Exam spring 2020

a

a.1

The best value for λ for lasso is 0.01831564 and for ridge 0.01831564.
The role of the penalty parameter λ in a lasso model is to control shrinkage. The penalty parameter will shrink coefficients towards 0 and some to 0. We want a λ that minimize the cross validated mean squared error.
When λ is high, variables that is not correlated with the response is highly penalised. When λ increases, bias increases and variance decrease. High λ gives more shrinkage.
When there is a larger number of parameters than observations we can use lasso, this is what we have in this task.

a.2

Mean squared error is the same as misclassification error for AML and ALL, I get a misclassification error of 0 for lasso test, lasso train and ridge test, and 0.04166667 for ridge train.
Because we make the model based on the training data, the model will follow the training data better than the test data. The test data can be different and thus not follow the same pattern as the training data, the error will therefore be larger on the test data. We can say that the model is biased to the training data.

a.3

The deterministic procedure means that we need to use leave-one-out cross-validation (LOOCV), since LOOCV is the only deterministic procedure.
It is a good choice with LOOCV, since our dataset is relatively small. With k-folds I make sure that my model gets sufficiently trained. LOOCV is approximately unbiased but have a high variance. The number of folds is a bias-variance trade-off. The advantage of having a larger k is that the model gets less bias, the disadvantage is more variance and higher computational time. The advantage of having a smaller k is that the model get less bias, the disadvantage is more variance.
We use cross-validation for comparing models. If our dataset would have been larger I would have chosen 5- or 10-folds.
Why LOOCV is deterministic:
It is no randomness in the training/validation set splits. LOOCV will always give the same predictions when given the same dataset. This is because LOOCV does the same calculation every time. A single observation is used for the validation and the rest is in the training set, and we do this with every observation. There is only one way of dividing in k-folds.

b

b.1

The lasso and ridge models have different probability for the same patient, when we create a lift plot the probabilities is ordered with the highest probability first (to the left). This means that different probabilities can be ordered the same in a lift plot. An example is a y that is predicted under 0.5 in lasso and over 0.5 in ridge, but since the lift plot only depends on the true values y, their curve in lift plot is the same. The order of the probabilities of lasso and ridge doesn't need to be the same, because points with the same true value can swap places and the lift curve will still be the same.
Since the lift curves of lasso and ridge are equal it tells us that the one misclassification in ridge model is almost correct, i.e. close to 0.5.

b.2

The points (0.25, 3) is the ideal lift, this tells us that from point (0.00, 3) to (0.25, 3) every prediction of whether a person have AML or ALL has been correct to this point. The point (0.25, 3) also tells us that 1/4 of the highest probabilities have a 3 times higher proportion of the number 1 than the whole set. Number 1 equals having ALL.

c

c.1

The variable names of the 9 genes found in step 1 is, genes from lowest to highest p-value of the 9 best: 1882, 6854, 6041, 3252, 4847, 2121, 758, 1745, 4377.
Because we have more explanatory variables than observations, step 1 is necessary before we implement step 3. We need fewer explanatory variables when fitting the data.
The misclassification error computed in step 4 on the testset is 0.

c.2

The procedure is wrong since we are using the test data in calculating the prediction of the testset. We need to remove the test data when we use two sample t-test, this means we can use the training set. When we include testset data in the predictions, the model will fit the testset more than when we exclude them. It is very important to not use the testset when training, so our results from step 4 can not be compared to a model who have not used part of the testset when training, like lasso and ridge logistic regression at point a.2.

c.3

The variable names of the correctly selected 9 genes, genes is from lowest to highest p-value of the 9 best:
1882 6854 4847 6623 2121 4973 6041 3252 1745
The correct estimate of the misclassification error:
0.08333333 in test and 0 in training.

d

d.1

The misclassification error for all 10 different results:

ALL, ALL.10	0.3333333
ALL, ALL.11	0.2916667
ALL10, ALL.11	0.3472222
ALL, AML	0.01388889
ALL.10, AML	0.01388889
ALL.11, AML	0.01388889
ALL, AML.2	0.2777778
ALL.10, AML.2	0.2777778
ALL.11, AML.2	0.01388889
AML, AML.2	0.01388889

I get 5 different errors, with 0.0139 the most times. The error is small.
The results are so different because K-means find a local optimum, not a global optimum. Then it is important for the result where the algorithm starts (different initialization points for the centroids), and therefore it is important to run k-means many times with different starts.
I am using logistic regression because k-means are making clusters that don't overlap, which makes them separable. Logistic regression will therefore predict the data-points.

d.2

Since we have two classes, AML and ALL, there is not a problem choosing K, since it must be two. It is in general problematic since we normally don't know how many classes we have, and therefore need to "guess" K to find out how many clusters we need.

It is not possible to compute its best values by cross-validation because CV will set K=n. We can use the variation to find the total distance with-in clusters, this gets smaller and smaller with more clusters so it is difficult to know when to stop. Higher K gives less error, since CV minimise the error K will be set to n. when CV set K=n we have n clusters and zero error, but this is obviously not correct.

We can find how many classes it is in response-variables, if we don't know how many classes it is a strategy to select K is using the elbow method. Using the elbow method we calculate the the squared error for each patient (point), the squared error is the square of the distance between a point and the predicted cluster center for that point. Then we use the sum of all squared error for the point and calculate it for different values of K. We plot the result against K. In the beginning it is a steep curve, this curve will flatten out and we will not get much better result for increasing K. We are looking for the "elbow"/kink in the plot, where increasing K only slightly improve the score and decreasing k drastically decrease the score.

e

e.1

See figure 1 for dendrogram and figure 2 for dendrogram with colors for ALL and AML. It doesn't look like it is good clustering, this suggest this isn't the best method. From the dendrogram it looks like ALL.40 is an outlier. If we don't consider ALL.40 we can divide in two cluster, since ALL.40 is in its own cluster. The left cluster is ALL except two AML. The cluster to the right is about 50/50 ALL and AML. If we choose to divide into 5 clusters, the groups is more correct, we then have a cluster with ALL.40, a cluster with only two misclassifications, a cluster with only one misclassification, another cluster with only one misclassification and a cluster with three misclassifications. We need to use 5 clusters to separate ALL and AML patients properly. Since we know the truth, the result is not great. We need to use 5 clusters to separate such that there is just a few misclassifications.

e.2

See figure 3. From the plot we can see that the first principal components can explain almost 10 % of the variance. We can see that we can remove a couple of principal component and still have a high accuracy and then get a less complex model.
I have chosen to use the function fviz_eig to provide the plot, that plot the variance against the number of dimensions.

e.3

See figure 4 and 5.
Looking at the distances it seems reasonable to divide into 3 clusters. Comparing the distance with the dendrogram from e.1, the distance is higher when the clusters merge in this dendrogram. The 3 clusters only have a few misclassifications, one with five misclassifications, one with two misclassifications and one with four misclassifications. Since we can divide into fewer clusters when using

only the first two principal components, it suggests those contain the most important information. The reason that the result in e.1 is worse might be that we are using noise from the data. It looks good, it is not a big difference between where the middle and right cluster merge and the left with them. If the left and middle had merged it would have been a good result for 2 clusters. A reason for the misclassifications may be that there are differences in a type of leukemia or it may be some error in classification when collecting the data.

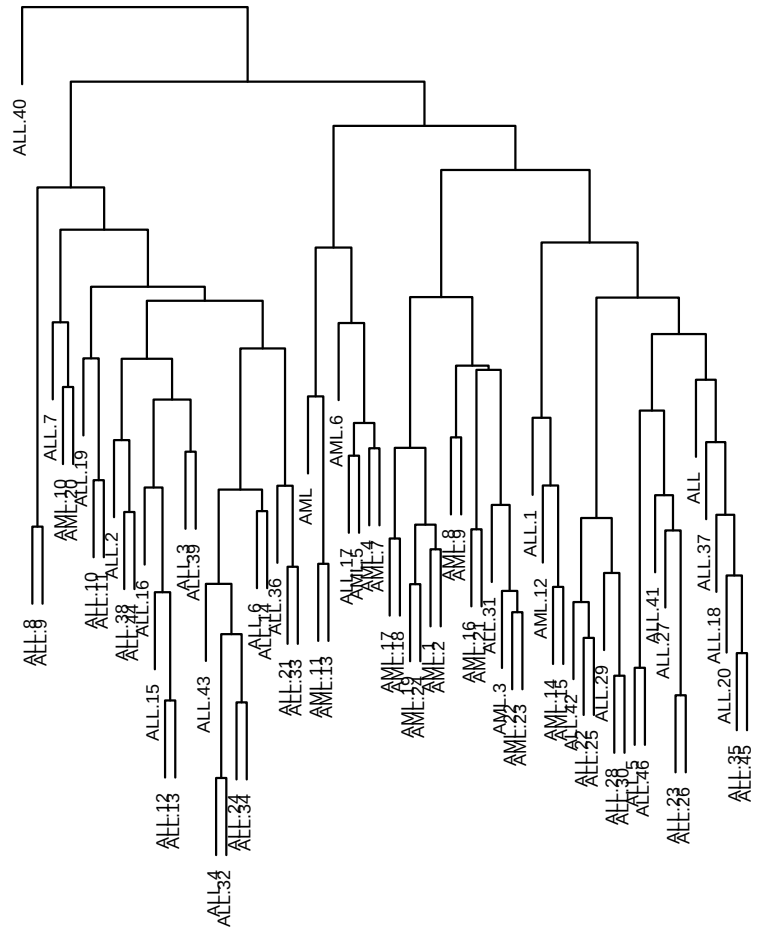

Figure 1: dendrogram for e.1

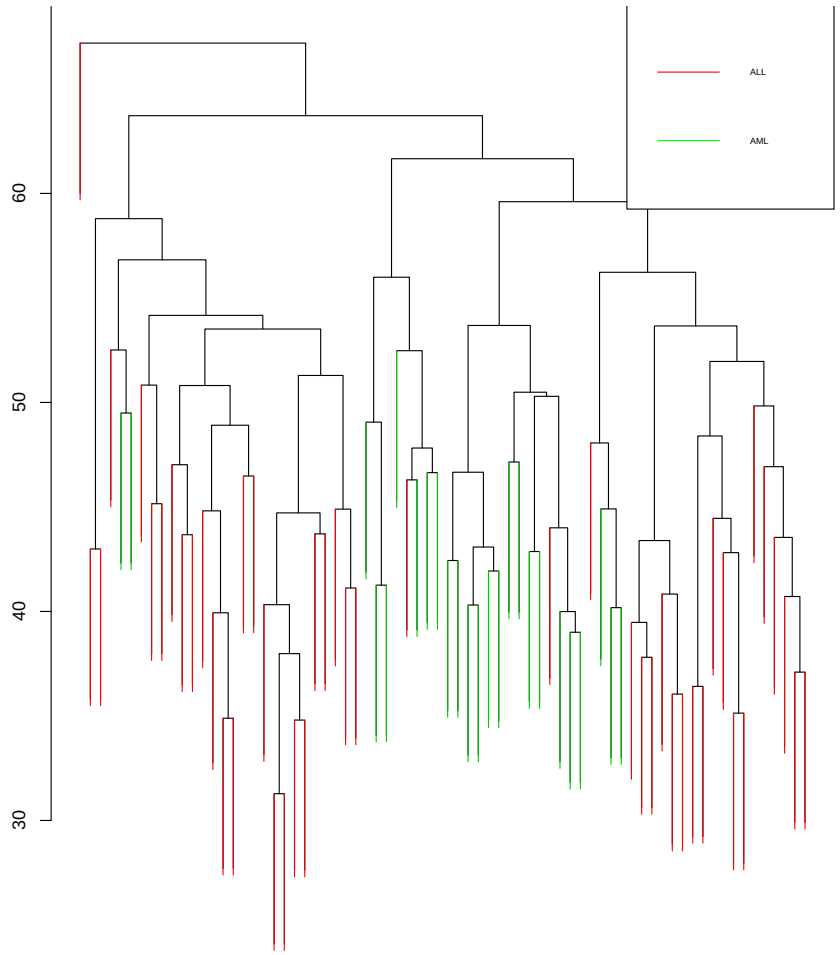

Figure 2: dendrogram for e.1

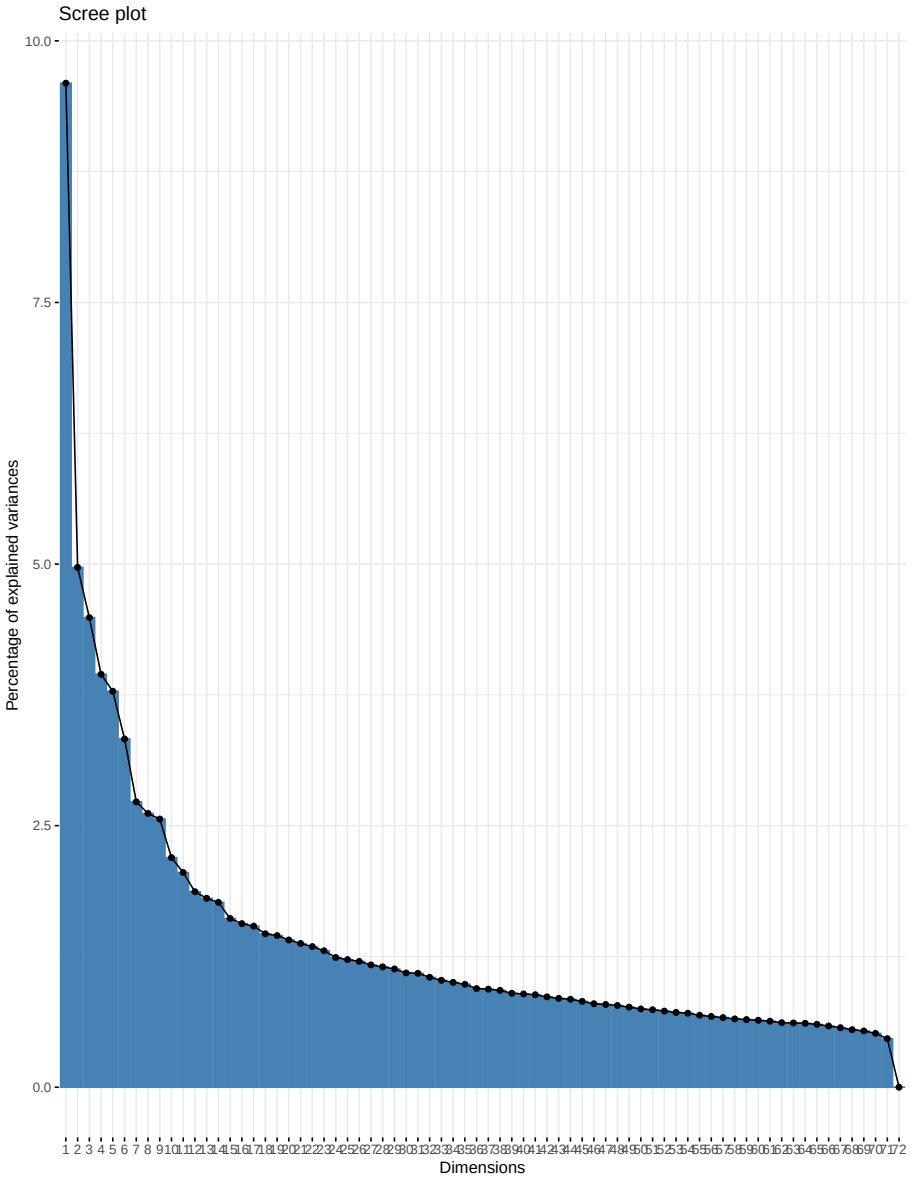

Figure 3: percentage of the original variance

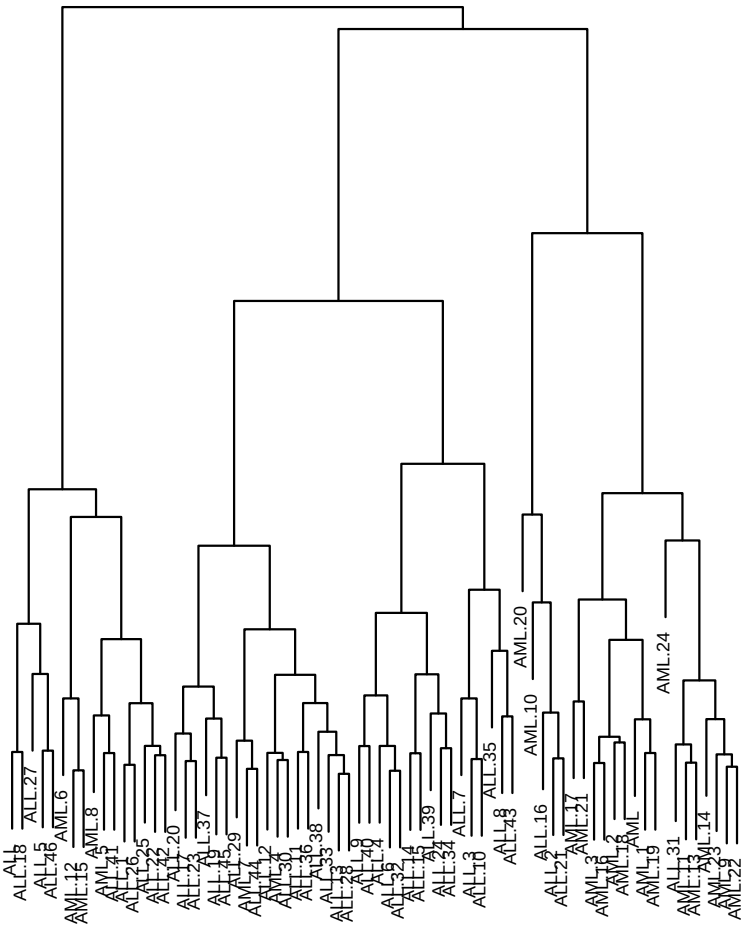

Figure 4: dendrogram for e.3

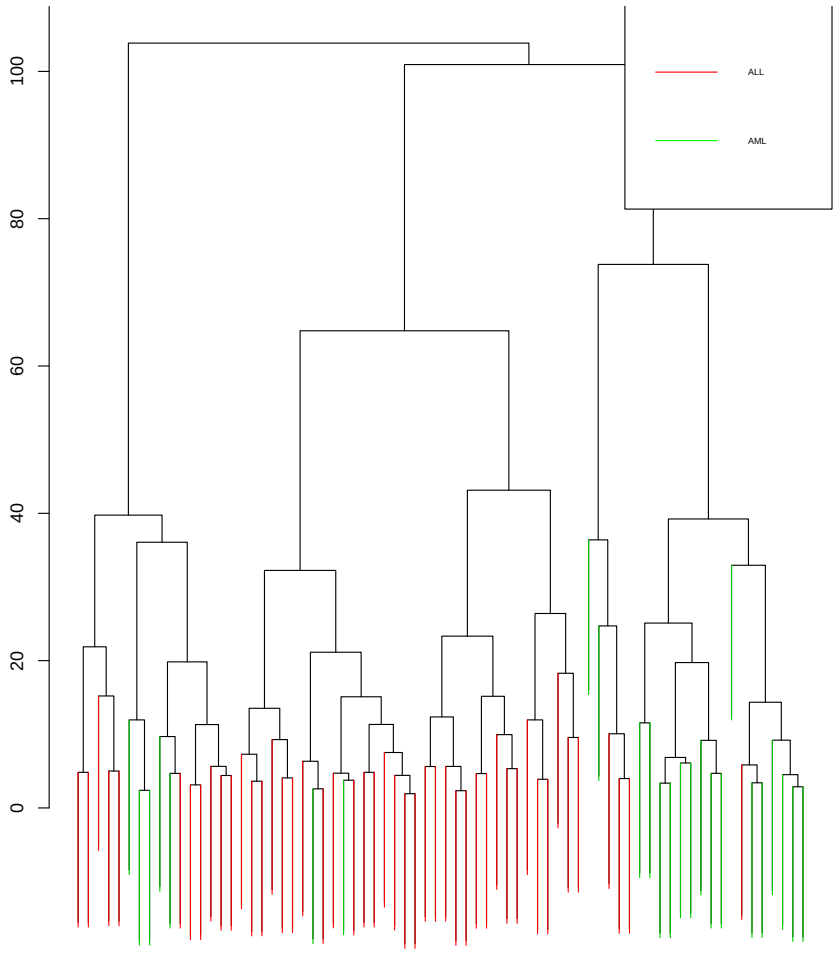

Figure 5: dendrogram for e.3

Code and output for all task

R code .1: Output

```
> data = read.csv('leukemia_big', header = TRUE)
> datat = t(data) #trspose since rows is observations
>
> set.seed(2100)
>
> testnames = c("ALL.4", "ALL.8", "ALL.10", "ALL.11", "ALL.13", "ALL.18", "AML", "AML.1",
+   "AML.4", "AML.6", "AML.8", "ALL.23", "ALL.26", "ALL.29", "ALL.31", "ALL.32", "ALL.35",
+   "ALL.39", "ALL.40", "ALL.41", "ALL.42", "AML.16", "AML.22", "AML.24")
>
> test_true = is.element(rownames(datat), testnames)
> test.data = datat[test_true,]
> train.data = datat[!test_true,]
>
> data.type = grepl("ALL", rownames(datat))
> test.y = data.type[test_true]
> train.y = data.type[!test_true]
> #a.1
> library(glmnet)
> lambdas = exp(-7:2)
> lasso =  cv.glmnet(x = train.data, y = train.y,  nfolds = 48, alpha = 1, standardize = TRUE, family = "binomial", lambda = lambdas, grouped = FALSE)
> lasso$lambda.min
[1] 0.01831564
>
> mod.ridge = cv.glmnet(x = train.data, y = train.y, nfolds = 48, alpha = 0, standardize = TRUE, family = "binomial", lambda = lambdas, grouped = FALSE)
> mod.ridge$lambda.min
[1] 0.01831564
>
> #a.2
> mse.logistic.cv = function(cv.mod, x, y) {
+   y.probs = predict(cv.mod$glmnet.fit, s=cv.mod$lambda.min, newx=x, type="response")
+   y.hat = y.probs >= 0.5
+   return(mean((y - y.hat)^2))
+ }
>
```

```
> cbind(mse.logistic.cv(lasso, train.data,train.y),
    mse.logistic.cv(lasso, test.data, test.y),
    mse.logistic.cv(mod.ridge, train.data,train.y),
    mse.logistic.cv(mod.ridge, test.data, test.y))
      [,1] [,2] [,3]       [,4]
[1,]    0    0    0 0.04166667
>
>
> #c
> ALL = grepl("ALL", rownames((datat)))
> AML = grepl("AML", rownames((datat)))
> p.values <- vector(mode='numeric', length=ncol(datat))
> for (i in (1:ncol(datat))) {
+   t.test.result = t.test(datat[ALL, i], datat[AML, i])
+   p.values[i] = t.test.result$p.value
+ }
> best.p.values = order(p.values)[1:9]
> p.values[best.p.values]
[1] 4.640163e-19 1.916666e-17 6.135201e-17 1.295706e-16
    1.331165e-16 1.040298e-14 2.710576e-14
[8] 3.836943e-14 1.104672e-13
>
> test.c = test.data[, best.p.values]
> train.c = train.data[, best.p.values]
> log.model = glm(train.y ~ ., data =
    as.data.frame(train.c), family = 'binomial')
Warning messages:
1: glm.fit: algorithm did not converge
2: glm.fit: fitted probabilities numerically 0 or 1 occurred
>
> log.predict.train = predict(log.model, newdata =
    as.data.frame(train.c),  type="response")
> log.hat.train = log.predict.train >= 0.5
> error.logistic.train = mean((train.y - log.hat.train)^2)
>
> log.predict.test = predict(log.model, newdata =
    as.data.frame(test.c),  type="response")
> log.hat.test = log.predict.test >= 0.5
> error.logistic.test = mean((test.y - log.hat.test)^2)
>
> cbind(error.logistic.train, error.logistic.test)
     error.logistic.train error.logistic.test
[1,]                    0                   0
>
>
> #c.3
> ALL2 = grepl("ALL", rownames((train.data)))
> AML2 = grepl("AML", rownames((train.data)))
> p.values2 <- vector(mode='numeric',
    length=ncol(train.data))
```

```
> for (i in (1:ncol(train.data))) {
+   t.test.result = t.test(train.data[ALL2, i],
    train.data[AML2, i])
+   p.values2[i] = t.test.result$p.value
+ }
> best.p.values2 = order(p.values2)[1:9]
> best.p.values2
[1] 1882 6854 4847 6623 2121 4973 6041 3252 1745
> p.values2[best.p.values2]
[1] 1.657566e-14 3.758817e-11 1.527052e-10 3.322114e-10
    3.661996e-10 1.039764e-09 1.207423e-09
[8] 1.365703e-09 1.863012e-09
>
> test.c2 = test.data[, best.p.values2]
> train.c2 = train.data[, best.p.values2]
> log.model2 = glm(train.y ~ ., data =
    as.data.frame(train.c2), family = 'binomial')
Warning messages:
1: glm.fit: algorithm did not converge
2: glm.fit: fitted probabilities numerically 0 or 1 occurred
>
> log.predict.train2 = predict(log.model2, newdata =
    as.data.frame(train.c2),  type="response")
> log.hat.train2 = log.predict.train2 >= 0.5
> error.logistic.train2 = mean((train.y - log.hat.train2)^2)
>
> log.predict.test2 = predict(log.model2, newdata =
    as.data.frame(test.c2),  type="response")
> log.hat.test2 = log.predict.test2 >= 0.5
> error.logistic.test2 = mean((test.y - log.hat.test2)^2)
>
> cbind(error.logistic.train2, error.logistic.test2)
     error.logistic.train2 error.logistic.test2
[1,]                     0           0.08333333
>
>
> #d
> init.observation = c("ALL", "ALL.10", "ALL.11", "AML",
    "AML.2")
> init = which(is.element(rownames(datat),
    init.observation))
> for (i in 2:length(init)) {
+   for (j in 1:(i-1)) {
+     place = init[c(i, j)]
+     k1 = kmeans(datat, datat[place,])
+     mod.k1 = glm(ALL ~ k1$cluster, family = 'binomial')
+     print(i)
+     print(j)
+     print(mean((ALL - (mod.k1$fitted.values >= 0.5))^2))
+   }
```

```
+ }
[1] 2
[1] 1
[1] 0.3333333
[1] 3
[1] 1
[1] 0.2916667
[1] 3
[1] 2
[1] 0.3472222
[1] 4
[1] 1
[1] 0.01388889
[1] 4
[1] 2
[1] 0.01388889
[1] 4
[1] 3
[1] 0.01388889
[1] 5
[1] 1
[1] 0.2777778
[1] 5
[1] 2
[1] 0.2777778
[1] 5
[1] 3
[1] 0.01388889
[1] 5
[1] 4
[1] 0.01388889
>
> #e
> d = dist(datat)
> h1 = hclust(d, method = "complete")
> plot(h1, xlab = "", ylab = "", axes = FALSE, main = "",
    sub = "", lwd = 2)
> #plot(h1, cex = 0.6, hang = -1)
> #rect.hclust(h1, k = 5, border = 2:6)
>
>
> pc = prcomp(datat, center = TRUE, scale = TRUE)
> d2 = dist(pc$x[,1:2])
> h2 = hclust(d2, method = "complete")
> plot(h2, xlab = "", ylab = "", axes = FALSE, main = "",
    sub = "", lwd = 2)
> plot(h2, cex = 0.6, hang = -1)
> #rect.hclust(h2, k = 2, border = 2:4)
>
> library(factoextra)
```

```
>
> fviz_eig(pc, ncp = 72)
> colDen <- ifelse(data.type, 1, 2)
> ColorDendrogram(h1, colDen, main = "", branchlength =
    7.5, sub="cluster␣e.1",ylab = "")
> legend("topright", legend=c("ALL", "AML"),
+        col=c("red", "green"), lty=1, cex=0.5)
> ColorDendrogram(h2, colDen, main = "", branchlength = 21,
    sub="cluste␣e.3r",ylab = "")
> legend("topright", legend=c("ALL", "AML"),
+        col=c("red", "green"), lty=1, cex=0.5)
```