Elements of statistical learning Ch. 2 notes

James Chuang

December 28, 2016

Contents

Two simple approaches to prediction: Least Squares and Nearest Neighbors

This post follows Chapter 2.3 in the Elements of Statistical Learning. Note that the data for exactly reproducing the figures in the book are available in `mixture.example` from the R package `ElemStatLearn`, but here I'll be going through the example from scratch.

This section develops the ***linear model fit by least squares*** and the ***k-nearest neighbor (kNN)*** prediction methods. The linear model makes large assumptions about the structure of the data, and yields stable but possibly inaccurate predictions (i.e., it is a relatively 'inflexible', or high-bias/low-variance prediction method). The kNN method makes mild structural assumptions- its predictions are often accurate but can be unstable (i.e., it is a relatively 'flexible', or low-bias/high-variance prediction method).

First, I generate training data from a Gaussian mixture model as described in section 2.3.3:

> First, we generated 10 means m_k from a bivariate Gaussian distribution $N((1,0)^T, \mathbf{I})$ and labeled this class BLUE. Similarly, 10 more were drawn from $N((0,1)^T, \mathbf{I})$ and labeled class ORANGE. Then for each class we generated 100 observations as follows: for each observation, we picked an m_k at random with probability $1/10$, and then generated a $N(m_k, \mathbf{I}/5)$, thus leading to a mixture of Gaussian clusters for each class.

```
library(MASS)
library(tibble)
library(reshape2)
library(ggplot2)
library(mvtnorm)

#a function to generate data as in ESL 2.3.3
#npoints = points to generate PER COLOR
oracle = function(npoints){
  #set mean vectors for blue and orange m_k:
  blue.mu = c(1,0)
  og.mu = c(0,1)

  #choose m_k for blue and orange
  blue.means = as_data_frame(mvrnorm(n=10, mu=blue.mu, Sigma=diag(2)))
  blue.means$color = factor(rep(0,nrow(blue.means)),
                            levels = c(0,1),
                            labels = c("BLUE", "ORANGE"))
  og.means = as_data_frame(mvrnorm(n=10, mu=og.mu, Sigma=diag(2)))
  og.means$color = factor(rep(1, nrow(og.means)),
                          levels = c(0,1),
                          labels = c("BLUE", "ORANGE"))

  #randomly choose npoints number of m_k for blue and orange
```

```
  blue.randmean = blue.means[sample(1:nrow(blue.means),npoints, replace = TRUE),1:2]
  og.randmean = og.means[sample(1:nrow(og.means),npoints, replace = TRUE),1:2]

  #generate datapoints from MVrandom normal with means chosen above
  blue.data = as_data_frame(t(apply(blue.randmean,
                                    MARGIN=1,
                                    function(x) mvrnorm(n=1, mu=x, Sigma=diag(2)/5))))
  og.data = as_data_frame(t(apply(og.randmean,
                                  MARGIN=1,
                                  function(x) mvrnorm(n=1, mu=x, Sigma=diag(2)/5))))
  blue.data$color = factor(rep(0, nrow(blue.data)),
                           levels = c(0,1),
                           labels = c("BLUE", "ORANGE"))
  og.data$color = factor(rep(1, nrow(og.means)),
                         levels = c(0,1),
                         labels = c("BLUE", "ORANGE"))

  #return plotdata, a dataframe containing training data
  #return meandata, a dataframe containing the generating m_k
  return(list(plotdata = rbind(blue.data, og.data),
              meandata = rbind(blue.means, og.means)))
}

#generate training data wih 100 points of each color
train = oracle(100)
```

```
## Warning: `as_data_frame()` is deprecated, use `as_tibble()` (but mind the new semantics).
## This warning is displayed once per session.
```

```
#plot training data
(a = ggplot() +
      geom_point(data = train$plotdata,
                 aes(x=V1, y=V2, color=color),
                 shape=21, size=2, stroke=1) +
      geom_point(data = train$meandata,
                 aes(x=V1, y=V2, color=color),
                 size=3) +
      scale_color_manual(values= c("#6495ED","#FF8C00"), guide=FALSE) +
      xlab("X1") + ylab("X2"))
```

predict using linear regression

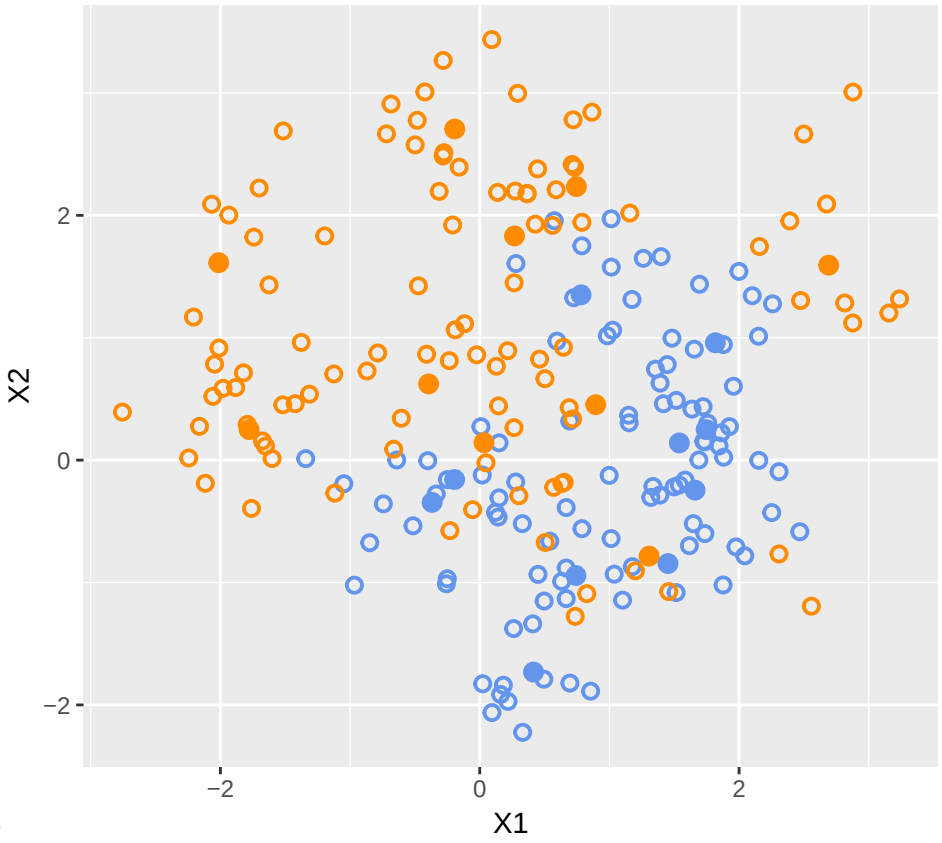
Two-class classification problem. Denote the binary coded target as Y, and treat it as a quantitative output. Predictions $\hat{Y}$ will typically lie in $[0, 1]$, and we can assign $\hat{G}$ the class label according to whether $\hat{y} > 0.5$.

Linear model:

$$\hat{Y} = X^T \hat{\beta}$$

Use method of least squares to fit $\hat{\beta}$ by minimizing the residual sum of squares:

$$\text{RSS}(\beta) = \sum_{i=1}^{N} (y_i - x_i^T \beta)^2$$



data as in ESL 2.3.3-1.bb

Figure 1: The training data. Filled points represent the means m_k of the Gaussians used to generate each class.

RSS(β) is a quadratic function, and hence its minimum always exists, though it may not be unique. The solution is easiest to characterize in matrix notation:

$$\text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta)$$

, where $\mathbf{X} \in \mathbb{R}^{N \times p}$ with each row an input vector, and $\mathbf{y} \in \mathbb{R}^N$ representing the outputs in the training set. Differentiating w.r.t. β we get the ***normal equations***:

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0$$

If $\mathbf{X}^T\mathbf{X}$ is nonsingular, then the unique solution is given by

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

, and the fitted value at the ith input x_i is $\hat{y}_i = \hat{y}(x_i) = x_i^T\hat{\beta}$. At an arbitrary input x_0 the prediction is $\hat{y}(x_0) = x_0^T\hat{\beta}$. The entire fitted surface is characterized by the p parameters $\hat{\beta}$.

Encode the output class variable G, which can take on the values BLUE or ORANGE, and encode it as a response Y where 0 is BLUE and 1 is ORANGE. Then convert the fitted values $\hat{Y}$ to a fitted class variable $\hat{G}$ according to the rule:

$$\hat{G} = \begin{cases} \text{ORANGE} & \text{if } \hat{Y} > 0.5 \\ \text{BLUE} & \text{if } \hat{Y} \leq 0.5 \end{cases}$$

```
X = as.matrix(cbind(rep(1, nrow(train$plotdata)), train$plotdata[,-3]))
y = ifelse(train$plotdata$color=="ORANGE", 1, 0)

#my linear model function
mylm = function(X,y){
  return(beta.hat = solve(t(X) %*% X) %*% t(X) %*% y)
}

beta.hat = mylm(X,y)

#set ranges for grid based on ranges of training data
x.min = round(min(train$plotdata$V1), digits=1)-0.1
x.max = round(max(train$plotdata$V1), digits=1)+0.1
y.min = round(min(train$plotdata$V2), digits=1)-0.1
y.max = round(max(train$plotdata$V2), digits=1)+0.1

x.range = seq(from=x.min, to=x.max, by=0.1)
y.range = seq(from=y.min, to=y.max, by=0.1)

x.new = expand.grid(x.range, y.range)
names(x.new) = names(train$plotdata[,-3])

#predict grid using linear model
linreg.yhat = as.matrix(cbind(rep(1, nrow(x.new)), x.new)) %*% beta.hat
linreg.pred = ifelse(linreg.yhat>0.5,"ORANGE","BLUE")
linreg.plotdata = cbind(x.new, linreg.pred)

#plot linear model classification
(b = ggplot() +
      geom_point(data = linreg.plotdata,
                 aes(x=V1, y=V2, color=linreg.pred),
                 size=0.05) +
      geom_point(data = train$plotdata,
                 aes(x=V1, y=V2, color=color),
```

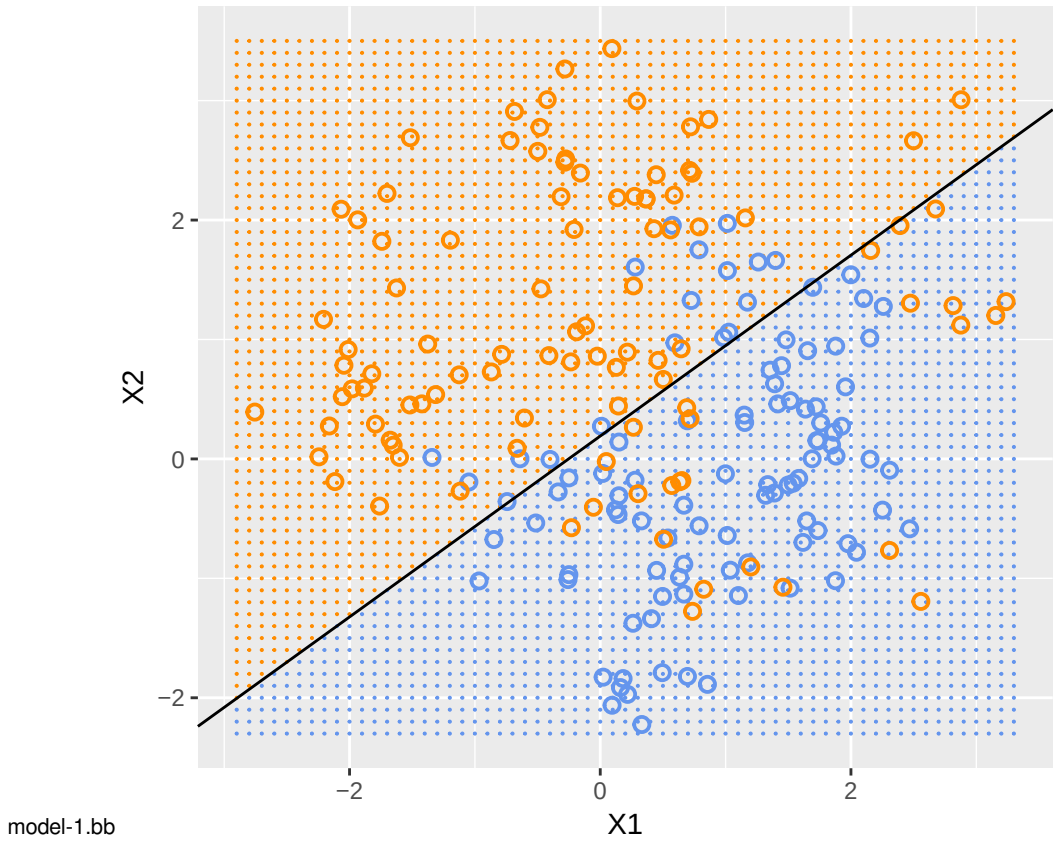


Figure 2: Classification by linear regression. The orange shaded region represents points that will be classified as ORANGE, while the blue region will be classified as BLUE

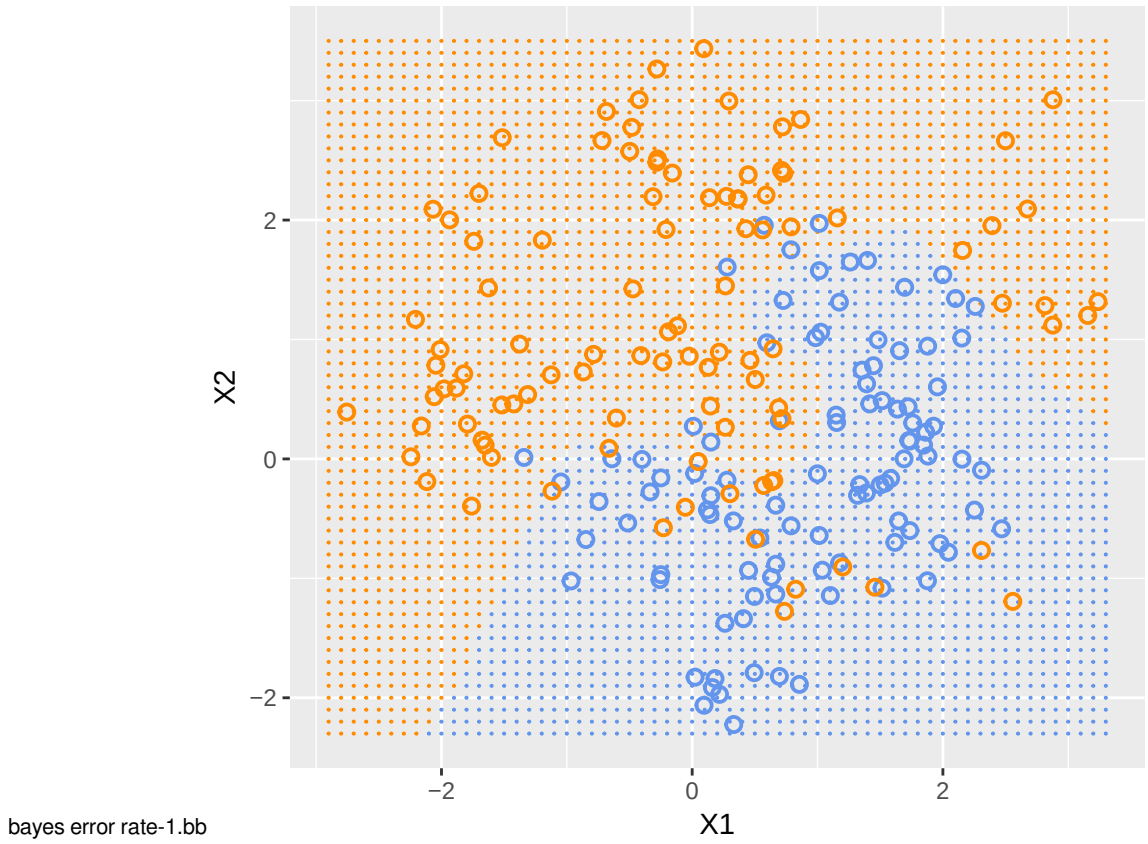
```
                shape=21, size=2, stroke=1) +
      geom_abline(slope = -beta.hat[2]/beta.hat[3],
                  intercept = (.5-beta.hat[1])/beta.hat[3],
                  color="black") +
      scale_color_manual(values= c("#6495ED","#FF8C00"), guide=FALSE) +
      xlab("X1") + ylab("X2"))
```

Because in this case we know the generating function of the distribution, we can find the Bayes-optimal decision boundary, which says that we simply classify to the class which was most likely to have generated the data point:

$$\hat{G}(x) = \max_{g \in G} \Pr(g \mid X = x)$$

. In R, this amounts to summing the probabilities of a point being generated by the multivariate normal distributions centered at the BLUE m_k, summing the probabilities of a point being generated by the multivariate normal distributions centered at the ORANGE m_k, and classifying to the class with the higher probability.

```
mybayes = function(xnew,means){
  bayes.df = data_frame(pblue=numeric(nrow(xnew)), pog=numeric(nrow(xnew)))
  for (i in 1:10){
    bayes.df$pblue = bayes.df$pblue +
        dmvnorm(x=xnew,
                mean=as.vector(as.numeric(means[i,1:2])),
                sigma = diag(2)/5)
    bayes.df$pog = bayes.df$pog +
```



bayes error rate-1.bb

Figure 3: The optimal Bayes decision boundary.

```
        dmvnorm(x=xnew,
                mean=as.vector(as.numeric(means[i+10,1:2])),
                sigma = diag(2)/5)
  }
  bayes.df$prediction = ifelse(bayes.df$pog>bayes.df$pblue, "ORANGE", "BLUE")
  bayes.plotdata = cbind(xnew, bayes.df$prediction)
  names(bayes.plotdata)[3] = "color"
  return(bayes.plotdata)
}
grid.bayes = mybayes(x.new, train$meandata)
```

```
## Warning: `data_frame()` is deprecated, use `tibble()`.
## This warning is displayed once per session.
```

```
(bayes.plot = ggplot() +
        geom_point(data = grid.bayes,
                   aes(x=V1, y=V2, color=color),
                   size=0.05) +
        geom_point(data = train$plotdata,
                   aes(x=V1, y=V2, color=color),
                   shape=21, size=2, stroke=1) +
        #geom_point(data = train$meandata, aes(x=V1, y=V2, color=color), size=4) +
        scale_color_manual(values= c("#6495ED","#FF8C00"), guide=FALSE) +
        xlab("X1") + ylab("X2"))
```

predict using nearest-neighbor methods

The k-nearest neighbor fit for $\hat{Y}$ is defined as follows:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

, where $N_k(x)$ is the neighborhood of x defined by the k closest points x_i in the training sample, defined by Euclidean distance. In words, to classify a new point, kNN finds the k closest points in the training data (a neighborhood), and classifies to the most abundant class in the neighborhood.

```
#knn classification function:
#   inputs: xnew- a matrix of new x values to be classified
#           x- matrix of training inputs
#           y- vector of training labels
#           k- # of nearest neighbors parameter
myknn = function(xnew, x, y, k){
  N.new = nrow(xnew)
  ynew = numeric(N.new)

  allpoints = rbind(xnew, x)
  dmatrix = as.matrix(dist(allpoints))[seq(N.new+1, nrow(allpoints)),seq(1, N.new)]
  rownames(dmatrix) = 1:length(y)

  for (i in 1:N.new){
    dist.vector = dmatrix[,i]
    dist.sorted = sort(dist.vector)
    nn.indices = as.numeric(names(dist.sorted[1:k]))
    ynew[i] = sum(y[nn.indices])/k
  }
  return(ynew)
}

#function for returning prediction error given predictions and labels
prediction.error = function(yhat, y){
  return(sum(diag((table(yhat,y)/length(y)) %*% matrix(c(0,1,1,0), ncol=2))))
}

#function for plotting knn
plotknn = function(k){
  knn.yhat = myknn(x.new, train$plotdata[,-3], y, k)
  knn.pred = ifelse(knn.yhat>0.5,"ORANGE","BLUE")
  knn.plotdata = as_data_frame(cbind(x.new, knn.pred))
  boundary.data = as_data_frame(cbind(x.new, knn.yhat))

  c = ggplot() +
        geom_point(data=knn.plotdata,
                   aes(x=V1, y=V2, color=knn.pred),
                   size=0.05) +
        geom_point(data=train$plotdata,
                   aes(x=V1,y=V2, color=color),
                   shape=21, size=2, stroke=1) +
        #geom_contour(data=boundary.data, aes(x=V1,y=V2,z=knn.yhat), bins=1, color="black") +
        scale_color_manual(values= c("#6495ED","#FF8C00"), guide=FALSE) +
```

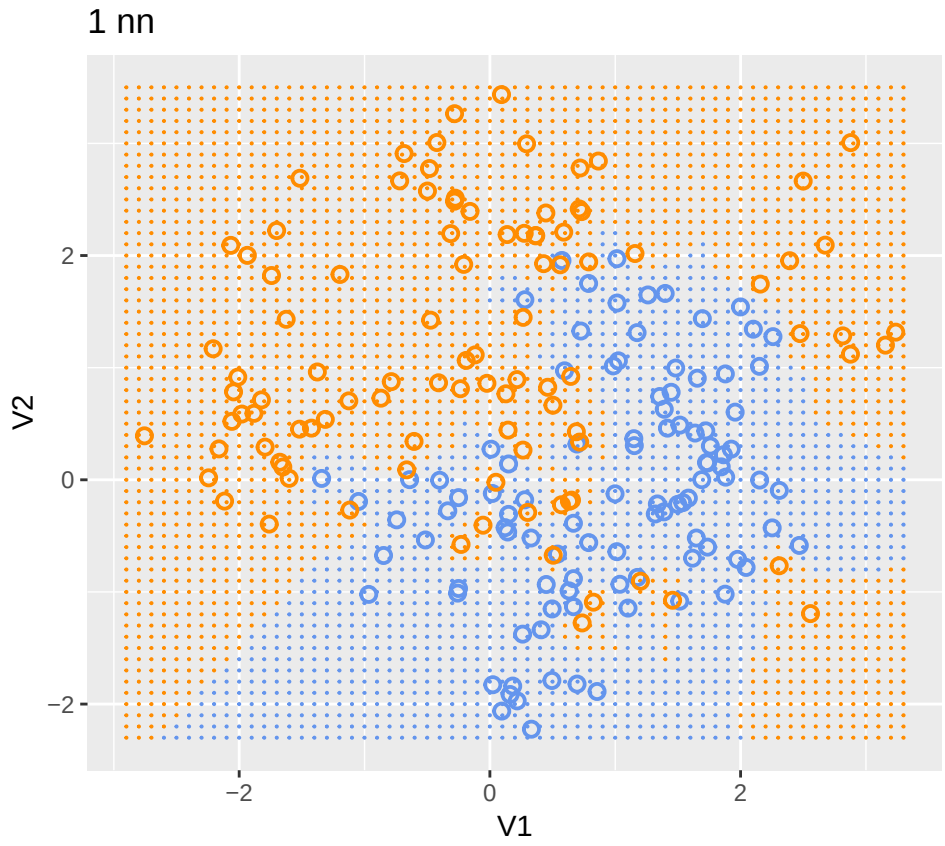


Figure 4: 1-nearest neighbor classification.

```
        ggtitle(paste(k, "nn"))
  return(c)
}

(c = plotknn(k=1))
```

```
(d = plotknn(k=15))
```

The flexibility of kNN classification is a function of k (the effective degrees of freedom, and hence the flexibility, increases as k decreases). To try to find the optimal k, we can vary k and measure classification accuracy on an independently generated test data set.

```
#calculate linear model training error
train.linreg.yhat = X %*% beta.hat
train.linreg.pred = ifelse(train.linreg.yhat>0.5, 1, 0)
linreg.trainerr = prediction.error(y, train.linreg.pred)

#generate test data
test = oracle(5000)

X.test = as.matrix(cbind(rep(1, nrow(test$plotdata)), test$plotdata[,-3]))
y.test = ifelse(test$plotdata$color=="ORANGE", 1, 0)

#predict on test data using linear model,
test.beta.hat = mylm(X.test, y.test)
```

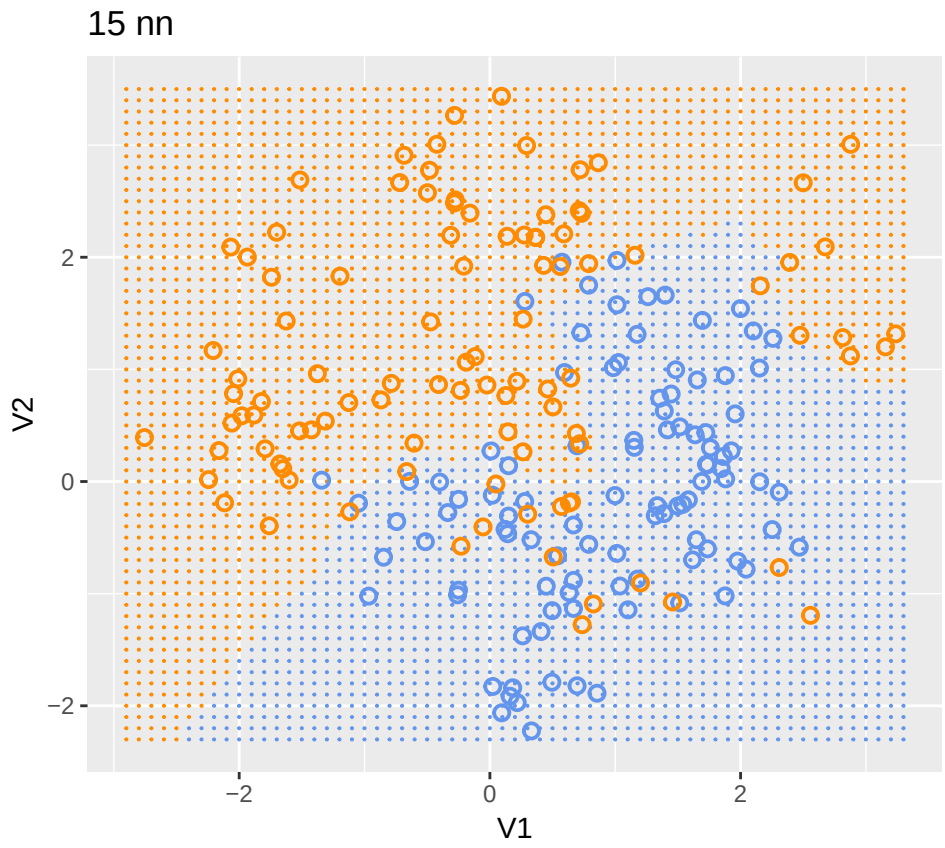


Figure 5: 15-nearest neighbor classification.

```
test.linreg.yhat = X.test %*% test.beta.hat
test.linreg.pred = ifelse(test.linreg.yhat>0.5, 1, 0)
linreg.testerr = prediction.error(y.test, test.linreg.pred)

#calculate Bayes error
bayes.error = prediction.error(ifelse(mybayes(as_data_frame(X[,-1]),
                                              train$meandata)[,3]=="ORANGE",
                                       1,0),
                               y)

#calculate training and test errors for KNN
kvector = c(1,3,5,7,11,21,31,45,69,101,151)
#kvector = c(1,5,10)

knn.error.df = data_frame(k = integer(), trainerr = double(), testerr = double())

rowcount = 1
for (k in kvector){
  knn.error.df[rowcount,1] = k
  knn.error.df[rowcount,2] = prediction.error(myknn(X[,-1], X[,-1], y, k), y)
  knn.error.df[rowcount,3] = prediction.error(myknn(X.test[,-1], X[,-1], y, k), y.test)
  rowcount = rowcount + 1
}

#plot error as function of k
(error.plot = ggplot(data = melt(knn.error.df, id.vars='k'),
                     aes(x=k, y=value, group=variable, color=variable)) +
              geom_point() +
              geom_line() +
              scale_x_log10() +
              scale_color_manual(values = c("#2F4F4F","#AE0C00")) +
              geom_hline(yintercept = bayes.error)
)
```

As we decrease k, the bias of kNN classification decreases, while the variance increases. The tradeoff between these two causes test error to decrease and then increase again as k decreases.

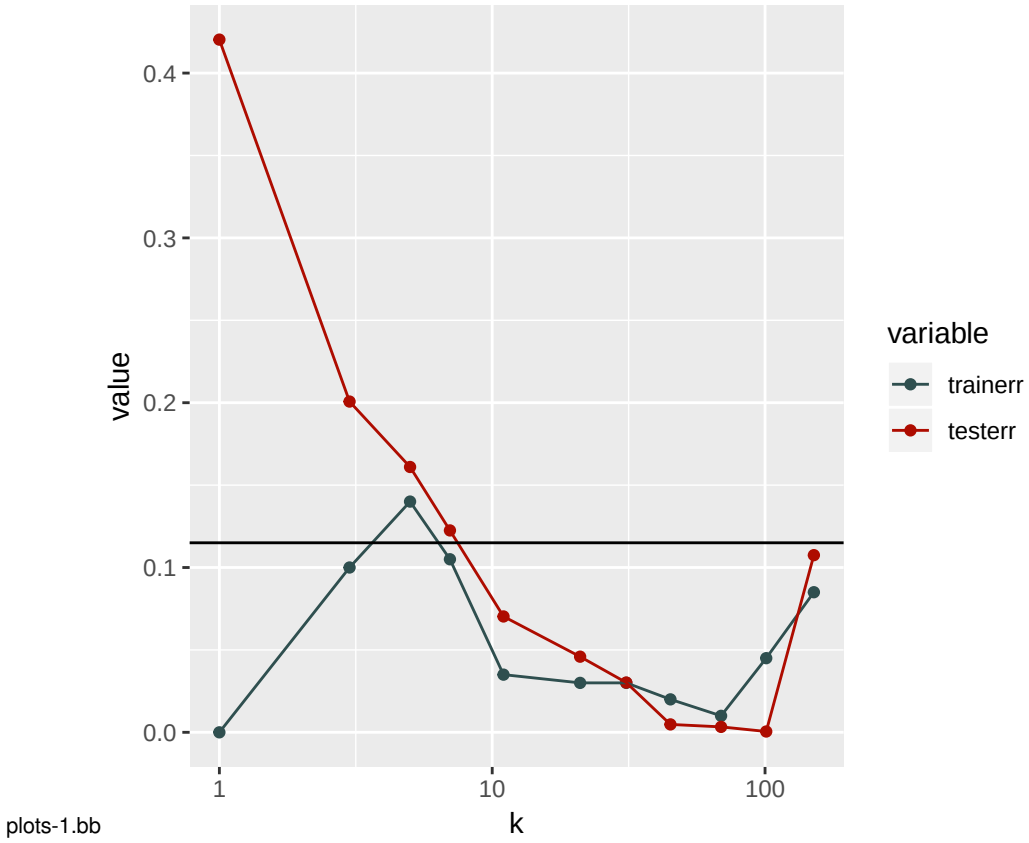


Figure 6: kNN training and test errors as a function of k. The Bayes-optimal error rate is in black.