Homework 1, STATS 315A

Stanford University, Winter 2019

Joe Higgins

Question 1

Linear versus Knn: you are to run a simulation to compare KNN and linear regression in terms of their performance as a classifier, in the presence of an increasing number of noise variables. We will use a binary response variable Y taking values $\{0, 1\}$, and initially X in R^2. The joint distribution for (Y, X) is $(1-\pi)f_0(x)$ if $y = 0$ $h_{YX}(y, x) = \pi f_1(x)$ if $y = 1$ where $f_0(x)$ and $f_1(x)$ are each a mixture of K Gaussians: $f_j(x) = \sum_{k=1}^{K} \omega_{kj}\phi(x; \mu_{kj}, \Sigma_{kj}), j = 0, 1$ (1) $\phi(x; \mu, \Sigma)$ is the density function for a bivariate Gaussian with mean vector μ and covariance matrix Σ, and the $0 < \omega_{kj} < 1$ are the mixing proportions, with $\sum_k \omega_{kj} = 1$.

(a) We will use $\pi = .5$, $K = 6$, $\omega_{kj} = \frac{1}{6}$ and $\Sigma_{kj} = \sigma^2 I = 0.2 = I$, $\forall k, j$. The six location vectors in each class are simulated once and then fixed. Use a standard bivariate gaussian with covariance I, and mean-vector $(0, 1)$ for class 0 and $(1, 0)$ for class 1 to generate the 12 location vectors.

```
#Set up sampling parameters and location vectors
rm(list = ls())
pi <- 0.5
K <- 6
J <- 1
num_classes <- J+1
omega <- matrix(1/6, nrow = K, ncol = J)
sigma2 <- 0.2

mu_0 <- c(0, 1)
mu_1 <- c(1, 0)
location_vectors_0 <- mvrnorm(K, mu = mu_0, Sigma = diag(2))
location_vectors_1 <- mvrnorm(K, mu = mu_1, Sigma = diag(2))

location_vectors <- array(c(location_vectors_0, location_vectors_1), dim = c(K,num_classes,num_classes)
```

(b) Write a function to generate a sample of N points from a density as in (1), with the parameter settings as in 1(a). The function takes as inputs a centroid matrix, N, *pi*, *omega* (vector) and *sigma*2 (σ^2), and outputs a matrix X.

```
ks <- seq(1,K,1)
js <- seq(0,J,1)

mixture_of_Gaussians <- function(j, centroid, omega, sigma2){
  mixture_location <- c(0, 0)
  #pick one of the centroids then sample from that.
  for(k in ks){
    mixture_location <- mixture_location + omega[k] * mvrnorm(1, mu = c(centroid[k,1,j+1], centroid[k,2
  }
  return(mixture_location)
}

generate_sample <- function(centroid, N, pi, omega, sigma2){
  X <- matrix(NA, nrow=N, ncol=3)
  for(i in 1:N){
```

```
    ######
    #change to binomial
    y <- 0
    if(runif(1, 0, 1) < pi){
      y <- 1
    }
    ######
    x <- mixture_of_Gaussians(y, centroid, omega, sigma2)
    X[i,] <- c(y, x)
  }
  return(X)
}
```

(c) Use your function to generate a training set of size 300 from h_{YX} , as well as a test set of size 20K. This should leave you with `xtrain`, `ytrain`, `xtest` and `ytest`.

```
train_size <- 300
test_size <- 20000

X <- generate_sample(location_vectors, train_size, pi, omega, sigma2)
y_train <- X[,1]
x_train <- X[,2:3]
X <- generate_sample(location_vectors, test_size, pi, omega, sigma2)
y_test <- X[,1]
x_test <- X[,2:3]
```

(d) Write a function to compute the Bayes classifier for this setup. It should take as input the 12×2 matrix of means, sigma, and an input matrix X. Your function should classifiy all the rows of X.

```
likelihood <- function(x, j, centroid, sigma2, pi){
  l <- 0
  for(k in ks){
    l <- l + omega[k] * dmvnorm(x, mean = c(centroid[k,1,j+1], centroid[k,2,j+1]), sigma = sqrt(sigma2)
  }

  return(l)
}

bayes_classification <- function(centroid, sigma2, pi, X){
  y_hats <- c()
  for(i in 1:nrow(X)){
    x <- X[i,]
    p_0 <- likelihood(x, 0, centroid, sigma2)*(1-pi)
    p_1 <- likelihood(x, 1, centroid, sigma2)*(pi)
    classification <- which.max(c(p_0, p_1)) - 1
    y_hats <- c(y_hats, classification)
  }
  return(y_hats)
}

y_hat_bayes <- bayes_classification(location_vectors, sigma2, pi, x_train)

get_accuracy <- function(y_hat, y){
  correct <- y_hat_bayes == y_train
  pct_correct = sum(correct)/length(correct)
```

```
  return(pct_correct)
}

get_accuracy(y_hat_bayes, y_train)
```

```
## [1] 0.98
```

Question 3: Bootstrap

Frequentist inference goes as follows. We have a random sample of N observations from a population distribution F, and we compute some statistic of interest (lets assume a scalar). The sampling distribution of that statistic is what we would get if we were able to repeat the process a large number of times: drawing samples of size N from F and computing the statistic each time. From this we could compute a quantity of interest, such as the standard deviation (standard error), which gives us an idea of the precision of our statistic. Many of our favorite models have convenient assumptions which allow us to derive theoretical expressions for say, the standard error, which can then be computed for our one and only sample. Sometimes these calculations are intractable.

The *bootstrap*, invented by our own Prof Brad Efron around 1980, is a cunning device for approximating the sampling distribution. Let $\hat{F}$ be the empirical distribution of our sample (the distribution that puts mass $1/N$ on each of the observed samples). We now draw a sample of size N from $\hat{F}$. This amounts to drawing from the original N observations a sample of size N with replacement. We compute our statistic on this bootstrap sample. This process is repeated a large number B times, and we use the bootstrap distribution of our statistic to approximate its sampling distribution.

(a) Assume the model $y = f(x) + \epsilon$ with $x \sim U[0, 1]$ and $\epsilon \sim N(0, (0.2)^2)$. For us $f(x) = 1 + x^2$. We have a sample of size 50 from this model. We fit a polynomial regression of degree 2 (we don't know the true model, so we include a linear term as well as a quadratic term), and build a prediction function $f(x)$ from the fitted model. We are interested in estimating $x_0 = \hat{f}^{-1}(1.3)$. (This kind of thing is done in dose-response experiments, and we are trying to estimate the dose that achieves a desired output). Write a function for taking a sample and delivering an estimate for $\hat{x}_0$.

```
rm(list = ls())
generate_model_from_pop <- function(n){
  X <- runif(n,0,1)
  X2 <- X^2
  y_noiseless <- 1+X2

  eps <- rnorm(n, 0, sqrt(0.2^2))
  y <- y_noiseless + eps
  return(lm(y ~ X + X2))
}

generate_model_from_boostrap <- function(n, X){
  X_bs <- sample(X, n, replace = TRUE)
  X2 <- X_bs^2
  y_noiseless <- 1+X2

  eps <- rnorm(n, 0, sqrt(0.2^2))
  y <- y_noiseless + eps
  return(lm(y ~ X_bs + X2))
}

f_hat <- function(x, model){
```

```
  features <- c(1, x, x^2)
  y_hat <- features %*% model$coefficients
  return(y_hat)
}

f_hat_inverse <- function(y, model){
  a <- model$coefficients[3]
  b <- model$coefficients[2]
  intcp <- model$coefficients[1]
  c <- intcp - y

  x_hat <- (-b + sqrt(b^2 - 4*a*c)) / (2*a)

  return(x_hat)
}

model <- generate_model_from_pop(50)
x_hat_0 <- f_hat_inverse(1.3, model)
x_hat_0
```

```
##         X
## 0.5291299
```

(b) Generate a sample from this model, and compute $\hat{x}_0$ for this sample.

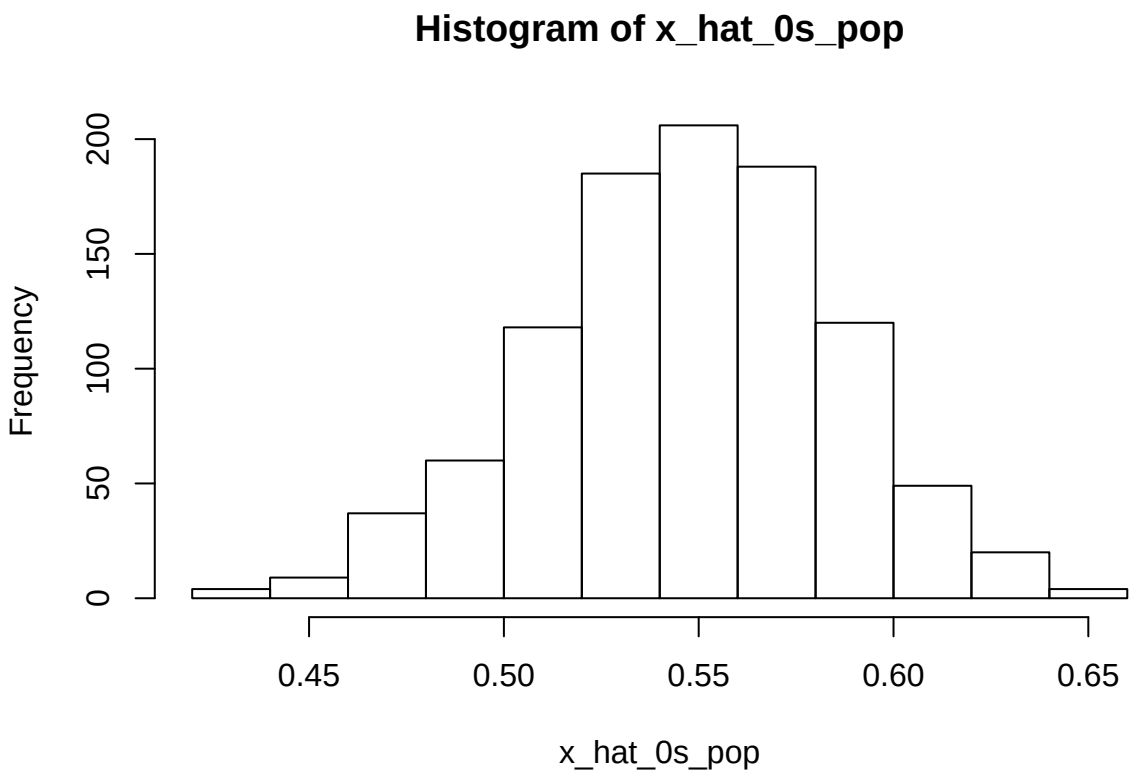
```
model <- generate_model_from_pop(50)
x_hat_0 <- f_hat_inverse(1.3, model)
x_hat_0
```

```
##         X
## 0.5290684
```

```
#generate sameple means create 50 x and y values
#generate model is rerunning the linear model, we will generate 1000 lms with 50 datapoints each and lo
```

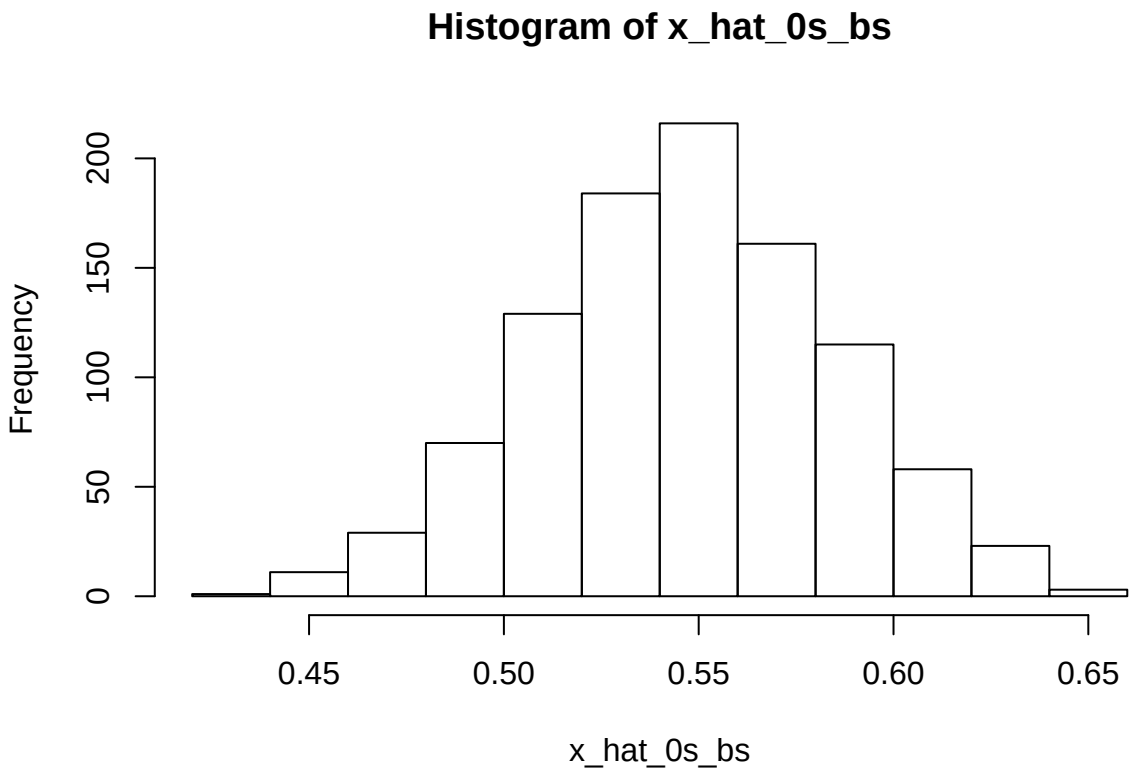
(c) Simulate a 1000 realizations from this model, to approximate the sampling distribution of $\hat{x}_0$.

```
realizations <- 1000
x_hat_0s_pop <- c()
for(i in 1:realizations){
  model <- generate_model_from_pop(50)
  x_hat_0s_pop <- c(x_hat_0s_pop, f_hat_inverse(1.3, model))
}

hist(x_hat_0s_pop)
```



(d) Using just your original sample, compute the bootstrap distribution of $\hat{x}_0$.

```
realizations <- 1000
X <- runif(50,0,1)
x_hat_0s_bs <- c()
for(i in 1:realizations){
  model <- generate_model_from_boostrap(50, X)
  x_hat_0s_bs <- c(x_hat_0s_bs, f_hat_inverse(1.3, model))
}

hist(x_hat_0s_bs)
```



(e) Compare these two distributions, and in particular their standard deviations.

```
sd(x_hat_0s_pop)
```

```
## [1] 0.03819153
```

```
sd(x_hat_0s_bs)
```

```
## [1] 0.03805213
```