2. Compute training and test error of all four versions of the algorithm.

```r
### To do naive Bayes in R, we need the response variable to be a factor
### object (unlike GAM). Let's re-split the data using the same indices we
### computed above and convert our response to a factor. I like to also
### explicitly assign an order to facors using the levels input.
perm <- sample (x= nrow ( vehdata ))
set1 <- vehdata [ which ( perm <= 3* nrow ( vehdata )/4) , ]
set2 <- vehdata [ which ( perm > 3* nrow ( vehdata )/4) , ]
X.train = set1[,-19]
X.valid = set2[,-19]
Y.train = set1[, 19]
Y.train = as.factor(Y.train)
Y.valid = set2[, 19]
Y.valid = as.factor(Y.valid)

### We fit naive Bayes models using the NaiveBayes() function from the klaR
### package. This function uses predictor matrix/response vector syntax. You
### can also specify if you want to use kernel density estimation by setting
### usekernel=TRUE. Setting this to false uses the normal distribution for
### each predictor.
fit.NB.userkernel = NaiveBayes(X.train, Y.train, usekernel = T)
fit.NB.notuserkernel = NaiveBayes(X.train, Y.train, usekernel = F)


### Next, let's get predictions, their corresponding confusion matrix and
### the misclassification rate. Predictions from NaiveBayes models give
### predicted class labels and probabilities, so we have to extract the
### class labels using $class

#Kernel
pred.NB.userkernel.raw = predict(fit.NB.userkernel, X.valid)
pred.NB = pred.NB.userkernel.raw$class

table(Y.valid, pred.NB, dnn = c("Obs", "Pred"))
(mis.NB = mean(Y.valid != pred.NB))
```

(a) **Report them in the order**

i. No PC, Kernel

```r
#Kernel
pred.NB.userkernel.raw = predict(fit.NB.userkernel, X.valid)
pred.NB = pred.NB.userkernel.raw$class

table(Y.valid, pred.NB, dnn = c("Obs", "Pred"))
(mis.NB = mean(Y.valid != pred.NB))
```

```
> (mis.NB = mean(Y.valid != pred.NB))
[1] 0.3867925
```

ii. No PC, Normal

```r
#Kernel
pred.NB.notuserkernel.raw = predict(fit.NB.notuserkernel, X.valid)
pred.NB = pred.NB.notuserkernel.raw$class

table(Y.valid, pred.NB, dnn = c("Obs", "Pred"))
(mis.NB = mean(Y.valid != pred.NB))
```

```
> (mis.NB = mean(Y.valid != pred.NB))
[1] 0.5235849
```

iii. PC, Kernel

```
#####PC
### It can be helpful with naive Bayes to first do a principal components
### analysis (see Lecture 7). We will do PCA on the training set, then
### apply the same transformation to the validation set using the predict()
### function. Remember to scale the predictors by setting scale. to true.
fit.PCA = prcomp(X.train, scale. = T)

X.train.PC = fit.PCA$x  # Extract the PCs
X.valid.PC = predict(fit.PCA, set2)


### Now we can use the NaiveBayes() function in exactly the same way as
### above.
fit.NB.PC.userkernel = NaiveBayes(X.train.PC, Y.train, usekernel = T)
fit.NB.PC.notuserkernel = NaiveBayes(X.train.PC, Y.train, usekernel = F)

#Kernel
pred.NB.PC.userkernel.raw = predict(fit.NB.PC.userkernel, X.valid.PC)
pred.NB = pred.NB.PC.userkernel.raw$class

table(Y.valid, pred.NB, dnn = c("Obs", "Pred"))
(mis.NB = mean(Y.valid != pred.NB))
```

```
> pred.NB = pred.NB.PC.userkernel.raw$class
> (mis.NB = mean(Y.valid != pred.NB))
[1] 0.259434
```

iv. PC, Normal

```
> pred.NB = pred.NB.PC.notuserkernel.raw$class
> (mis.NB = mean(Y.valid != pred.NB))
[1] 0.245283
```

(b) **Comment on how the test errors compare to each other. Do PC or kernel density estimation seem to help?**
Yes, the value with PC looks much lower value. The value with PC and without kernel shows the lowest value

(c) **How does test error compare to other methods?**
LDA and Qda shows the error lower than 0.2, so these cannot be the best methods.