Stat 542 HW 5

Yizhou Zhang

April 19, 2018

Q1a

The following codes define a stump function with three arguments `x, y, w`. The `x` and `y` are the training sets and the label, whiel `w` are the initial weights. In each iteration, the algorithm identifies the optimal cut point in terms of maximizing Gini Index, use the cut point to make predictions, calculate the weighted error rate, obtain the α, and updates the weights for next iteration. Three values are returned, `cutoff` which is the optimal cutoff, `left` that is the label for all `x < cutoff`, and `right` for the rest of points.

```
w = rep(1/length(x), length(x))
stump = function(x,y,w){

cut = quantile(x, probs = ((1:90)/100))
score = rep(0, length(cut))

for (i in seq_along(cut)){

#ge the left-node and right-node points
G_L = which(x <= cut[i])
G_R = which(x > cut[i])

#compute the probabilities for left and right nodes
P_L =  sum(w[G_L] * ifelse(y[G_L]==1, 1,0 ))/ (sum(w[G_L]))

P_R =  sum(w[G_R] * ifelse(y[G_R]==1, 1,0 )) /(sum(w[G_R]))

#compute the Gini Impurities
score[i] = -(sum(w[G_L])/sum(w))*(P_L*(1-P_L)) - (sum(w[G_R])/sum(w))*(P_R*(1-P_R))
}
cutoff  = cut[which(score == max(score))]
left = which(x <= cutoff)

wagl =mean(w[left] * y[left])
wagr = mean(w[-left] * y[-left])

f_L = sign(wagl)
f_R = sign(wagr)

return(list("cutoff"=cutoff, "left"= f_L,"right"= f_R))
}
```

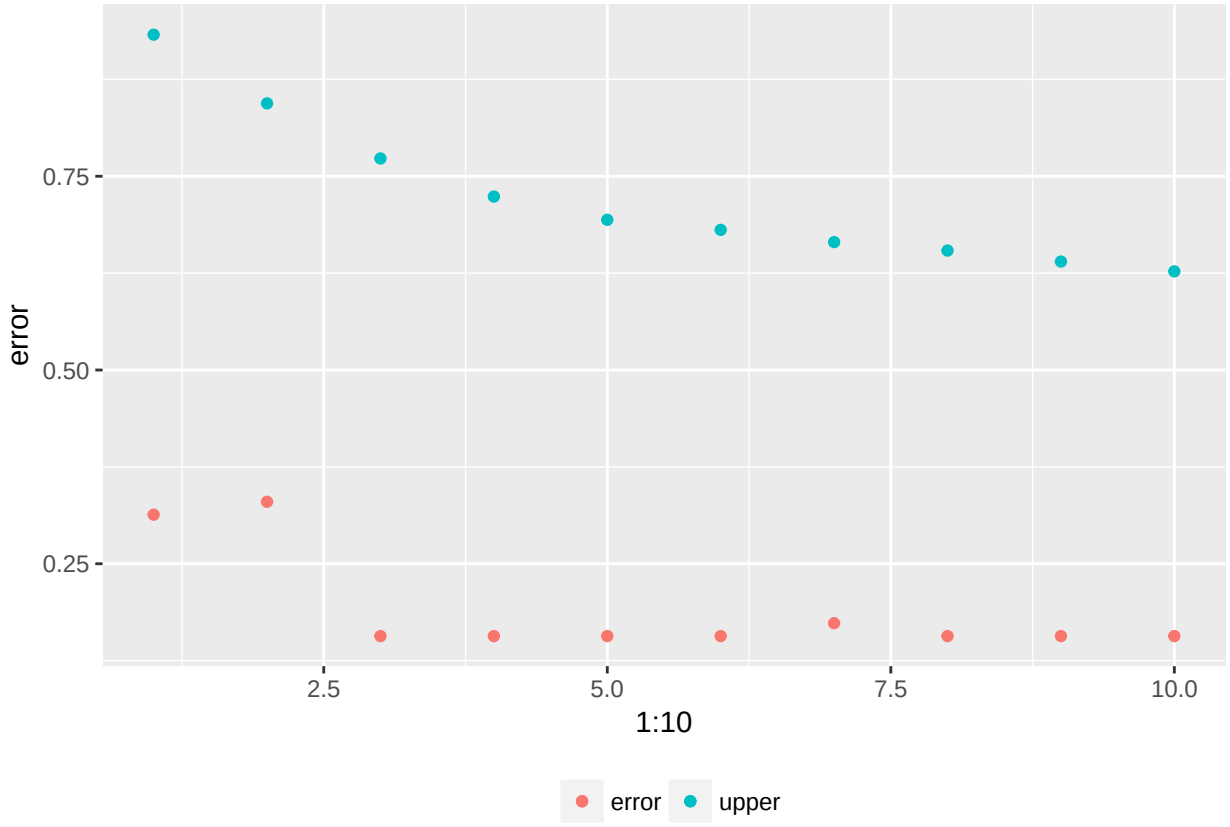
Q1b

The following codes defines an adaboost function.

```
ada = function(x,y,T){
  pred = matrix(nrow =length(x), ncol = T)
  w = rep(1/length(x), length(x))
```

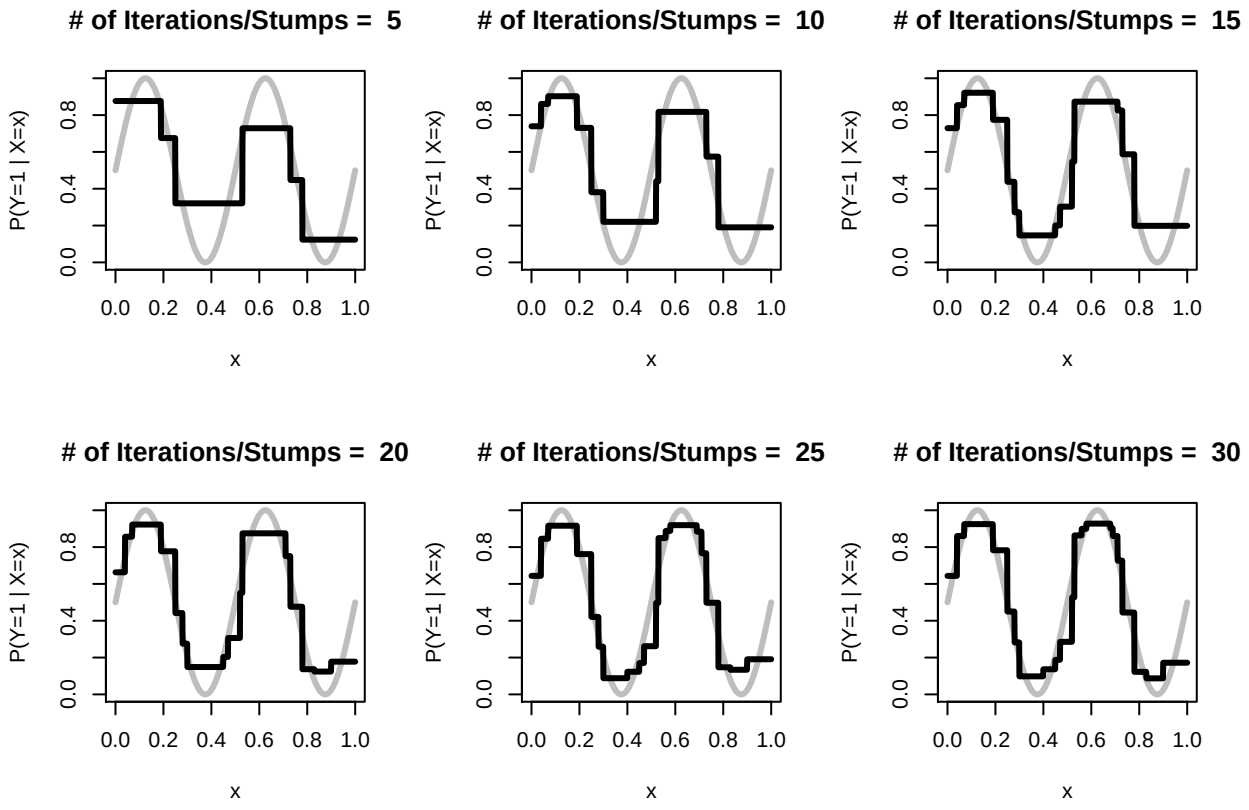
```
  epsilon = rep(0, T)
  alpha = rep(0,T)
  upper = rep(0,T)
  boundaries = rep(0,T)
  l_rule = rep(0,T)
  r_rule = rep(0,T)

  for (t in 1:T){
    a = stump(x,y,w)
    boundaries[t] =  a$cutoff
    l_rule[t] = a$left
    r_rule[t] = a$right

    pred[,t] = ifelse(x < a$cutoff, a$left, a$right)
    epsilon[t] = sum( w * (y != pred[,t]))
    alpha[t] = 0.5*log((1-epsilon[t])/epsilon[t])

    w =(w/sum(w))*exp((-1)*alpha[t]*y*pred[,t])
  }
  raw_prob = pred %*% alpha
  Fx = sign(pred %*% alpha)
  upper = exp(-2 *sum((0.5-epsilon)^2))

  return(list("final" = Fx, "matrix" = pred, "eps" = epsilon, "fitted"=raw_prob,
              "alpha" = alpha,"weight" =w, "upper" = upper,
              "cutoffs" = boundaries, "left" = l_rule, "right"= r_rule))
}
```

The plot below compares the error rates and the exponential upper bounds for stump numbers one to ten. It could be seen that as the number of stumps increases, both the error rates and the upper bound decreases at a decreasing speed.



The plot below further illustrates the underlying probabilities of the data generating process, with stump numbers from 5 to 30. The probability approximation is apparently increasing with more stumps.

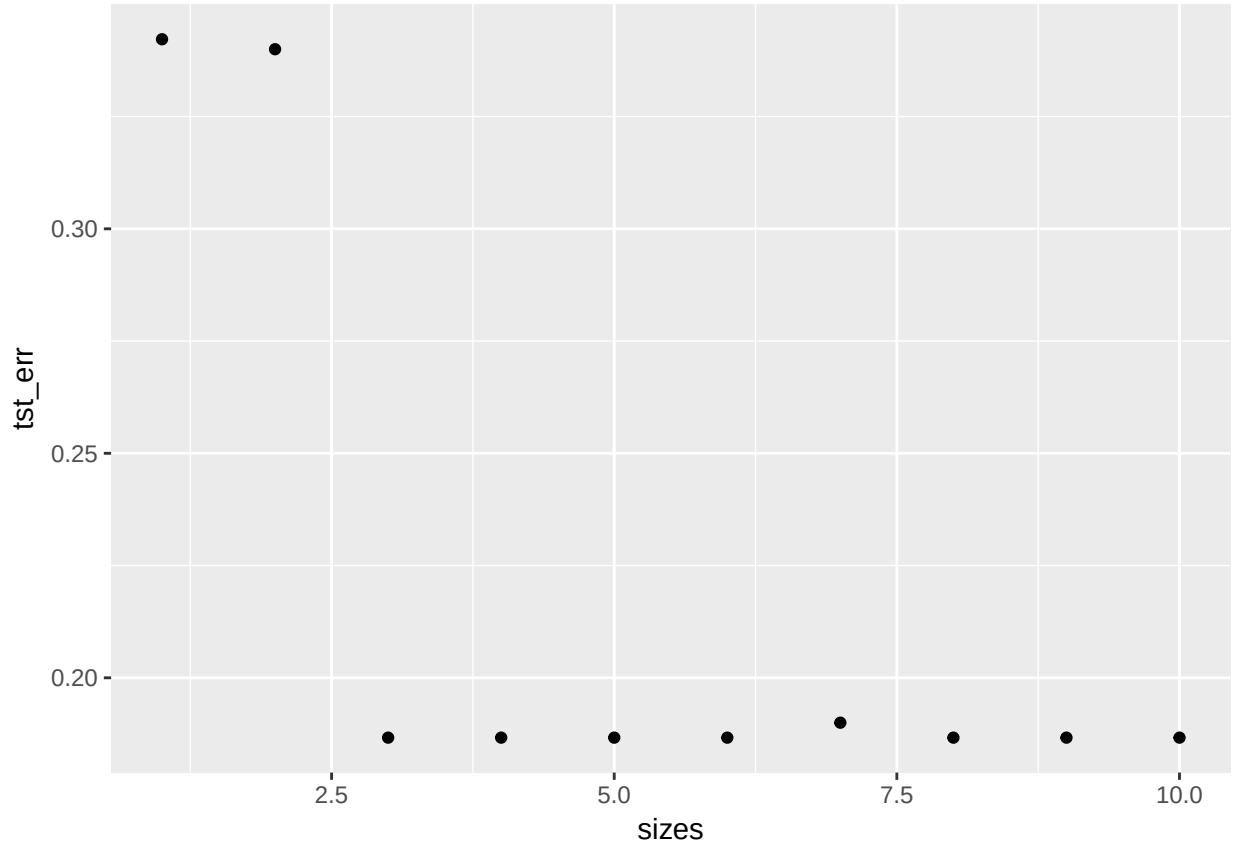
```
#get probability
x = seq(0, 1, 0.001)
y = (rbinom(length(x),1,(sin(4*pi*x)+1)/2) -0.5)*2

size = (1:6)*5
Fx = matrix( nrow = length(x), ncol = length(size))

for (i in seq_along(size)){

b = ada(x,y,size[i])
Fx[,i] = b$fitted
}

par(mfrow =c(2,3))

for (i in seq_along(size)){
    plot(x, (sin(4*pi*x)+1)/2, type = "l", lwd = 3, ylab = "P(Y=1 | X=x)", col = "gray")
    lines(x, 1/(1+exp(-2*Fx[,i])), lwd = 3)
    title(paste("# of Iterations/Stumps = ", size[i]))
}
```



I further apply the method on a test set to investigate overfitting. The codes below defines a function to fit a trained model on a set of test sets. First fit the training set to `ada` model with `T` iterations. Record the T cutoff used by each of the T stumps $f_t(x)$, and the `alpha` vector that contains the weights of the stumps. Then fit each point x_i of the test set to each stump $f_t(x)$, get the fitted values, and get the weighted sum of the fitted values $F(x_i) = \sum_t \alpha_t f_t(x_i)$. The last step compares $F(x_i)$ to the actual value y_i.

```
set.seed(6133339)
x_tst = runif(900)
y_tst = (rbinom(900,1,(sin(4*pi*x_tst)+1)/2) -0.5)*2


pred_tst = function(x_tst, y_tst, T){

pred = matrix(nrow = length(x_tst), ncol = T)

a = ada(x_tr, y_tr, T)
alpha= a$alpha

for (i in 1:T){
pred[,i]= ifelse(x_tst<a$cutoffs[i], a$left[i], a$right[i])
}

last = sign(pred %*% alpha)
err_tst = length(which(y_tst != last))/length(y_tst)

return(err_tst)
}
```

```r
#Plot the errors
sizes = 1:10
tst_err = rep(0, length(sizes))

for (i in seq_along(sizes)){
tst_err[i] = pred_tst(x_tst, y_tst, sizes[i])
}

ggplot()+
geom_point(aes(sizes,tst_err))
```



The above plot of testing error is very similar to the plot of training errors: The error rates drops significantly after 2 stumps and and stablizes at aroulnd `0.19`. Recall the probability plot above - after a few stumps the underlying probability of the data generating process is almost perfectly replicated. In other words, the error rates of the training data mainly comes from randomness, thus cannot be further reduced. The same logic applies to the testing set, because the testing set is just an extension of the training set. Given the specific functional form of probabilities, the testing set is just repeating the `sin(x)` cycles with a different length. The testing errors, similar to the training errors, result from randomness in the dat generation process and cannot be further reduced.Adding more and more stumps just makes the replication of underlying probabilities better and better, but cannot eliminate randomness. If this logic is correct, then there is no overfitting.

Q2 Prove the Sherman-Morrison Formula

$$
\begin{aligned}
&(A - bb^T)[A^{-1} + \frac{A^{-1}bb^TA^{-1}}{1 - b^TA^{-1}b}] \\
&= (A - bb^T)A^{-1} + \frac{(A - bb^T)A^{-1}bb^TA^{-1}}{1 - b^TA^{-1}b} \\
&= AA^{-1} - bb^TA^{-1} + \frac{AA^{-1}bb^TA^{-1} - bb^TA^{-1}bb^TA^{-1}}{1 - b^TA^{-1}b} \\
&= I - bb^TA^{-1} + \frac{bb^TA^{-1} - bb^TA^{-1}bb^TA^{-1}}{1 - b^TA^{-1}b} \\
&= I - bb^TA^{-1} + \frac{b(1 - b^TA^{-1}b)b^TA^{-1}}{1 - b^TA^{-1}b} \\
&= I - bb^TA^{-1} + bb^TA^{-1} \\
&= I \\
&thus,\ (A - bb^T)^{-1} = A^{-1} + \frac{A^{-1}bb^TA^{-1}}{1 - b^TA^{-1}b}
\end{aligned}
$$

Q3 Take the Tate Collection dataset from Kaggle.

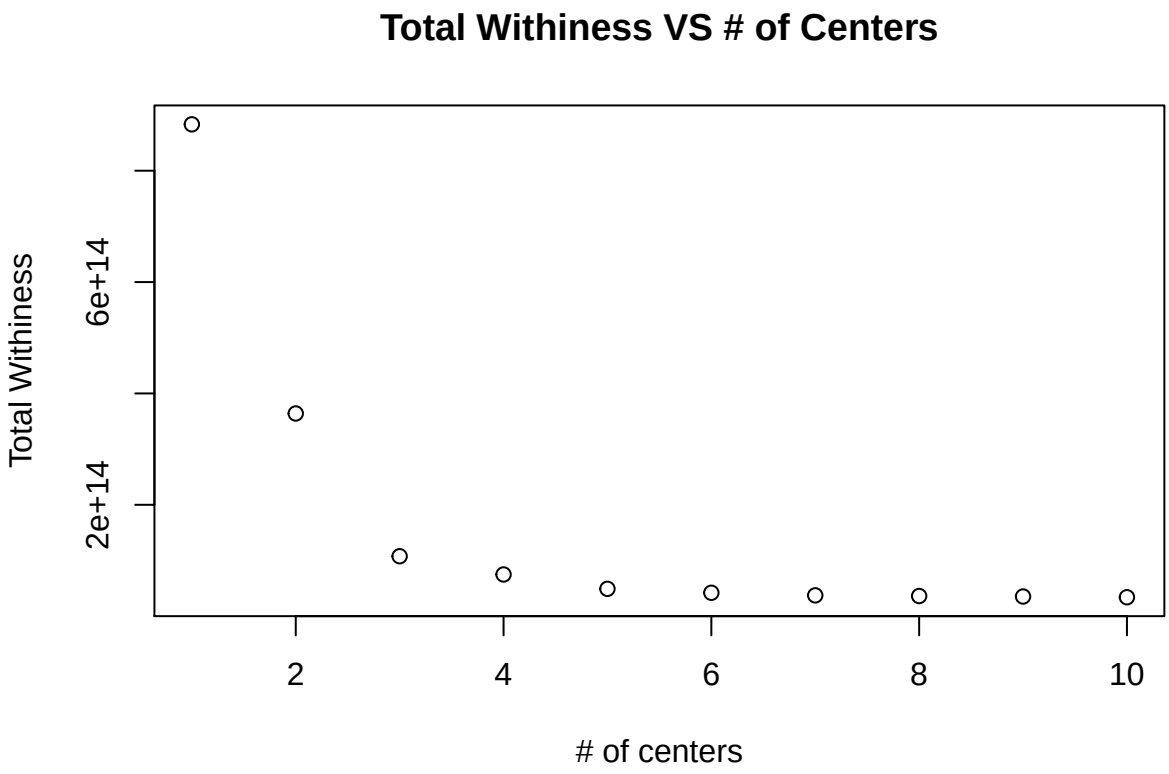
a. What is your goal when performing this clustering?

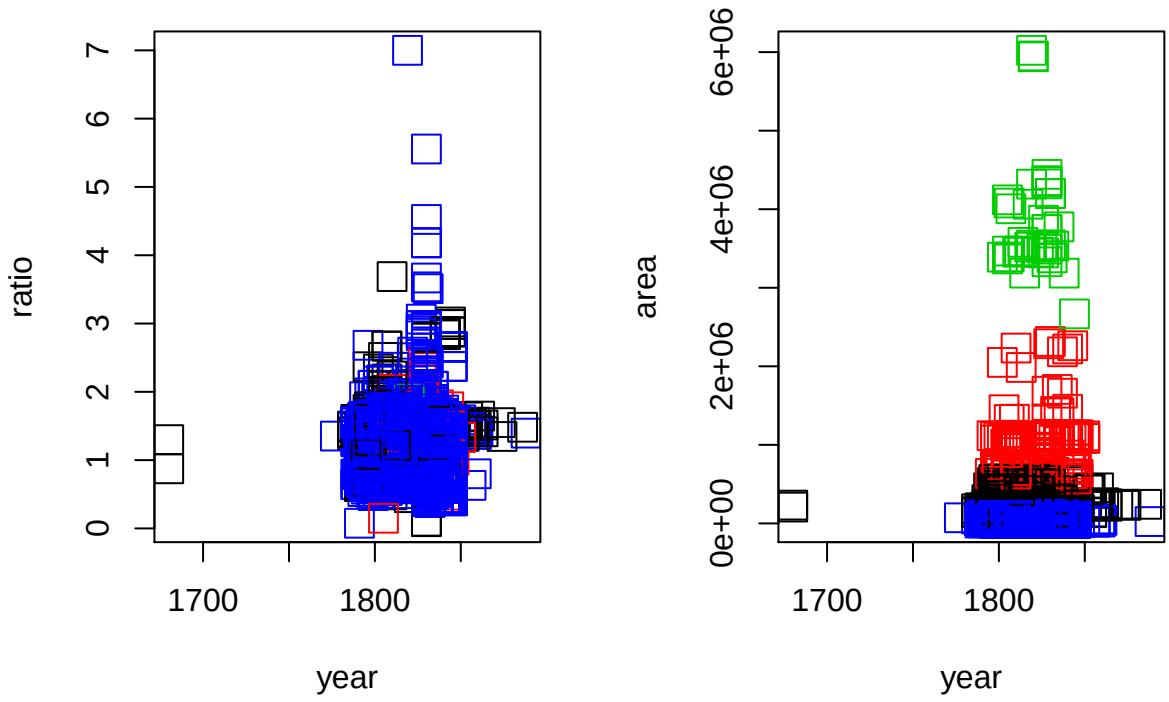
The clustering analysis will focus on the work of one artist: Turner, Joseph Mallord William. Tate Britain has the largest collection of Turner's work at 39389 pieces. In other words, Turner is the most represented artist and his work composed more than half of all Tate's collections.

The goal is to investigate Turner's preferences over paint dimensions over time. I want to investigate one question: can we infer the creation year of a piece, given information of only its dimensions ? For this purpose, I am using three variables `year`, `ratio = height/width` and `area = height * width`.

The first clustering method is K-means. Specifically, I applied the method with `1:10` centers and select the optimal center by the `tot.withinss` criteria. The plot below shows that 4 centers might be suitable, given the sharp decrease of total withiness at n=4.

```
distance = rep(0,10)
for (i in seq_along(distance)){
kmean = kmeans(Turner[,c("year","ratio","area")], centers = i, nstart = 10)
distance[i] = kmean$tot.withinss
}
```


Total Withiness VS # of Centers
Total Withiness
2e+14
6e+14
2
4
6
8
10
# of centers



The two 2-d plots above and the 3-d plot below all display the original data with clustering labels. The results indicate that `area` is the most important variables that dominates the clustering scheme. There are four groups with respective characteristics:

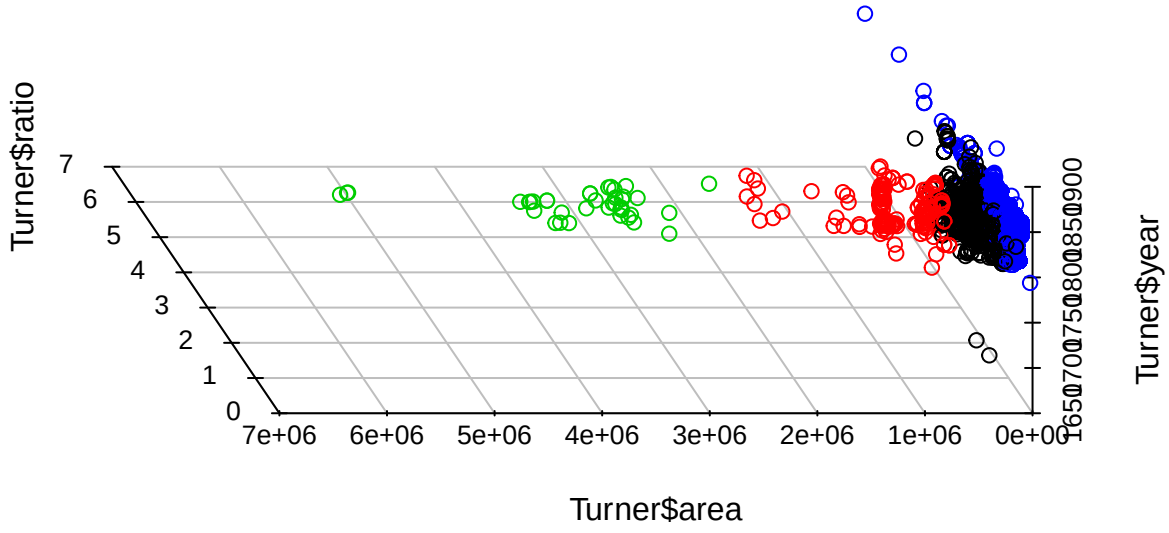
1. "The small Portraits": group 4 is the smallest in dimension with areas between 2023 and 116280. Its average ratio is 1.1443441, featuring portrait more than landscapes. The year span is 1777, 1888, covering roughly his whole career. It also includes some of the "tallest" pieces, with the height-width ratio greater than five.

2. "The normal small portraits".group 3 is also small in dimension with areas between 2667099 and 6022800. Its average ratio is 1.5353362, featuring portrait more than landscapes. The year span is 1802, 1844, almost all of Turner's career.

3. "The big portraits".Gropu 1 has big dimensions with `area` between 116518 and 562200. The average ratio of this group is 1.2953973, meaning that its orientation is more portrait than landscape. Its time span is 1680, 1886, narrower than that of the first two groups.

4. "The Mega portraits".Group 2 hsa the biggest dimension with areas between 564573 and 2316160. Its average ratio is 1.3779895, featuring portrait more than landscapes. Its time span is 1795, 1850, which concentrates on the middle phase of Turner's career. This implies that Turner painted his biggest pieces during the middle of his carrer,when he was prime in creativity and strength to handle bigger pieces.

To summarize, Turner's work features more portrait than landscape. Regarding the previously raised question, the dimensions of his work is not very informative of the piece's creation year. However, very large paints are usually painted in the middle years of Turner's career bewteen 1795, 1850.



Perform Model-based Clustering.

An alternative is model-based clustering, which consider the data as coming from a distribution that is mixture of two or more clusters. Unlike k-means, the model-based clustering uses a soft assignment, where each data point has nonzero probabilities of belonging to one cluster.

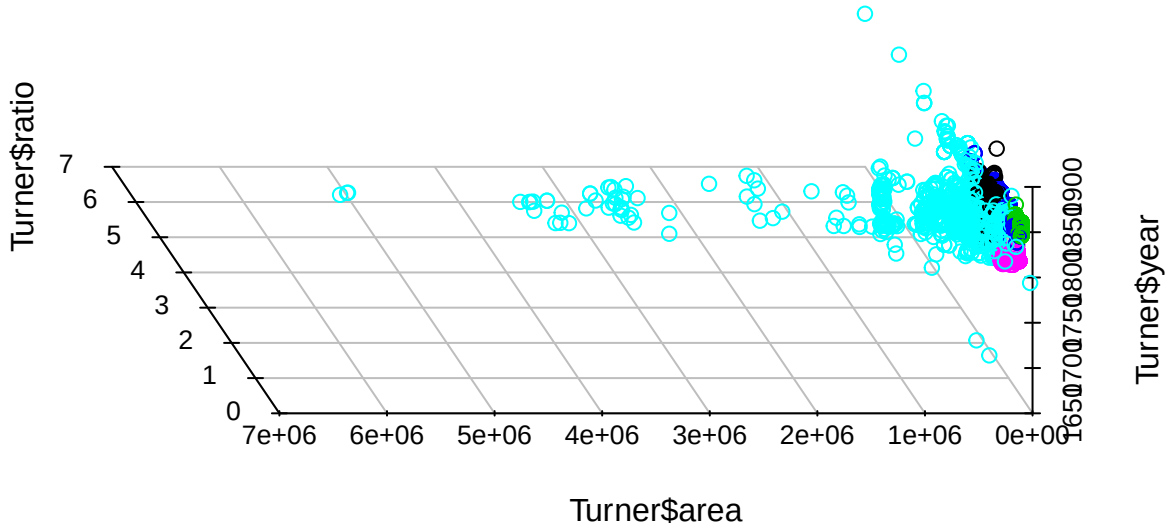
The advantage of model-based clustering is that the modeler does not have to pre-specify the number of clusters. The algorithm uses BIC to determine the optimal cluster numbers as well as the covariacne strucutres. The disadvantage of model-based clustering, however, is that the optimal clustering scheme might not be intuitive and be difficult to interpret. Additionally, model-based clustering cannot identify noise points because soft classification assigns each point to a class with the highest relative probability. Thirdly, by assuming gaussian underlying distribution, the clusters all assume circular or eplisodal shapes.

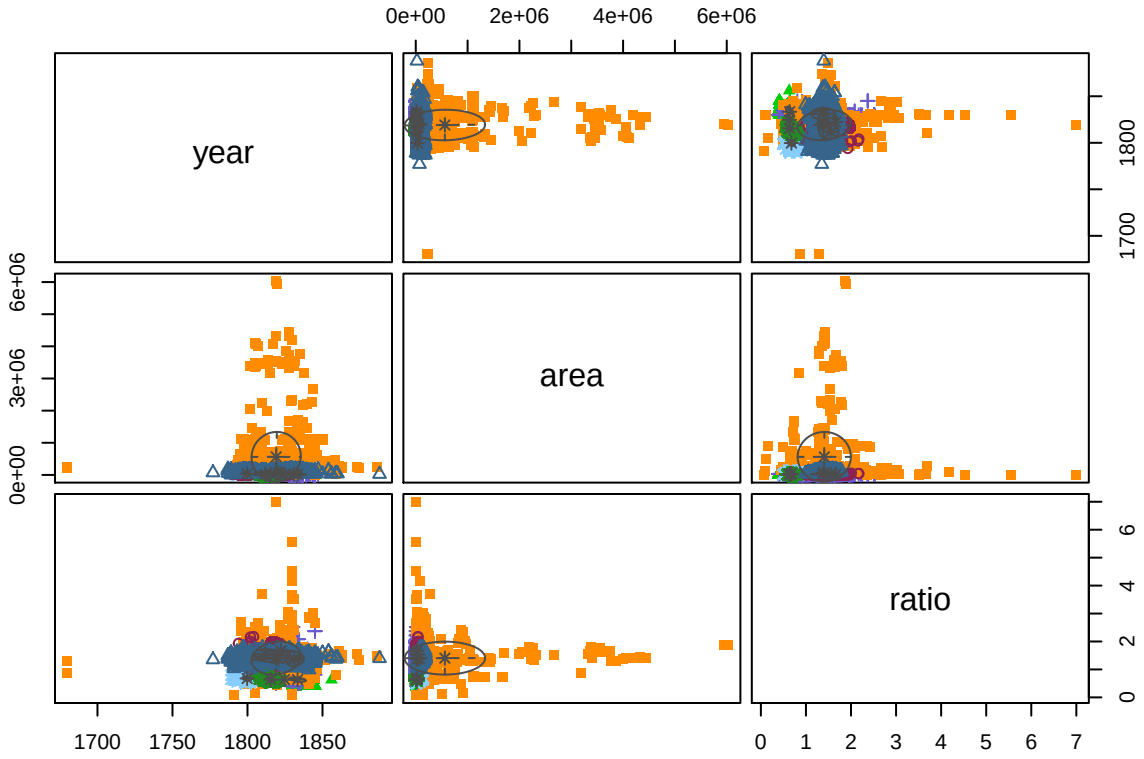
```
# Model Based Clustering
library(mclust)
fit = Mclust(Turner[,c("year","area", "ratio")])
summary(fit) # display the best model
```

```
## ----------------------------------------------------
## Gaussian finite mixture model fitted by EM algorithm
## ----------------------------------------------------
##
## Mclust VVV (ellipsoidal, varying volume, shape, and orientation) model with 9 components:
##
##  log.likelihood     n df      BIC      ICL
##         -543550 38227 89 -1088039 -1093090
```

```
##
## Clustering table:
##    1    2    3    4    5    6    7    8    9
## 5878 1489 7208 5837  914 4154 5686 2686 4375
```

The model-based clustering approach has selected a model with 9 components (i.e. clusters). The optimal selected model name is VVV model. That means each components has different ellipsoidal with varying volume, shape, and orientation. The two plots below display the classes in a 3-d plot and in a series of 2-d plots. The clustering scheme is not very intuitive with the classes heavily mingled with each other. And one class is imbalancedly represented, masking the others.Therefore,the plots provide an example of the major pros and cons of the model -based clustering approach: although the number of clusters is determined by algorithm rather than by pre-specification , interpretation could be difficult.




year
area
ratio
0e+00
2e+06
4e+06
6e+06
1700
1800
0e+00
3e+06
6e+06
0
2
4
6
1700
1750
1800
1850
0
1
2
3
4
5
6
7