Needell_Coen_HW2

Coen D. Needell

1/29/2020

The Bayes Classifier

1.

a)

```
set.seed(456)
```

b)

```
dat1 <- tibble(x1 = runif(200, -1, 1), x2 = runif(200, -1, 1))
```

c)

```
dat1$epsilon = rnorm(200, mean=0, sd=.5)
dat1$y = dat1$x1 + dat1$x1^2 + dat1$x2 + dat1$x2^2 + dat1$epsilon
```

d)

Y is the log-odds of success, so $y < 0$ means the probabilility of success is $\mathbb{P}(Success) < 0.5$. Recall:

$$\log\left(\frac{\Pr Success}{1 - \Pr Success}\right) = Y \tag{1}$$

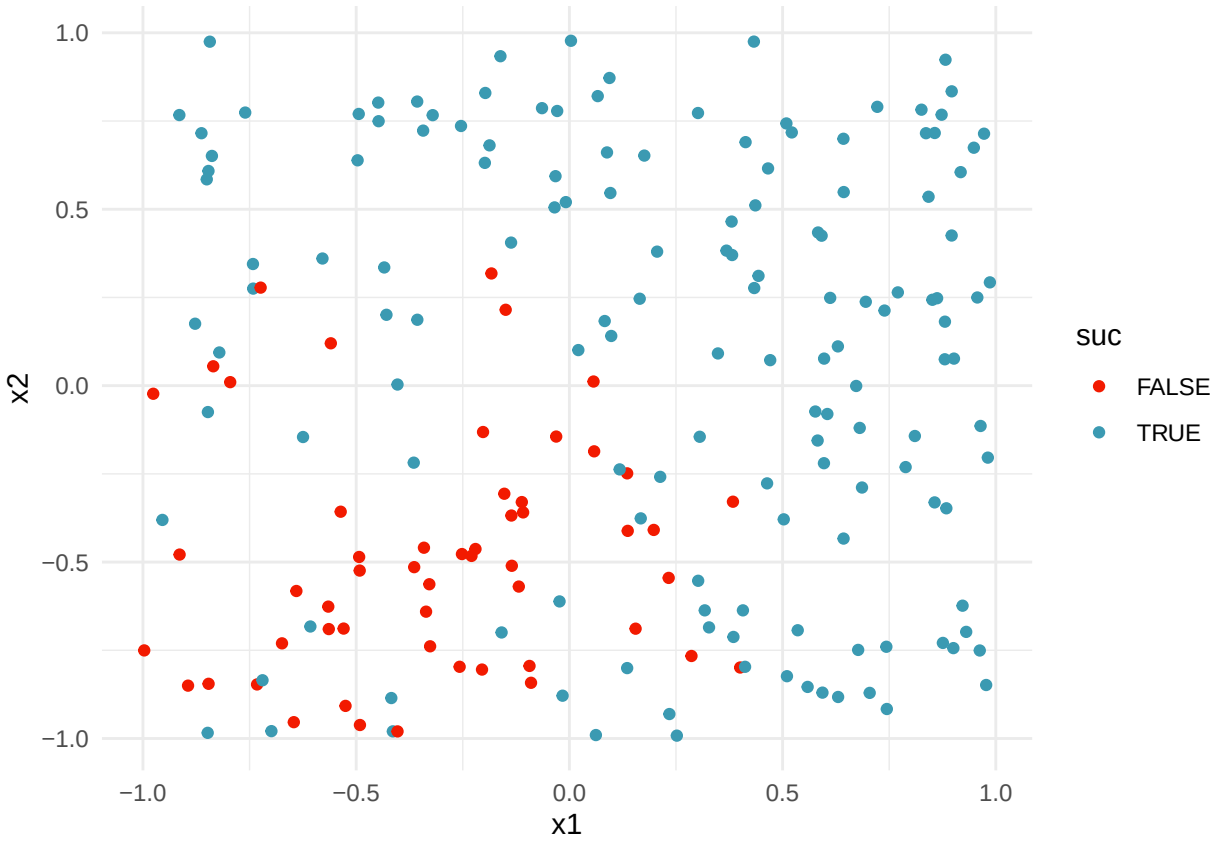
$$\therefore \tag{2}$$

$$\Pr(Success) = \frac{e^Y}{1 + e^Y} \tag{3}$$

```
dat1$p_suc = exp(dat1$y) / (1 + exp(dat1$y))
```

e)

```
dat1$suc <- dat1$p_suc > .5
dat1$suc <- as.character(dat1$suc)

ggplot(dat1, aes(x1, x2, col=suc)) +
  geom_point() +
  theme_minimal() +
  scale_color_manual(values=rev(wes_palette('Zissou1', 2, type='continuous')))
```



f)

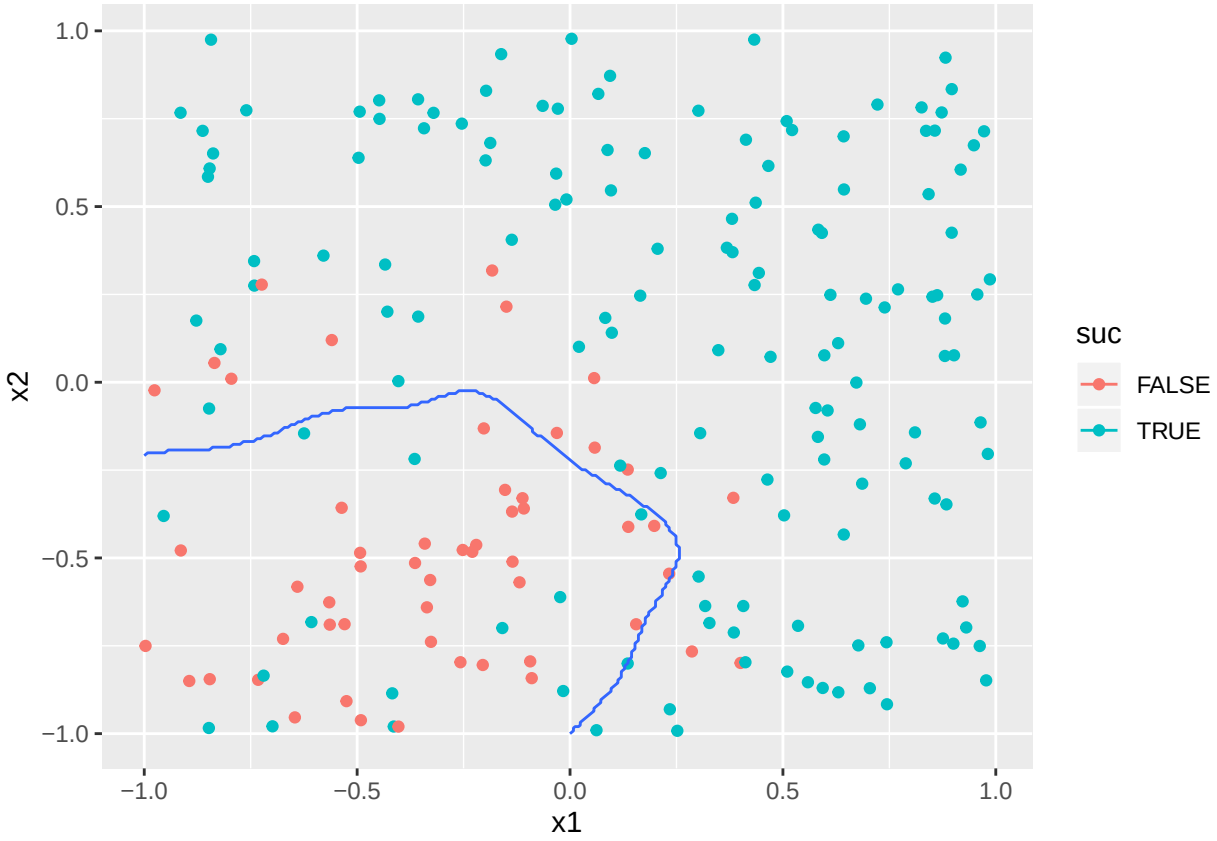
```
# I'm not going to do a split because we aren't
# doing any validation beyond caret's builtin cv

train_control1 <- trainControl(
  method='cv',
  number = 4
)

nbx1 <- dat1[, c('x1', 'x2')]
nby1 <- dat1$suc

naibayes1 <- train(
  x=nbx1,
  y=nby1,
  method = 'nb',
  trControl = train_control1
)
```

```
confusionMatrix(naibayes1)
```

```
## Cross-Validated (4 fold) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
```

```
##           Reference
## Prediction FALSE TRUE
##      FALSE  19.0  6.5
##      TRUE    8.0 66.5
##
##  Accuracy (average) : 0.855
```

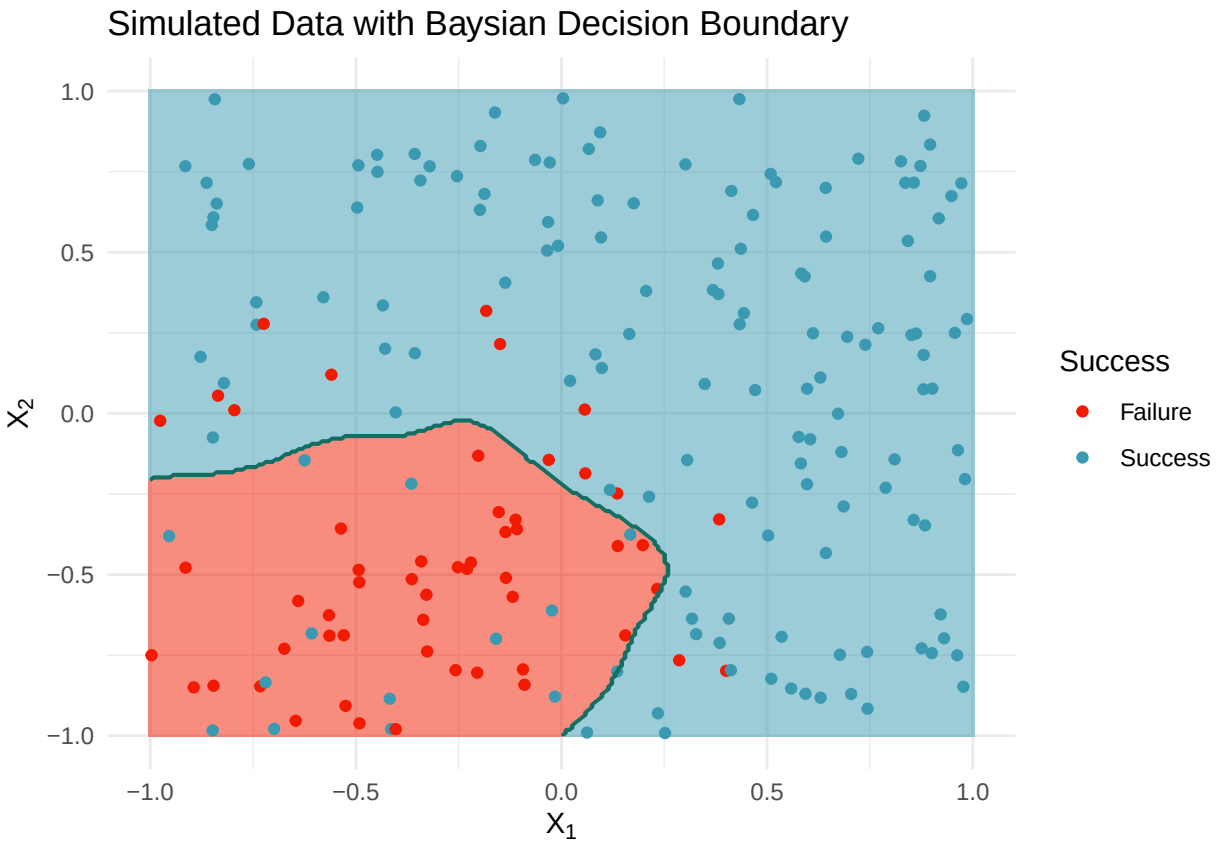
```r
linspace <- seq(-1,1,length.out = 250)
grid <- expand.grid(x1=linspace, x2=linspace)
pregrid <- predict(naibayes1, newdata=grid)
grid$suc <- as.numeric(pregrid)


ggplot(dat1, aes(x1, x2,z=suc, col=suc)) +
  geom_point() +
  geom_contour(data=grid, breaks=c(1.5))
```



g)

```r
ggplot(dat1, aes(x1, x2, z=suc)) +
  geom_raster(aes(x1, x2, fill=as.factor(suc)), data=grid, alpha=.5) +
  geom_point(aes(col=as.factor(suc)), data = dat1) +
  geom_contour(data=grid, binwidth=.5, col='#136F63') +
  theme_minimal() +
  scale_color_manual(values=rev(wes_palette('Zissou1', 2, type='continuous')),
                     labels=c('Failure', 'Success')) +
```

```
  labs(x=expression(X['1']), y=expression(X['2']), color='Success') +
  scale_fill_manual(values=rev(wes_palette('Zissou1', 2, type='continuous')),
                    guide='none') +
  ggtitle('Simulated Data with Baysian Decision Boundary')
```



Exploring Simulated Differences between LDA and QDA

2.

a)

```
f2a <- function(n){ # Not sure if using blank inputs and lapply is kosher
  # i
  dat <- tibble(x1=runif(n,-1,1), x2=runif(n,-1,1), eps=rnorm(n, 0, 1))
  dat$core <- dat$x1 + dat$x2
  dat$y <- dat$core > 0
  dat$ysim <- dat$y + dat$eps > 0

  # ii

  split <- initial_split(dat, prop=0.7)
  train <- training(split)
  test <- testing(split)

  # iii
```

```
  lda <- MASS::lda(ysim ~ x1 + x2, data=train)
  qda <- MASS::qda(ysim ~ x1 + x2, data=train)

  # iv
  trlda <- 1 - sum(predict(lda, train)$class == train$y) / length(train$y)
  telda <- 1 - sum(predict(lda, test)$class == test$y) / length(test$y)
  trqda <- 1 - sum(predict(qda, train)$class == train$y) / length(train$y)
  teqda <- 1 - sum(predict(qda, test)$class == test$y) / length(test$y)

  return(tibble(lda_train_err = trlda,
                lda_test_err = telda, qda_train_err = trqda, qda_test_err = teqda))
}

outa <- f2a(1000)
for(i in 1:999){
  outa <- bind_rows(outa, f2a(1000))

}
```

b)

```
skimr::skim(outa)
```

Table 1: Data summary

Name	outa
Number of rows	1000
Number of columns	4
Column type frequency:	
numeric	4
Group variables	None

Variable type: numeric

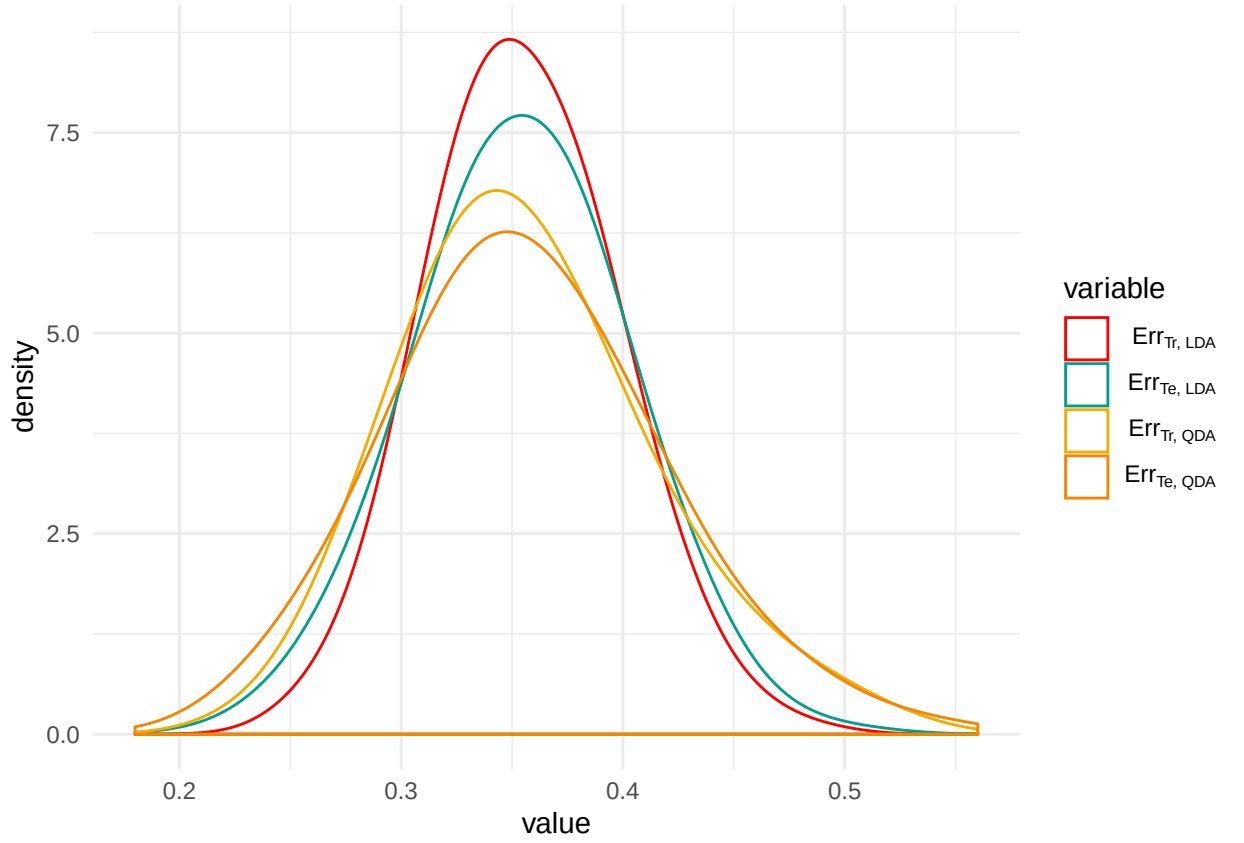
skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
lda_train_err	0	1	0.35	0.04	0.24	0.33	0.35	0.38	0.49	
lda_test_err	0	1	0.35	0.05	0.20	0.32	0.35	0.39	0.51	
qda_train_err	0	1	0.36	0.06	0.22	0.32	0.35	0.39	0.54	
qda_test_err	0	1	0.36	0.06	0.18	0.32	0.35	0.39	0.56	

```
dat2 <- reshape2::melt(outa)
```

```
## No id variables; using all as measure variables
```

```
ggplot(dat2, aes(value, col=variable)) +
  geom_density(adjust=2) +
 scale_color_manual(values=wes_palette('Darjeeling1'),
                    labels=c(expression(Err['Tr, LDA']),
                             expression(Err['Te, LDA']),
```

```
                          expression(Err['Tr, QDA']),
                          expression(Err['Te, QDA']))) +
  theme_minimal()
```



From the above graph, we can plainly see that the Linear and Quadratic non-parametric methods perform about the same. This isn't too surprising, since the linear boundary is a special case of the quadratic boundary. The thing that is interesting though is that the standard deviation for the QDA's error is higher than that for LDA, and both have a higher standard deviation for the test set. The former shows us the benefits of parsimony. Although on average both models perform about as well, LDA is more consistent. The latter feature illustrates that these methods generally are less consistent on the test set than the training set. In addition, the performance curves for the test sets are slightly right-tailed, slightly more so for QDA. This isn't so apparent from the graphs, but can be seen by examining the quartile statistics on the skimr table. This tells us that both cases are victims of overfitting, but the more parsimonious model is less so.

3

Note that this code is copied and pasted from problem *2* with minor changes.

```
f2b <- function(n){ # Not sure if using blank inputs and lapply is kosher
  # i
  dat <- tibble(x1=runif(1000,-1,1), x2=runif(1000,-1,1), eps=rnorm(1000, 0, 1))
  dat$core <- dat$x1 + dat$x2 + dat$x1^2 + dat$x2^2
  dat$y <- dat$core > 0
  dat$ysim <- dat$y + dat$eps > 0

  # ii
```

```
  split <- initial_split(dat, prop=0.7)
  train <- training(split)
  test <- testing(split)

  # iii

  lda <- MASS::lda(ysim ~ x1 + x2, data=train)
  qda <- MASS::qda(ysim ~ x1 + x2, data=train)

  # iv
  trlda <- 1 - sum(predict(lda, train)$class == train$y) / length(train$y)
  telda <- 1 - sum(predict(lda, test)$class == test$y) / length(test$y)
  trqda <- 1 - sum(predict(qda, train)$class == train$y) / length(train$y)
  teqda <- 1 - sum(predict(qda, test)$class == test$y) / length(test$y)

  return(tibble(lda_train_err = trlda,
                lda_test_err = telda,
                qda_train_err = trqda,
                qda_test_err = teqda))
}

outb <- f2b()
for(i in 1:999){
  outb <- bind_rows(outb, f2b())

}

skimr::skim(outb)
```

Table 3: Data summary

Name	outb
Number of rows	1000
Number of columns	4
Column type frequency:	
numeric	4
Group variables	None

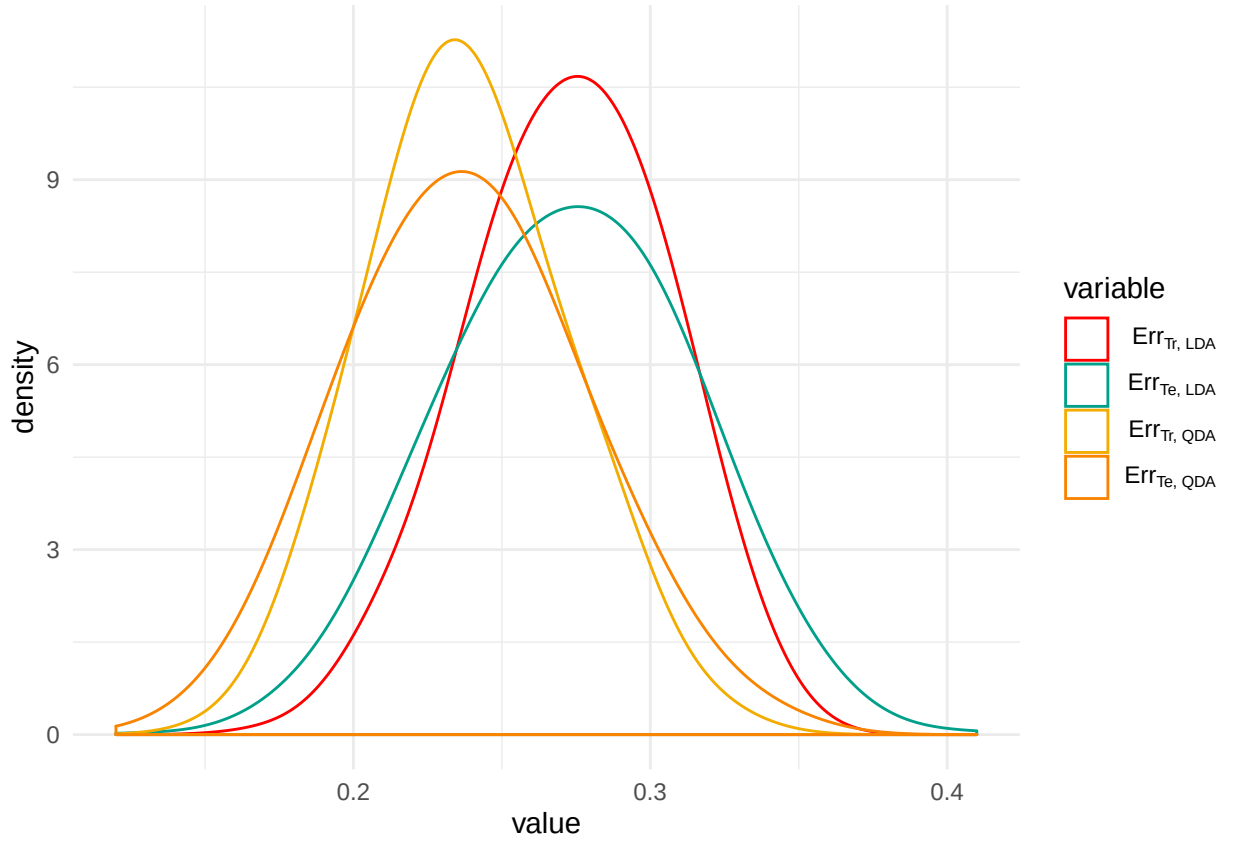
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
lda_train_err	0	1	0.27	0.03	0.17	0.25	0.27	0.30	0.35	
lda_test_err	0	1	0.27	0.04	0.13	0.25	0.27	0.30	0.41	
qda_train_err	0	1	0.24	0.03	0.15	0.22	0.24	0.26	0.34	
qda_test_err	0	1	0.24	0.04	0.12	0.21	0.24	0.26	0.35	

```
dat2 <- reshape2::melt(outb)
```

```
## No id variables; using all as measure variables
```

```
ggplot(dat2, aes(value, col=variable)) +
  geom_density(adjust=2) +
 scale_color_manual(values=wes_palette('Darjeeling1'),
                    labels=c(expression(Err['Tr, LDA']),
                             expression(Err['Te, LDA']),
                             expression(Err['Tr, QDA']),
                             expression(Err['Te, QDA']))) +
  theme_minimal()
```



We notice that, in contrast to problem 2, LDA and QDA do not perform similarly. If we run a two sample t test on these errors...

```
t.test(outa$lda_test_err, outa$qda_test_err)
```

```
##
##  Welch Two Sample t-test
##
## data:  outa$lda_test_err and outa$qda_test_err
## t = -0.90877, df = 1880.9, p-value = 0.3636
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.007000476  0.002567143
## sample estimates:
## mean of x mean of y
## 0.3542300 0.3564467
```

```r
t.test(outb$lda_test_err, outb$qda_test_err)
```

```
##
##  Welch Two Sample t-test
##
## data:  outb$lda_test_err and outb$qda_test_err
## t = 19.594, df = 1996.1, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.03143083 0.03842251
## sample estimates:
## mean of x mean of y
## 0.2728733 0.2379467
```

We reject the fail to reject the null hypothesis in the linear case, and we reject the null hypothesis in the quadratic case. The same analysis applies as in problem 2, except that since the QDA outperforms LDA, the more parsimonious model is a worse choice, and in fact is slightly more affected by overfitting in this case, whereas the QDA model actually has the test error slightly right-tailed, and the training error slightly left-tailed, telling us that it generalizes well.

4

a)

Note that again, code has been copied and pasted.

```r
f4a <- function(n){
  out <- f2a(n)
  for(i in 1:999){
    out <- bind_rows(out, f2a(n))
  }

  skimr::skim(out)

  dat <- reshape2::melt(out)
  return(dat)
}

a4 <- f4a(1e02)
```

```
## No id variables; using all as measure variables
```

```r
for(n in c(1e03, 1e04, 1e05)){
  a4 <- bind_cols(a4, f4a(n))
}
```

```
## No id variables; using all as measure variables
## No id variables; using all as measure variables
## No id variables; using all as measure variables
```

```r
a4 <- a4[-c(3,5,7)]
colnames(a4) <- c('errtype', 'e02', 'e03', 'e04', 'e05')
```

b)

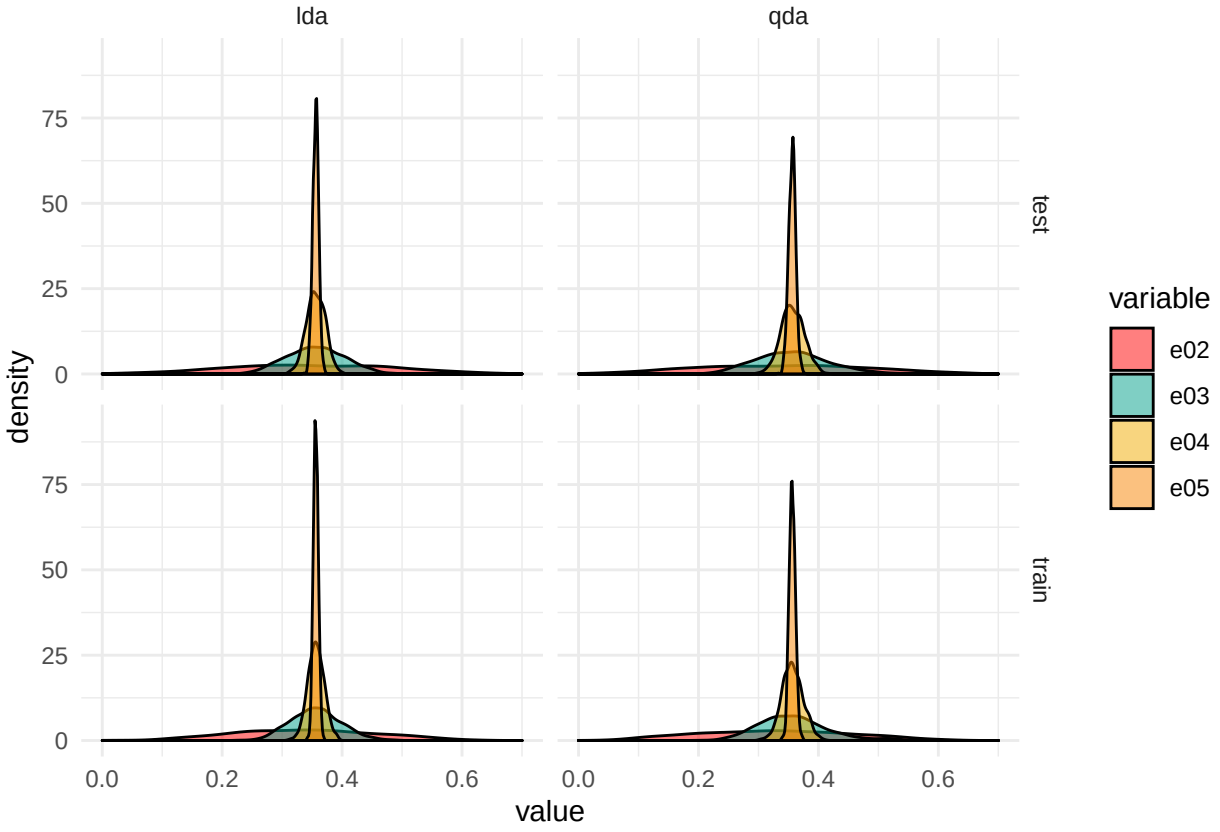
```r
b4 <- a4
b4 <- reshape2::melt(b4)
```

```
## Using errtype as id variables
```

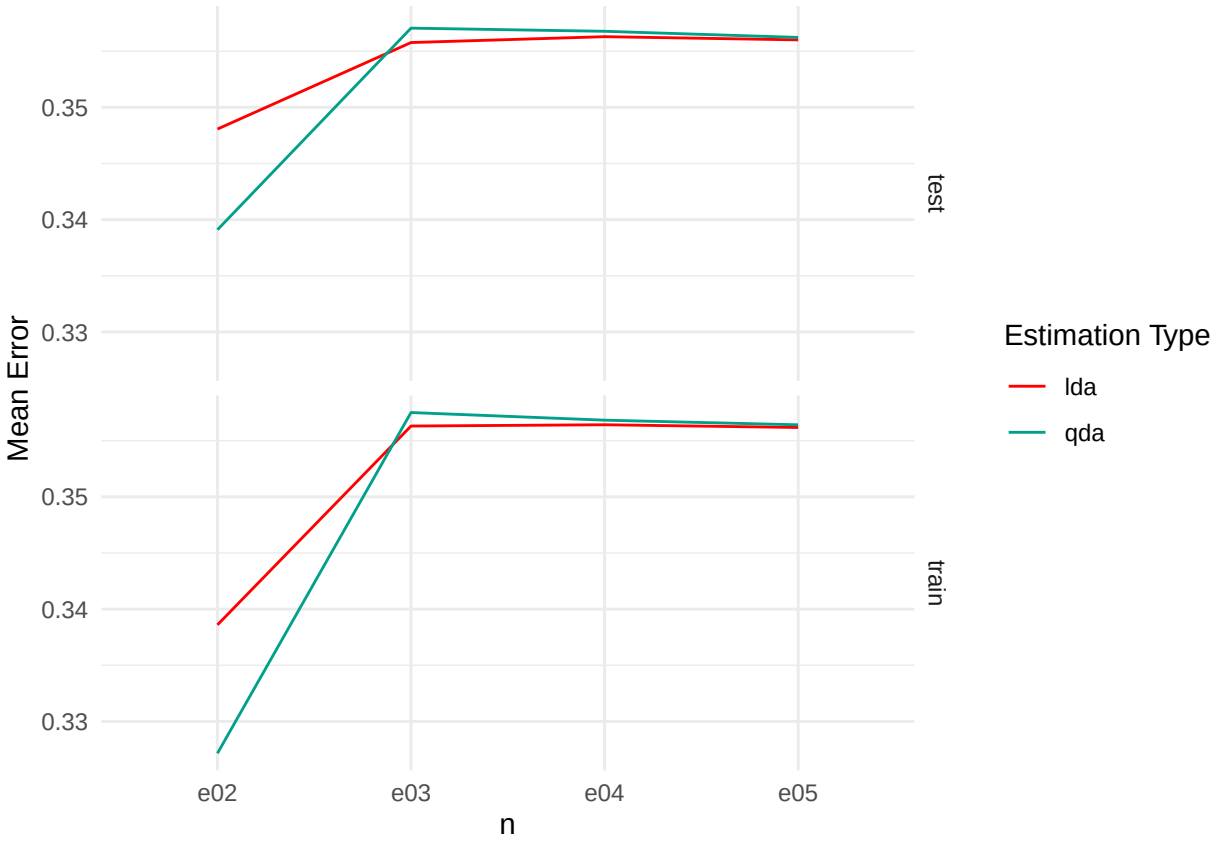
```
b4 <- separate(b4, errtype, c('Method','Type'))
```

```
## Warning: Expected 2 pieces. Additional pieces discarded in 16000 rows [1, 2, 3,
## 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, ...].
```

```
ggplot(b4, aes(x=value, fill=variable)) +
  geom_density(alpha=.5) +
  scale_fill_manual(values=wes_palette('Darjeeling1', 4)) +
  facet_grid(rows=vars(Type), cols=vars(Method)) +
  theme_minimal()
```




```
b42 <- aggregate(b4[4], by = list(b4$Method, b4$Type, b4$variable), mean)

ggplot(b42, aes(x=Group.3, y=value, col=Group.1)) +
  geom_path(aes(group=Group.1)) +
  facet_grid(rows=vars(Group.2)) +
  labs(x='n', y='Mean Error', color='Estimation Type') +
  scale_color_manual(values=wes_palette('Darjeeling1', 2)) +
  theme_minimal()
```



From these graphs, we notice a few things. The most noticable thing is that the standard deviation for each of the error distributions shrinks with n. The second thing is that n=1e02 has weirdly low error. But, it's clearly subject to some overfitting, since there's a much larger distinction between train and test errors in that case. Especially for QDA. Notably, the test error rate for QDA approaches the test error rate for LDA as n increases. This makes sense intuitively, since the linear form is a special case of the quadratic form, and larger training size means that there will be less overfitting.

Modeling Voter Turnout

5.

a)

```
mental_health <- read.csv("D:/UChicago/Modeling/problem-set-2/mental_health.csv", header=T)
mental_health <- na.omit(mental_health)

split5a <- initial_split(mental_health, prop=.7)
train5a <- training(split5a)
test5a <- testing(split5a)
```

b)

```
# i

mh_logit <- glm(vote96 ~ ., data=train5a, family=binomial)
```

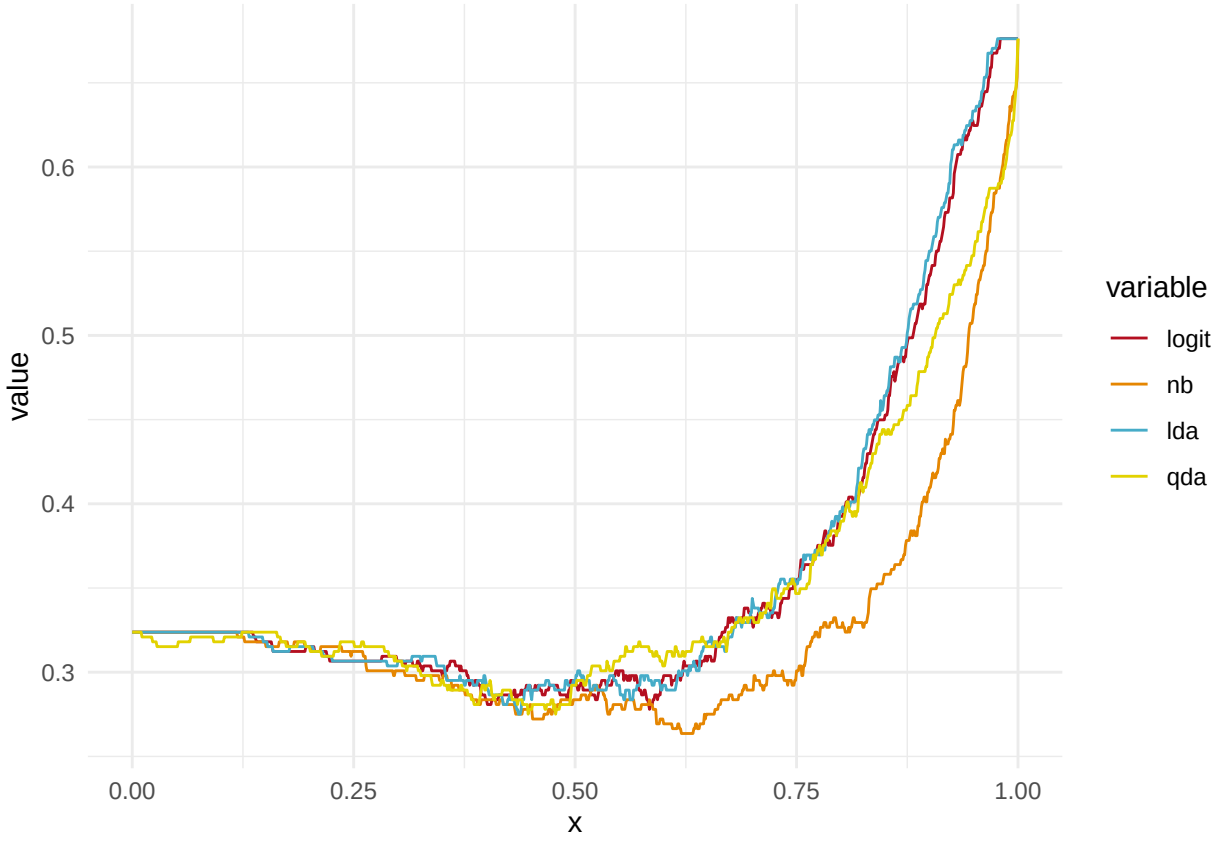
```r
# ii

mh_lindisc <- MASS::lda(vote96 ~ ., data=train5a)

# iii

mh_quaddisc <- MASS::qda(vote96 ~ ., data=train5a)

# iv

mh_naibayes <- train(
  x=train5a[-1],
  y=as.factor(train5a$vote96),
  method = 'nb'
)
# v

mse_knn <- tibble(k = 1:10,
                  knn_mod = map(k, ~ attr(class::knn(select(train5a, -vote96),
                                                     test = select(test5a, -vote96),
                                                     cl = train5a$vote96, k = ., prob = T), 'prob')))
```

c)

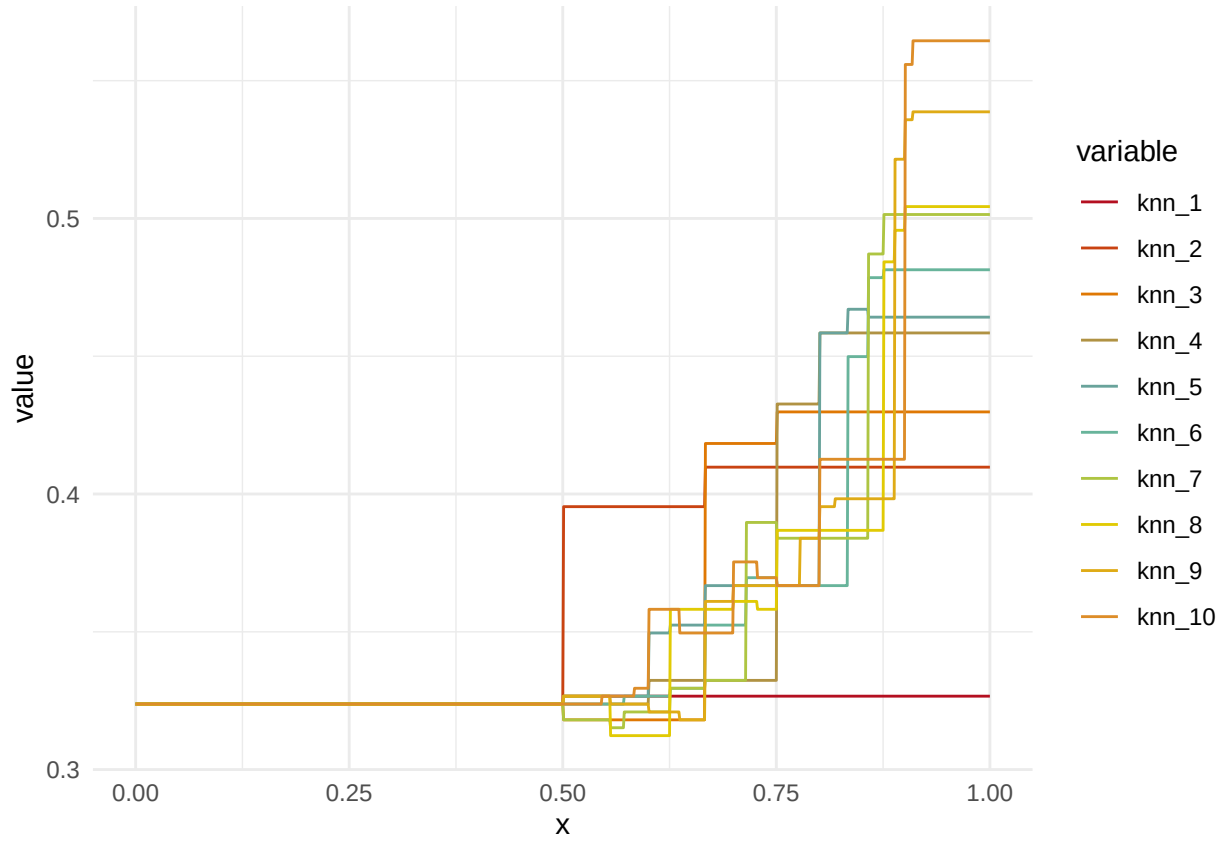
```r
logpre <- predict(mh_logit, test5a)
tests5 <- tibble(logit= exp(logpre) / (1 + exp(logpre)),
                 lda=predict(mh_lindisc, test5a)$posterior[,'1'],
                 qda=predict(mh_quaddisc, test5a)$posterior[,'1'],
                 nb=predict(mh_naibayes, test5a, type='prob')$`1`
)

tests5k <- as_tibble(mse_knn$knn_mod, .name_repair = 'minimal')
colnames(tests5k) <- sprintf("knn_%s", 1:10)
tests5 <- bind_cols(tests5, tests5k)

get_err <- function(probs, test, threshold){
  pred <- probs >= threshold
  er <- 1 - sum(pred == test) / length(test)
  return(er)
}

make_err_curve <- function(probs, test){
  grid <- seq(0,1,.001)
  out <- sapply(grid, get_err, probs=probs, test=test)
  return(out)
}

rocs <- lapply(tests5, FUN=pROC::roc, response=test5a$vote96)
err_curves <- as_tibble(sapply(tests5, make_err_curve, test=test5a$vote96))
err_curves$x <- seq(0, 1, .001)

err_curves_m <- reshape2::melt(err_curves[c('logit', 'nb', 'lda', 'qda', 'x')], id.vars='x')
```

```
err_curves_k <- reshape2::melt(err_curves[!names(err_curves) %in% c('logit', 'nb', 'lda', 'qda')], id.va
```

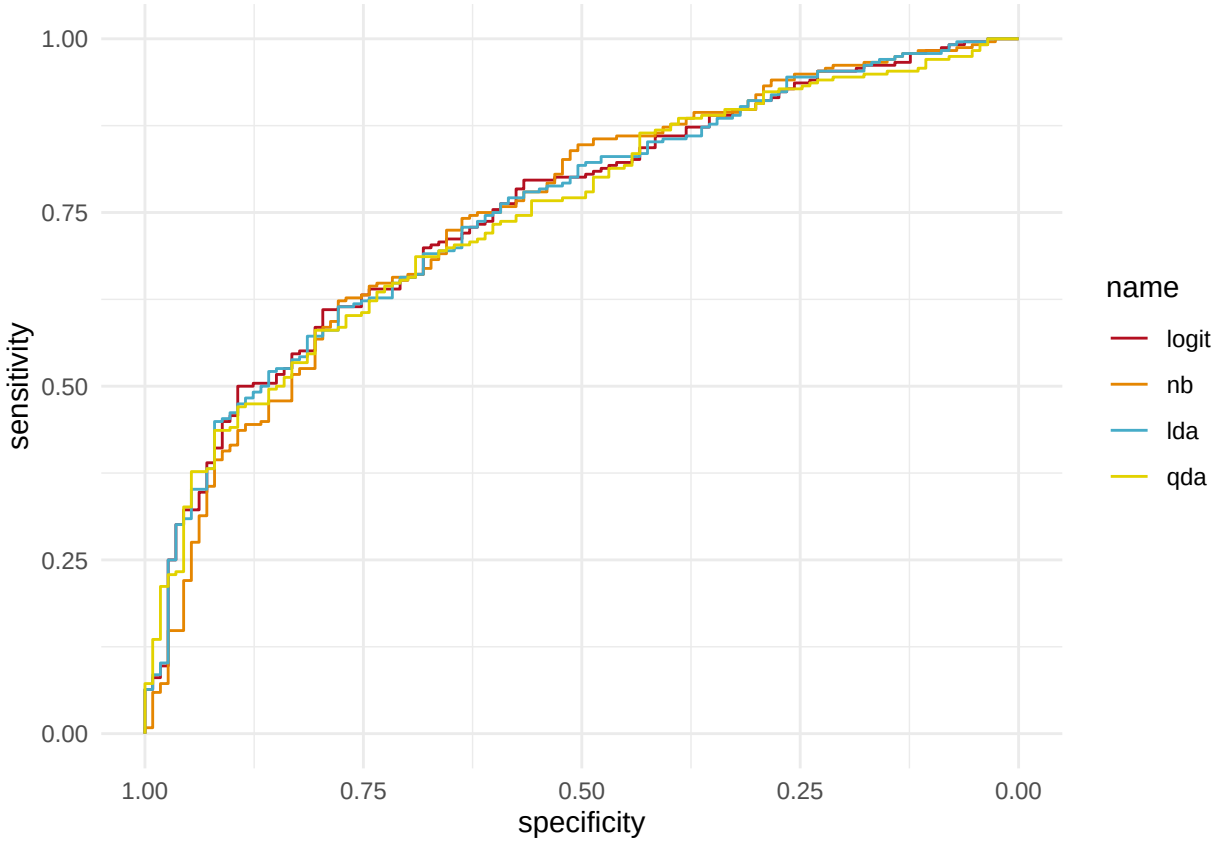
```
ggplot(err_curves_m, aes(x=x, y=value, col=variable)) +
  geom_line() +
  scale_color_manual(values=rev(wes_palette('FantasticFox1'))) +
  theme_minimal()
```



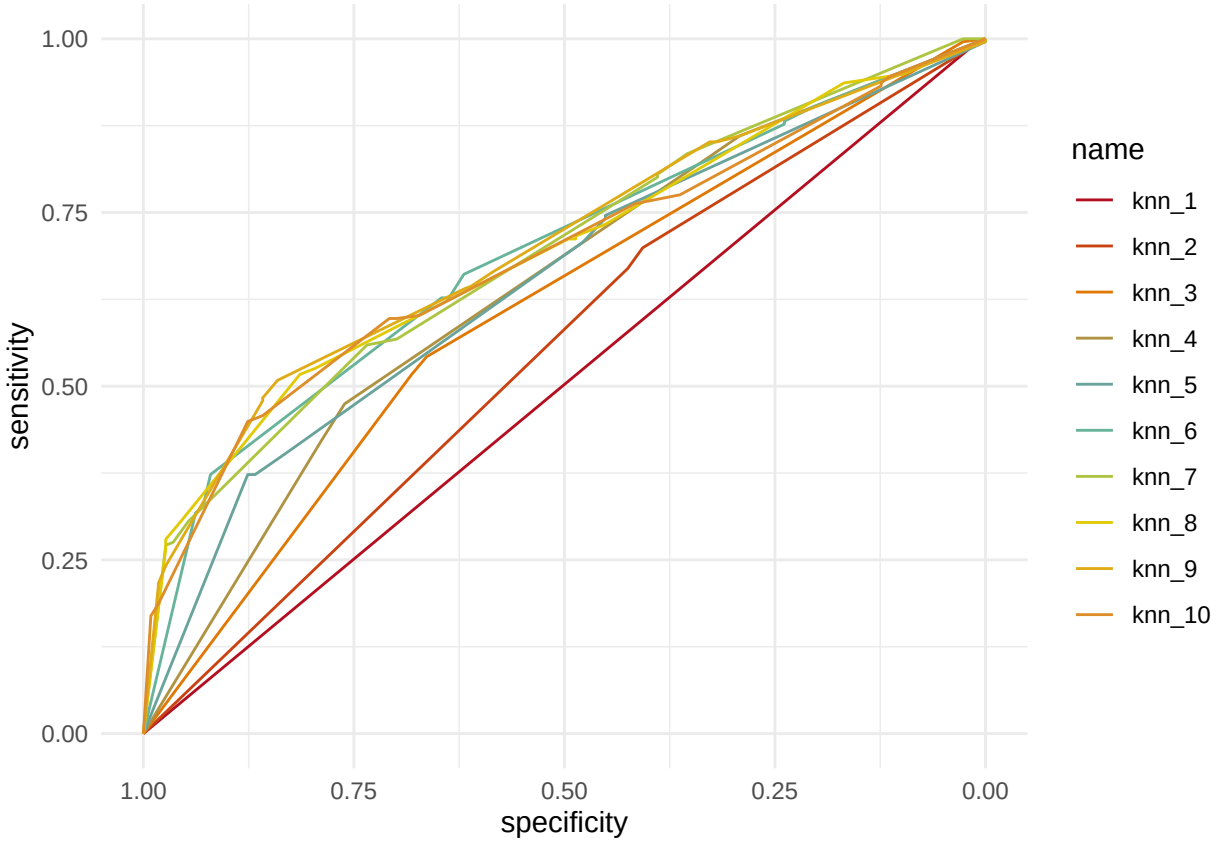

```
ggplot(err_curves_k, aes(x=x, y=value, col=variable)) +
  geom_line() +
  scale_color_manual(values=rev(wes_palette('FantasticFox1', 10, type='continuous'))) +
  theme_minimal()
```




```
pROC::ggroc(rocs[c('logit', 'nb', 'lda', 'qda')]) +
  scale_color_manual(values=rev(wes_palette('FantasticFox1', 5, type='continuous'))) +
  theme_minimal()
```




```
pROC::ggroc(rocs[5:14]) +
  scale_color_manual(values=rev(wes_palette('FantasticFox1', 10, type='continuous'))) +
  theme_minimal()
```




```
aucs <- c()
for(r in rocs){
  aucs <- c(aucs, r$auc)
}

aucs <- tibble(method = names(rocs), auc = aucs )
knitr::kable(aucs)
```

method	auc
logit	0.7523999
lda	0.7517249
qda	0.7448628
nb	0.7487251
knn_1	0.5024936
knn_2	0.5505100
knn_3	0.6082383
knn_4	0.6414242
knn_5	0.6488488
knn_6	0.6877531
knn_7	0.6908092
knn_8	0.6922341
knn_9	0.7001087
knn_10	0.6847908

d)

By 'performs the best' we're going to mean 'minimizes error'. For us, this is the naïve bayes method. The minimum of it's error occurs at a threshold of 0.424, which results in a an error less than all other method's minima. If you examine the error and ROC curves, it's clear that the naïve bayes method outperforms the other methods in most regimes. In addition, the AUC score for naïve bayes is higher than all other methods tested.