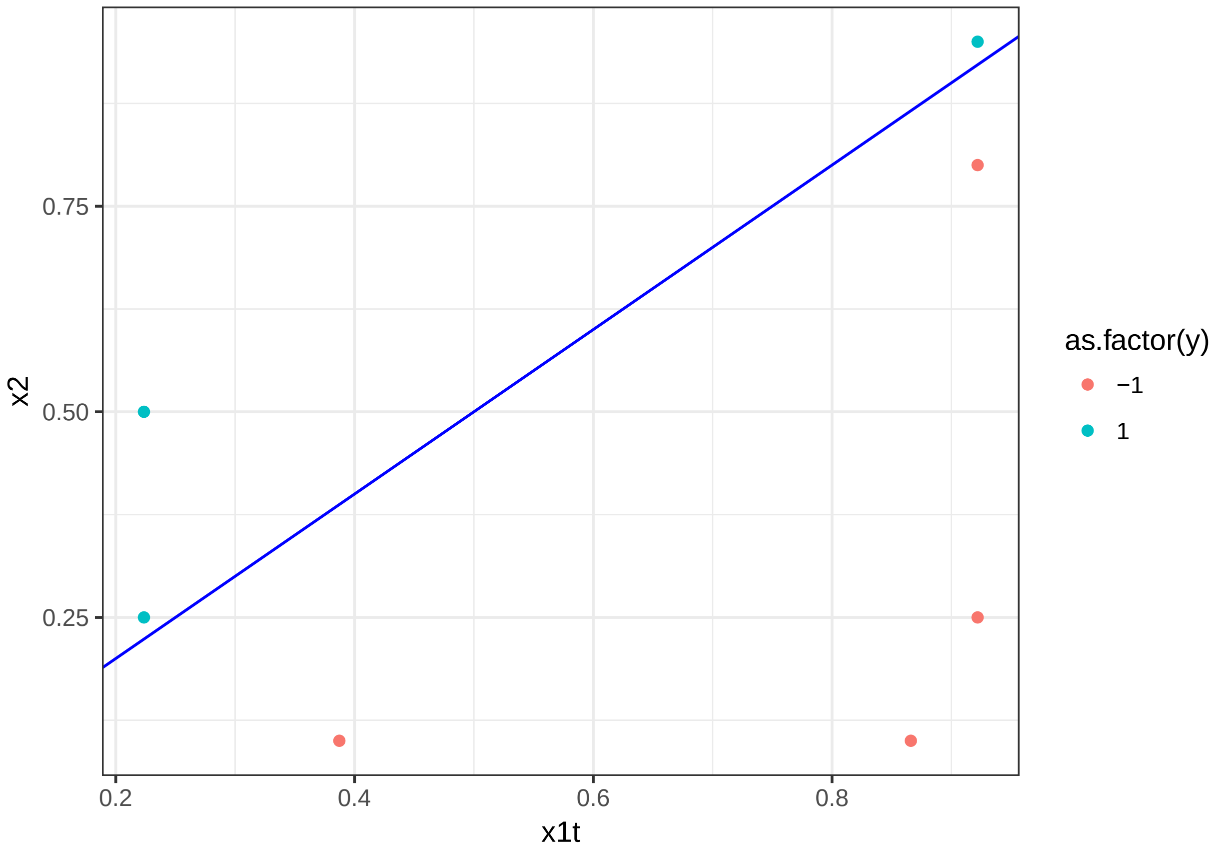


Second decision tree splits on x_2 first and the x_1, we can see $s_1 = 0.17$ and $s_2 = 0.45$. s_3 does not exist since there is no split with $x_2 <= 0.17$. This is not among the solutions that achieved the minimum empirical error.

e

Since we know the true function $f(\mathbf{x}) = \mathbb{I}_{x_2 - \sqrt{x_1} > 0}$, we can make a tranformation $x_2 = \sqrt{x_1}$. Then our optimal linear classifier will simply be $y = \mathbb{I}_{x_2 - x_1 > 0}$ and its error is 0.

```r
#plot with transformation on x1
shots = shots %>%
  mutate(x1t = sqrt(x1))
ggplot(shots) + geom_point(aes(x = x1t, y = x2, col = as.factor(y)))+
  theme_bw() + geom_abline(slope = 1, color = 'blue')
```



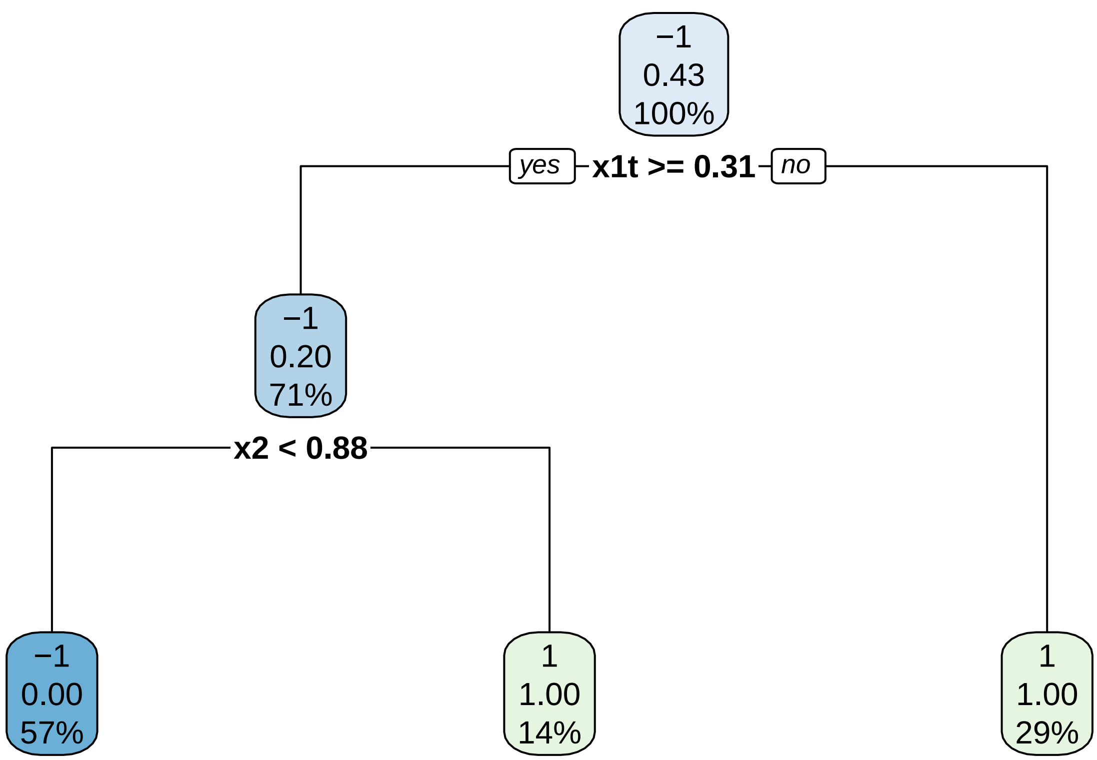
f

```
fit5 = rpart(as.factor(y)~x1t+x2, data = shots, method = "class", parms = list(split = "gini"),
             control=rpart.control(minsplit=1))
summary(fit5)
```

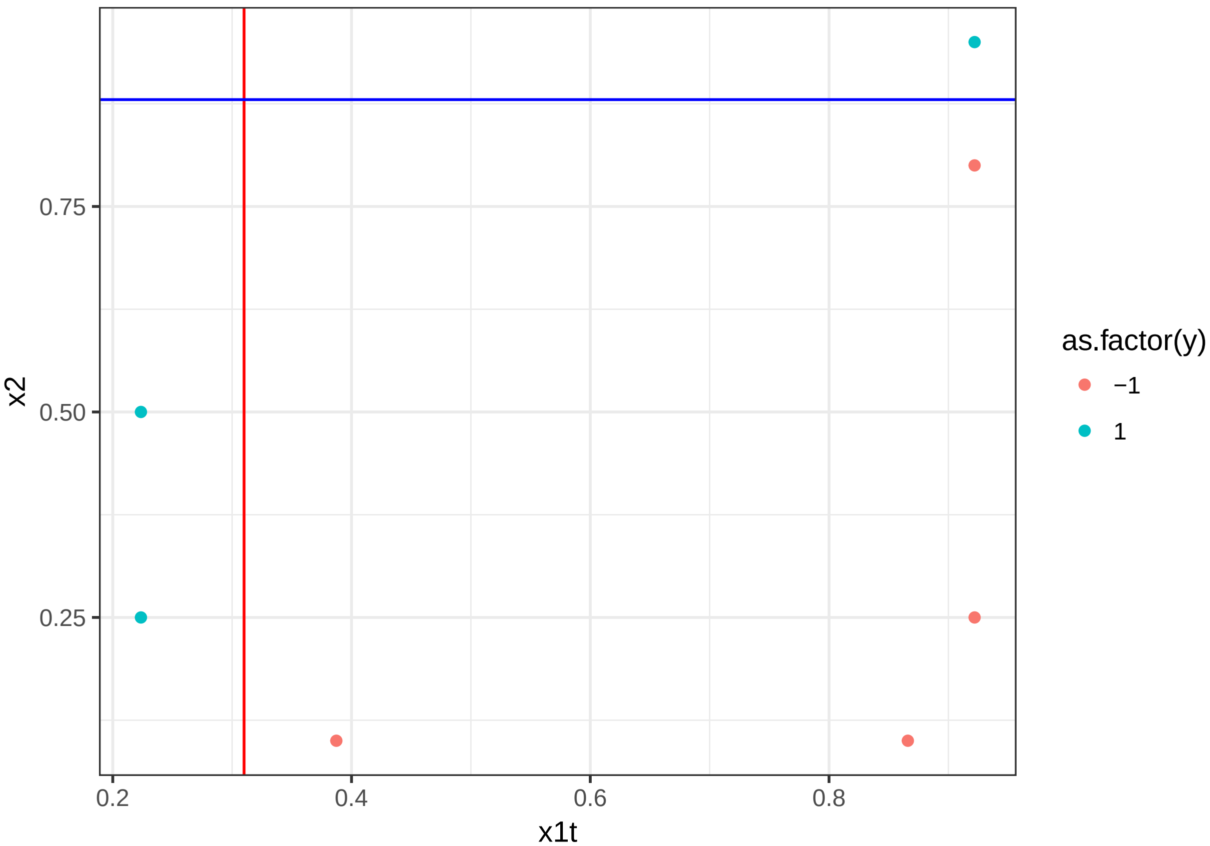
```
## Call:
## rpart(formula = as.factor(y) ~ x1t + x2, data = shots, method = "class",
##     parms = list(split = "gini"), control = rpart.control(minsplit = 1))
##   n= 7
##
##          CP nsplit rel error   xerror      xstd
## 1 0.6666667      0 1.0000000 1.000000 0.4364358
## 2 0.3333333      1 0.3333333 1.333333 0.4364358
## 3 0.0100000      2 0.0000000 1.333333 0.4364358
##
## Variable importance
## x1t  x2
##  53  47
##
## Node number 1: 7 observations,    complexity param=0.6666667
##   predicted class=-1  expected loss=0.4285714  P(node) =1
##     class counts:     4     3
##    probabilities: 0.571 0.429
##   left son=2 (5 obs) right son=3 (2 obs)
```

```
##   Primary splits:
##       x1t < 0.3054526 to the right, improve=1.828571, (0 missing)
##       x2  < 0.175     to the left,  improve=1.028571, (0 missing)
##
## Node number 2: 5 observations,    complexity param=0.3333333
##   predicted class=-1  expected loss=0.2  P(node) =0.7142857
##     class counts:     4     1
##    probabilities: 0.800 0.200
##   left son=4 (4 obs) right son=5 (1 obs)
##   Primary splits:
##       x2  < 0.875     to the left,  improve=1.6000000, (0 missing)
##       x1t < 0.8939899 to the left,  improve=0.2666667, (0 missing)
##
## Node number 3: 2 observations
##   predicted class=1   expected loss=0  P(node) =0.2857143
##     class counts:     0     2
##    probabilities: 0.000 1.000
##
## Node number 4: 4 observations
##   predicted class=-1  expected loss=0  P(node) =0.5714286
##     class counts:     4     0
##    probabilities: 1.000 0.000
##
## Node number 5: 1 observations
##   predicted class=1   expected loss=0  P(node) =0.1428571
##     class counts:     0     1
##    probabilities: 0.000 1.000
```

```
rpart.plot(fit5)
```



```
ggplot(shots) + geom_point(aes(x = x1t, y = x2, col = as.factor(y)))+
  theme_bw() + geom_vline(xintercept = 0.31, color = "red") + geom_hline(yintercept = 0.88, color = "blu
```



After the transformation on x_1, the decision tree achieved the same error.

h

We can represent the paint as area where $x_1 \geq 0.5, x_2 \leq 0.25$.

The true risk is $R^{\text{true}}(f) = \mathbb{E}_{(x,y)\sim D} l(f(\mathbf{x}), y)$, with the misclassification function $l(f(\mathbf{x}), y) = 1_{\text{sign}(f(\mathbf{x})\neq y)}$.

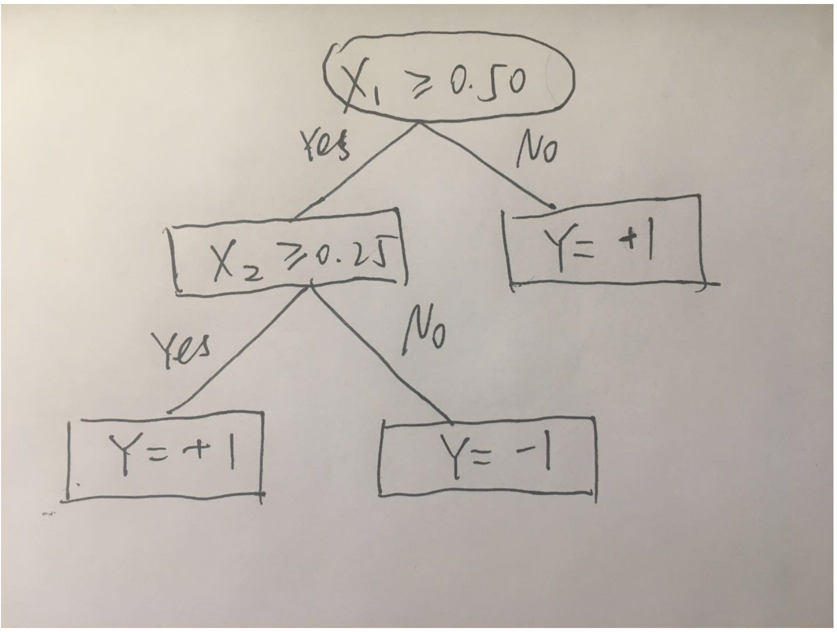
Since $x_1, x_2 \sim \text{Uniform}[0, 1]$, we transform this problem into finding the definite integral of area between $x_2 = \begin{cases} 0.25, & x_1 \geq 0.5 \\ 0, & x_2 < 0.5 \end{cases}$ and $x_2 = ax_1$, and it's intuitive that $a \leq \frac{1}{2}$.

$$
\begin{aligned}
R^{\text{true}}(f) &= \int_0^{\frac{1}{2}} ax_1 dx_1 + \int_{\frac{1}{2}}^{\frac{1}{4a}} (\frac{1}{4} - ax_1) dx_1 + \int_{\frac{1}{4a}}^{1} (ax_1 - 1) dx_1 \\
&= \frac{3}{4}a + \frac{1}{4a} - \frac{9}{8}
\end{aligned}
$$

To minimize $R^{\text{true}}(f)$, we have $\frac{dR^{\text{true}}(f)}{da} = \frac{3}{4} - \frac{1}{4a^2} = 0$, which means $a = \frac{\sqrt{3}}{3}$. Since $R^{\text{true}}(f)$ is decreasing on $[\frac{1}{2}, \frac{\sqrt{3}}{3}]$ and increasing afterwards, we have $R^{\text{true}}(f)_{\min} = \frac{\sqrt{3}}{2} - \frac{9}{8}$.

i

The optimal decision tree will have decision boundaries that replicates the shape of paint and the error is 0.



2

a

i

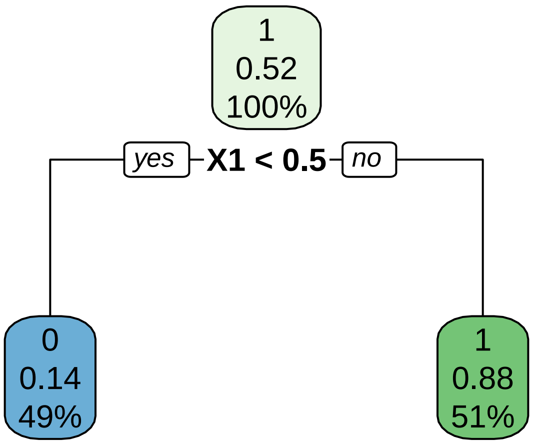
```
#read in datasets
train = read.csv("train.csv")
test = read.csv("test.csv")
```

```
#best split
stump1 = rpart(Y~., data = train, method = "class", parms = list(split = "gini"),
             control=rpart.control(minsplit=1, maxdepth = 1))
summary(stump1)
```

```
## Call:
## rpart(formula = Y ~ ., data = train, method = "class", parms = list(split = "gini"),
##     control = rpart.control(minsplit = 1, maxdepth = 1))
##   n= 500
##
##          CP nsplit rel error    xerror       xstd
## 1 0.7261411      0 1.0000000 1.0000000 0.04636138
## 2 0.0100000      1 0.2738589 0.2738589 0.03140616
##
## Variable importance
## X1 X2 X5
## 62 37  1
##
```

```
## Node number 1: 500 observations,    complexity param=0.7261411
##   predicted class=1  expected loss=0.482  P(node) =1
##     class counts:   241   259
##    probabilities: 0.482 0.518
##   left son=2 (243 obs) right son=3 (257 obs)
##   Primary splits:
##       X1 < 0.5 to the left,  improve=135.15930000, (0 missing)
##       X2 < 0.5 to the left,  improve= 52.78111000, (0 missing)
##       X3 < 0.5 to the left,  improve=  0.29823390, (0 missing)
##       X4 < 0.5 to the right, improve=  0.27751560, (0 missing)
##       X5 < 0.5 to the left,  improve=  0.05810746, (0 missing)
##   Surrogate splits:
##       X2 < 0.5 to the left,  agree=0.806, adj=0.601, (0 split)
##       X5 < 0.5 to the left,  agree=0.518, adj=0.008, (0 split)
##       X4 < 0.5 to the right, agree=0.516, adj=0.004, (0 split)
##
## Node number 2: 243 observations
##   predicted class=0  expected loss=0.1399177  P(node) =0.486
##     class counts:   209    34
##    probabilities: 0.860 0.140
##
## Node number 3: 257 observations
##   predicted class=1  expected loss=0.1245136  P(node) =0.514
##     class counts:    32   225
##    probabilities: 0.125 0.875
```

```
rpart.plot(stump1)
```




```
#best surrogate split
stump2 = rpart(Y~., data = train, method = "class", parms = list(split = "gini"),
             control=rpart.control(minsplit=1, maxdepth = 1, usesurrogate = 1))
summary(stump2)
```

```
## Call:
## rpart(formula = Y ~ ., data = train, method = "class", parms = list(split = "gini"),
##     control = rpart.control(minsplit = 1, maxdepth = 1, usesurrogate = 1))
##   n= 500
##
##          CP nsplit rel error    xerror       xstd
## 1 0.7261411      0 1.0000000 1.0000000 0.04636138
## 2 0.0100000      1 0.2738589 0.2738589 0.03140616
```

```
##
## Variable importance
## X1 X2 X5
## 62 37  1
##
## Node number 1: 500 observations,    complexity param=0.7261411
##   predicted class=1  expected loss=0.482  P(node) =1
##     class counts:   241   259
##    probabilities: 0.482 0.518
##   left son=2 (243 obs) right son=3 (257 obs)
##   Primary splits:
##       X1 < 0.5 to the left,  improve=135.15930000, (0 missing)
##       X2 < 0.5 to the left,  improve= 52.78111000, (0 missing)
##       X3 < 0.5 to the left,  improve=  0.29823390, (0 missing)
##       X4 < 0.5 to the right, improve=  0.27751560, (0 missing)
##       X5 < 0.5 to the left,  improve=  0.05810746, (0 missing)
##   Surrogate splits:
##       X2 < 0.5 to the left,  agree=0.806, adj=0.601, (0 split)
##       X5 < 0.5 to the left,  agree=0.518, adj=0.008, (0 split)
##       X4 < 0.5 to the right, agree=0.516, adj=0.004, (0 split)
##
## Node number 2: 243 observations
##   predicted class=0  expected loss=0.1399177  P(node) =0.486
##     class counts:   209    34
##    probabilities: 0.860 0.140
##
## Node number 3: 257 observations
##   predicted class=1  expected loss=0.1245136  P(node) =0.514
##     class counts:    32   225
##    probabilities: 0.125 0.875
```

```r
rpart.plot(stump2)
```