DATA 558: Statistical Machine Learning
Homework 1

Abhishek Saini

April 2022

1. Suppose that you are interested in performing regression on a particular dataset, in order to answer a particular scientific question. You need to decide whether to take a parametric or a non-parametric approach.

   (a) In general, what are the pros and cons of taking a parametric versus a non-parametric approach?

   In a parametric approach, we make assumptions about the functional form that we are trying to learn. This additional assumption makes the model simpler and relatively easier to train and interpret. However, since the model is simple it may not learn if the true functional form is so complex that it cannot be represented using the simpler model. Parametric models are also easier to interpret as we know the functional form. Parametric models require fewer training observations to learn the model due to the simplifying assumption made about the true functional form. Non parametric models can learn complex functional forms but require a large amount of training data to accurately learn the true representation. If the number of training samples are not sufficient, non-parametric models may fit the noise instead of the data and hence suffer from overfitting. Thus parametric models are generally less flexible than non-parametric models but are easier to interpret, require relatively less data to train and less likely to overfit the data.

   (b) What properties of the data or scientific question would lead you to take a parametric approach?

   If I am more interested in inference or explaining the patterns in the data, I would go for a parametric approach. If there are few training observations, I would go for a parametric approach. In some cases if a parametric approach is able to fit the data well, I would consider a parametric approach for prediction tasks as well because of the added benefit of easier interpretability.

   (c) What properties of the data or scientific question would lead you to take a non-parametric approach?

   If I am more interested in making accurate prediction models and the data cannot be

modelled well using a parametric approach, I would go for a non-parametric approach. If there are large number of training observations and the data cannot be modelled well using a parametric approach, I would go for a non parametric approach for prediction tasks.

2. In each setting, would you generally expect a flexible or an inflexible statistical machine learning method to perform better? Justify your answer.

   (a) Sample size n is very small, and number of predictors p is very large.
   I would expect an inflexible model to perform better because, the flexible model would need lot more training samples to be able to learn an accurate approximation of the function that describes the data or else it would overfit the data.

   (b) Sample size n is very large, and number of predictors p is very small.
   I would expect a flexible model to perform well as the large sample size would mean that the function is better able to learn the underlying function instead of overfitting the data. Even though the sample size is large, an inflexible model may not learn the functional form as it may not be flexible enough to learn the underlying functional form and could suffer from high bias.

   (c) Relationship between predictors and response is highly non-linear.
   I would expect a flexible model to perform better as the inflexible model may not always be able to learn the functional form due to the highly non-linear relation between predictors and response.

   (d) The variance of the error terms, i.e. $\sigma^2 = \text{Var}(\epsilon)$, is extremely high.
   I would expect an inflexible model to perform better as the flexible model may end up fitting the noise instead of the data.

3. For each scenario, determine whether it is a regression or a classification problem, determine whether the goal is inference or prediction, and state the values of n (sample size) and p (number of predictors).

   (a) I want to predict each student's final exam score based on his or her homework scores. There are 50 students enrolled in the course, and each student has completed 8 homeworks.
   This problem is better modeled as a regression problem because the value we want to predict can take any value between 0 (assuming no negative marks) and the maximum total possible score in the finals.
   The goal is prediction since we are interested in predicting the scores in the final exams based on the scores in their scores in the homeworks.
   $n = 50, p = 8$

   (b) I want to understand the factors that contribute to whether or not a student passes this course. The factors that I consider are (i) whether or not the student has previous programming experience; (ii) whether or not the student has previously studied linear algebra;

(iii) whether or not the student has taken a previous stats/probability course; (iv) whether or not the student attends office hours; (v) the student's overall GPA; (vi) the student's year (e.g. freshman, sophomore, junior, senior, or grad student). I have data for all 50 students enrolled in the course.

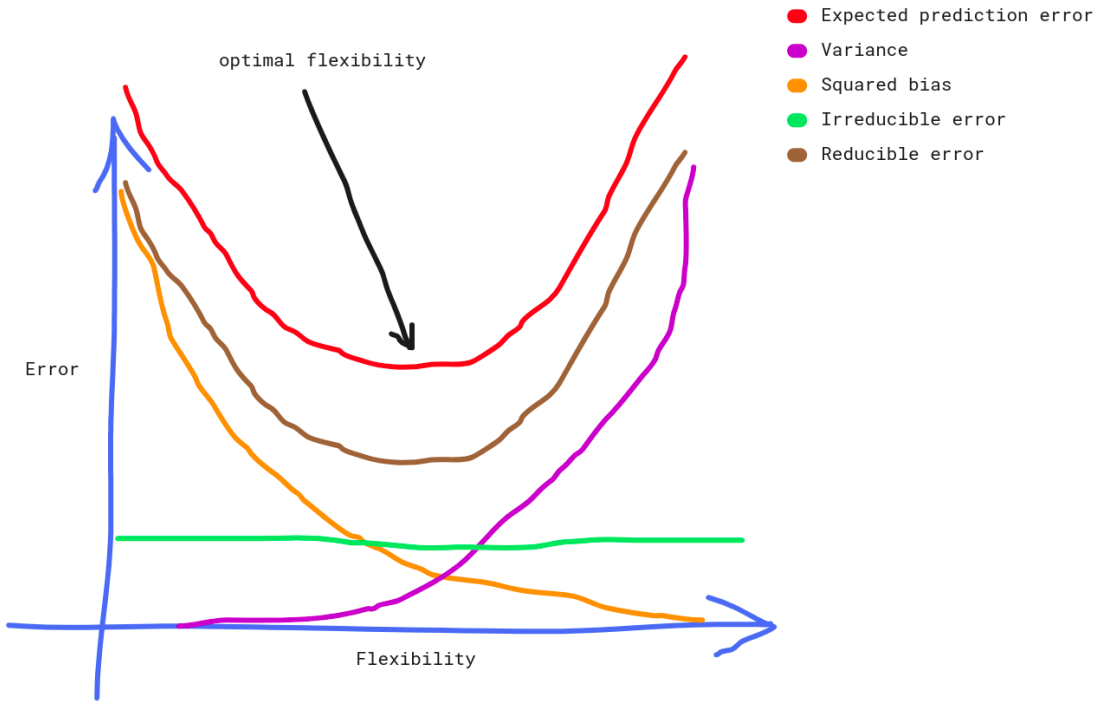
This problem is better modeled as a classification problem as the response that we are interested in takes only two values depending on whether the student passed or failed the course.

The goal is inference as we are interested in identifying the factors that contribute to a student passing or failing the course.

$n = 50, p = 6$ (assuming the the student's year is encoded using only one variable and we don't go for something like one-hot encoding)

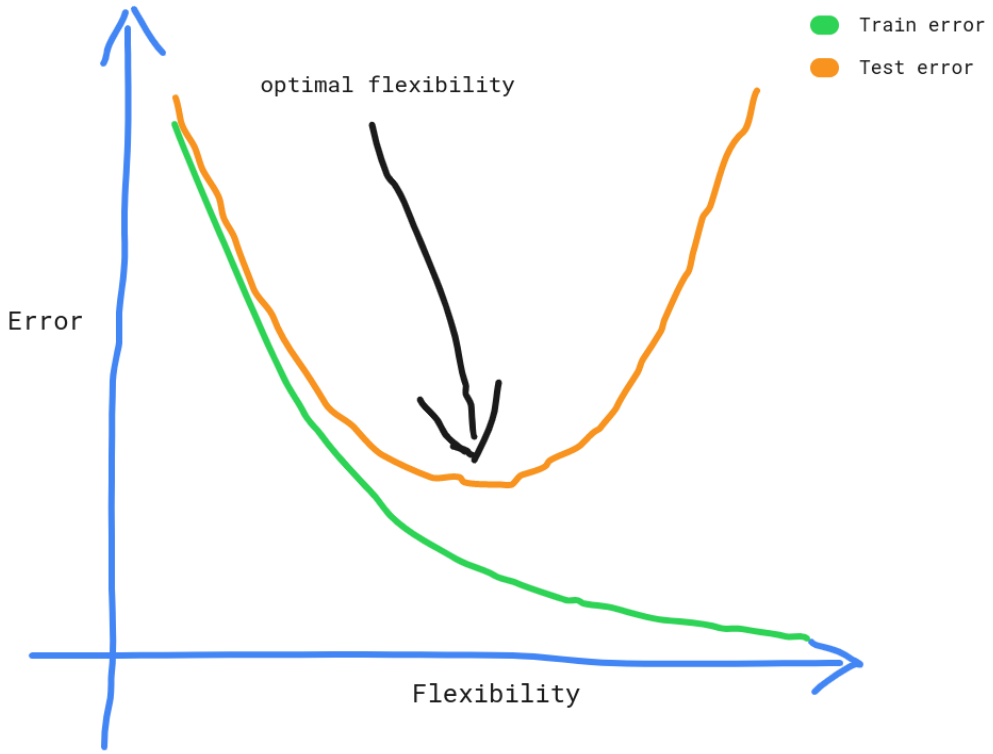
4. This problem has to do with the bias-variance trade-off and related ideas, in the context of regression. For (a) and (b), it's okay to submit hand-sketched plots: you are not supposed to actually compute the quantities referred to below on data; instead, this is a thought exercise.

   (a) Make a plot, like the one we saw in class, with "flexibility" on the x-axis. Sketch the following curves: squared bias, variance, irreducible error, reducible error, expected prediction error. Be sure to label each curve. Indicate which level of flexibility is "best".



   (b) Make a plot with "flexibility" on the x-axis. Sketch curves corresponding to the training

error and the test error. Be sure to label each curve. Indicate which level of flexibility is "best".



(c) Describe an $\hat{f}$ that has extremely low bias, and extremely high variance. Explain your answer.

An $\hat{f}$ that interpolates through all the points in the training dataset has a zero bias but an extremely high variance. The bias is zero because on average this strategy will pass through an unbiased estimate of the true functional form. The variance will be high because for each realization of the training set the function learnt will be very different.

(d) Describe an $\hat{f}$ that has extremely high bias, and zero variance. Explain your answer.

$\hat{f} = 0$ will have zero variance and extremely high bias. This is because no matter what the training set is, there is no variability in $\hat{f}$ since it is a constant. It will have high bias (except in the extremely unlikely case when the true functional form is that $f(x) = 0$) because the prediction always deviates from the true functional form.

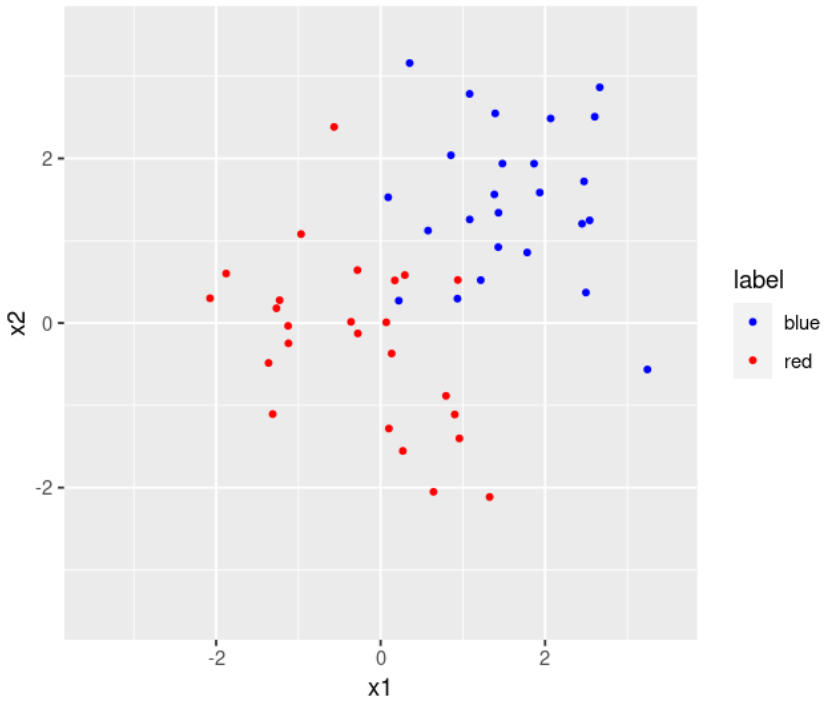
5. We now consider a classification problem. Suppose we have 2 classes (labels), 25 observations per class, and p = 2 features. We will call one class the "red" class and the other class the "blue" class. The observations in the red class are drawn i.i.d. from a $N_p(\mu_r, I)$ distribution, and the observations in the blue class are drawn i.i.d. from a $N_p(\mu_b, I)$ distribution, where

$\mu_r = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ is the mean in the red class, and where $\mu_b = \begin{pmatrix} 1.5 \\ 1.5 \end{pmatrix}$ is the mean in the blue class.

(a) Generate a training set, consisting of 25 observations from the red class and 25 observations from the blue class. (You will want to use the R function rnorm.) Plot the training set. Make sure that the axes are properly labeled, and that the observations are colored according to their class label.
The code below generates the training data.

```
library(ggplot2)
library(class)
library(ISLR2)
library(GGally)

set.seed(999)

n1 = 25

generate_df <- function(n1){
  # function to generate data for Q5
  x1 = rnorm(n1)
  x2 = rnorm(n1)
  label_red = rep("red", n1)
  df_red = data.frame(x1 = x1, x2 = x2, label = label_red)
  x1 = rnorm(n1, mean=1.5)
  x2 = rnorm(n1, mean=1.5)
  label_blue = rep("blue", n1)
  df_blue = data.frame(x1 = x1, x2 = x2, label = label_blue)

  X_df = rbind(df_red, df_blue)
  return (X_df)
}

# generate training data
X_df = generate_df(n1)

# plot a)
ggplot(X_df, aes(x1, x2)) +
  geom_point(aes(color=label), size=1) + xlim(-3.5, 3.5) + ylim(-3.5, 3.5) +
  scale_color_manual(values=c('blue', 'red')) + coord_fixed()
```

Plotting the training data, we get the plot shown below.



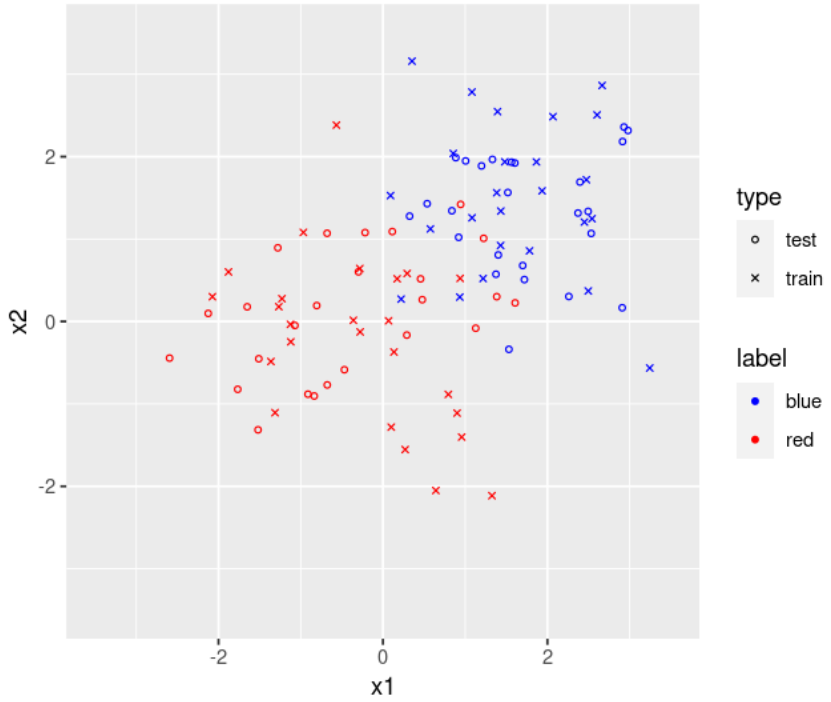
Plot of training data

(b) Now generate a test set consisting of 25 observations from the red class and 25 observations from the blue class. On a single plot, display both the training and test set, using one symbol to indicate training observations (e.g. circles) and another symbol to indicate the test observations (e.g. squares). Make sure that the axes are properly labeled, that the symbols for training and test observations are explained in a legend, and that the observations are colored according to their class label.
The code below plots the training and test data.

```
# generate test data
test_df = generate_df(n1)

# add type field to denote train/test data
X_df["type"] = rep("train", 2*n1)
test_df["type"] = rep("test", 2*n1)

# combine train and test into one data frame
X_df = rbind(X_df, test_df)

# plot b)
```

```
ggplot(X_df, aes(x1, x2, shape=type, color=label)) +
  geom_point(size=1) +
  xlim(-3.5, 3.5) + ylim(-3.5, 3.5) +
  scale_color_manual(values=c('blue', 'red')) +
  scale_shape_manual(values = c(1, 4)) + coord_fixed()
```

Plot below shows the training and test data.



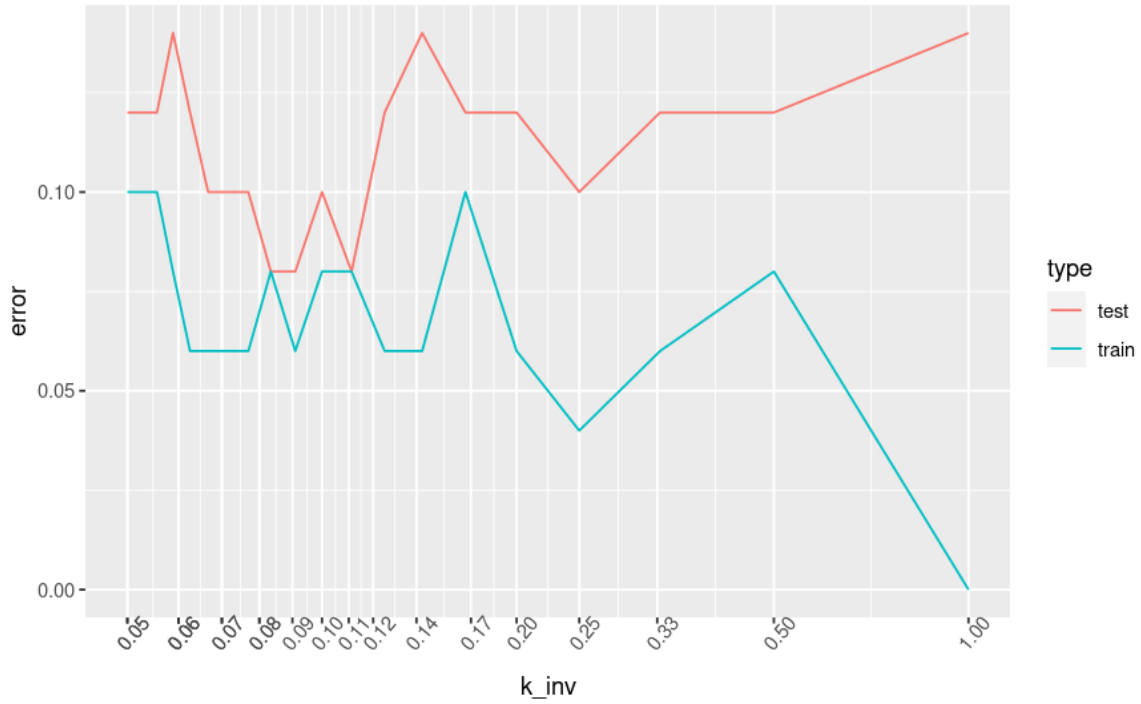
Plot of training and test data

(c) Using the knn function in the library class, fit a k-nearest neighbors model on the training set, for a range of values of k from 1 to 20. Make a plot that displays the value of $1/k$ on the x-axis, and classification error (both training error and test error) on the y-axis. Make sure all axes and curves are properly labeled. Explain your results.
The code to plot train and test error vs 1/k is given below.

```
# convert to numeric data type for prediction
X_df["y"] = as.numeric(X_df$label == "blue")

# get train and test data
train_df = X_df[X_df$type=="train",]
test_df = X_df[X_df$type=="test",]

K = 20
```

```
train_acc = rep(0, K)
test_acc = rep(0, K)
k_list = 1:K
temparray <- rep(0, (K*nrow(test_df)))
temparray = as.integer(temparray)
cl_test_arr <- array(temparray, dim = c(K, nrow(test_df)))
cl_train_arr <- array(temparray, dim = c(K, nrow(test_df)))

# run knn for different values of k
for(k in 1:K) {
  # knn classification on test data
  cl_test <- knn(train = train_df[, 1:2], test = test_df[, 1:2], cl = train_df$y, k =
  cl_test_arr[k,] = as.integer(cl_test) - 1
  test_acc[k] = mean(cl_test == test_df$y)
  # knn classification on train data
  cl_train <- knn(train = train_df[, 1:2], test = train_df[, 1:2], cl = train_df$y, k
  cl_train_arr[k,] = as.integer(cl_train) - 1
  train_acc[k] = mean(cl_train == train_df$y)
}

# create data frame to plot train, test errors

plot_df = rbind(data.frame(error = 1-train_acc, type="train", k_inv=1/k_list),
      data.frame(error = 1-test_acc, type="test", k_inv=1/k_list))

# plot (c)
ggplot(plot_df, aes(k_inv, error, group=type, col=type)) +
  geom_line() + scale_x_continuous(trans = 'log10', breaks = round(1/k_list, 2)) +
  theme(axis.text.x = element_text(angle = 50))
```

Plot below shows the training and test error vs 1/k.



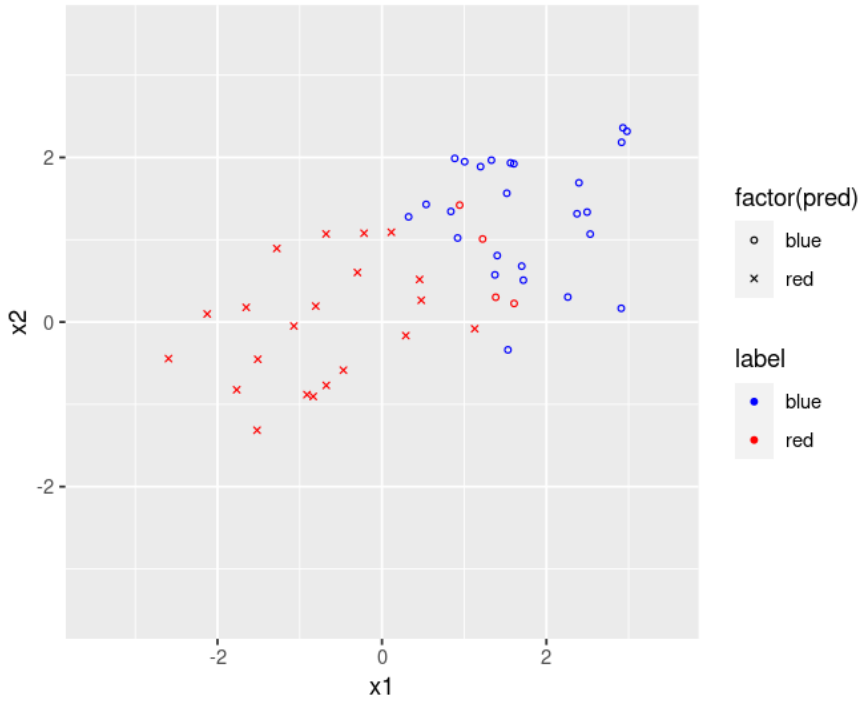
Train and test error vs k_inv

Training error is 0 when the model is most flexible i.e., $k = 1$. As the model becomes more flexible its performance on training set improves slightly but on test set it doesn't improve which suggests that the model isn't able to generalize well as it is fitting the noise instead of the data. Performance could be improved by addressing higher variance seen for more flexible models by instead choosing a simpler model.

(d) For the value of k that resulted in the smallest test error in part (c) above, make a plot displaying the test observations as well as their true and predicted class labels. Make sure that all axes and points are clearly labeled.
$k = 9$ is one of the values of k where the test error was the smallest. The code below plots the true test label vs the predictions made by KNN for $k = 9$.

```
k = 9
test_df["pred"] = ifelse(cl_test_arr[k,], "blue", "red")

p1 = ggplot(test_df, aes(x1, x2, shape=factor(pred), color=label)) +
  geom_point(size=1) +
  xlim(-3.5, 3.5) + ylim(-3.5, 3.5) +
  scale_color_manual(values=c('blue', 'red')) +
  scale_shape_manual(values = c(1, 4)) + coord_fixed()
p1
```

Plot below shows the true label and predicted labels on the test data when $k = 9$.



True and predicted labels on the test set for $k = 9$

(e) Recall that the Bayes classifier assigns an observation to the red class if $\mathbb{P}(Y = red|X = x) > 0.5$, and to the blue class otherwise. The Bayes error rate is the error rate associated with the Bayes classifier. What is the value of the Bayes error rate in this problem? Explain your answer.

Given any observation x_0, the Bayes Classifier will predict class j that maximizes $\mathbb{P}(Y = j|X = x_0)$.

From Bayes' theorem, we know that

$$\mathbb{P}(Y = red|X = x_0) = \frac{f_X(X = x_0|Y = red)\,\mathbb{P}(Y = red)}{f_X(X = x_0|Y = red)\,\mathbb{P}(Y = red) + f_X(X = x_0|Y = blue)\,\mathbb{P}(Y = blue)}$$

Since the number of data points in our training data have equal samples that are blue and red we can use this to create our prior distributions.

Therefore, $\mathbb{P}(Y = red) = \mathbb{P}(Y = blue) = 0.5$ Thus,

$$\begin{aligned}\mathbb{P}(Y = red|X = x_0) &= \frac{f_X(X = x_0|Y = red)0.5}{f_X(X = x_0|Y = red)0.5 + f_X(X = x_0|Y = blue)0.5} \\ &= \frac{f_X(X = x_0|Y = red)}{f_X(X = x_0|Y = red) + f_X(X = x_0|Y = blue)}\end{aligned}$$

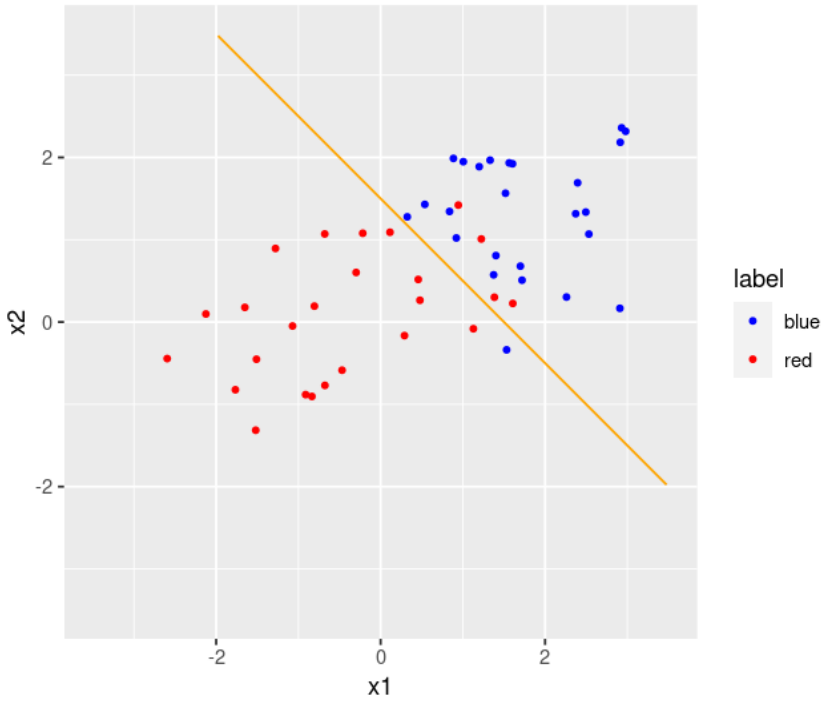
A similar calculation shows that,

$$\mathbb{P}(Y = blue|X = x_0) = \frac{f_X(X = x_0|Y = blue)}{f_X(X = x_0|Y = red) + f_X(X = x_0|Y = blue)}$$

Thus,

$$\max(\mathbb{P}(Y = red|X = x_0), \mathbb{P}(Y = blue|X = x_0)) =$$
$$\max(f_X(X = x_0|Y = red), f_X(X = x_0|Y = blue))$$

Since the pdfs of the observation given class are just the pdfs of the 2D normal distributions with the appropriate mean, for any observation, the predicted class is the one that has a higher pdf value at that point. Since the pdfs of the distribution of red and blue class differ only on the means, the two pdfs equal each other at points that are equidistant from the two means. This gives the Bayes decision boundary as the line between the two means shown in the figure below. All points such that $x_1 + x_2 > 1.5$ are predicted as blue and the rest as red.



Decision boundary of the Bayes Classifier is shown in orange.

Thus, this classifier will make an error whenever a point drawn from the blue distribution is below the line $x_1 + x_2 = 1.5$ or whenever a point drawn from the red distribution is

above the line $x_1 + x_2 = 1.5$. The Bayes error rate is thus given as,

$$\begin{aligned}
\mathbb{P}(error) &= \mathbb{P}(error|Y = red)\,\mathbb{P}(Y = red) + \mathbb{P}(error|Y = blue)\,\mathbb{P}(Y = blue) \\
&= \frac{1}{2}(\mathbb{P}(error|Y = red) + \mathbb{P}(error|Y = blue)) \\
&= \mathbb{P}(error|Y = red) && \text{by symmetry} \\
&= \int_{-\infty}^{\infty}\int_{\sqrt{4.5}/2}^{\infty} \frac{1}{2\pi}\exp\big(-(x^2+y^2)/2\big)\,dx\,dy \\
&= 0.144422
\end{aligned}$$

This result can be validated using simulations as well.

```
# number of samples to simulate
n1 = 100000
# generate data sampled from the red and blue distributions
X_df_gen = generate_df(n1)
# get predictions made by the Bayes classifier
X_df_gen["pred"] = ifelse(X_df_gen$x1+X_df_gen$x2>1.5, "blue", "red")
# Bayes error
1-mean(X_df_gen$label == X_df_gen$pred)
[1] 0.144215
```

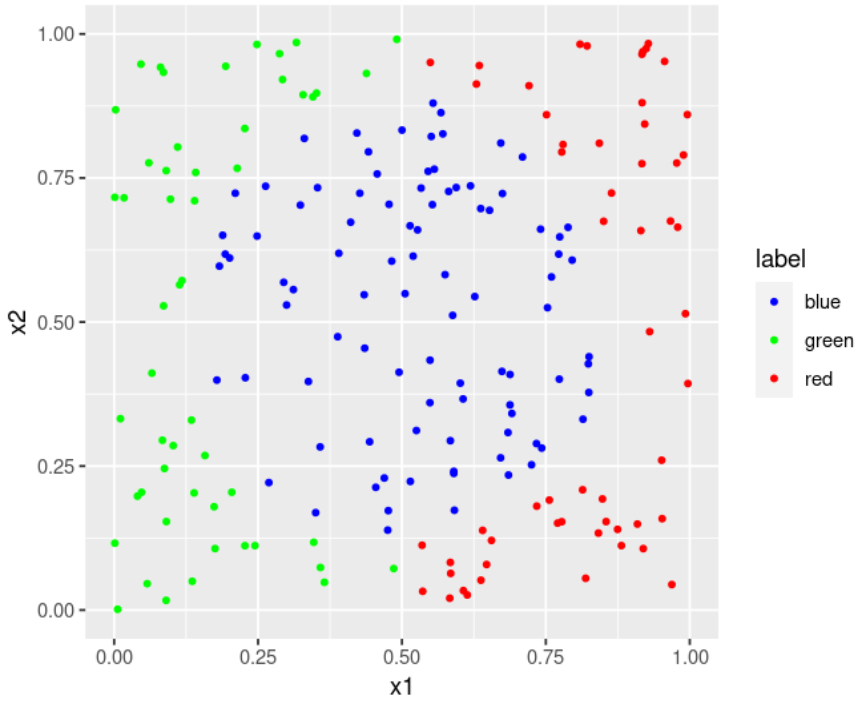
6. We will once again perform k-nearest-neighbors in a setting with $p = 2$ features. But this time, we'll generate the data differently: let $X_1 \sim Unif[0, 1]$ and $X_2 \sim Unif[0, 1]$, i.e. the observations for each feature are i.i.d. from a uniform distribution. An observation belongs to class "red" if $(X_1 - 0.5)^2 + (X_2 - 0.5)^2 > 0.15$ and $X_1 > 0.5$; to class "green" if $(X_1 - 0.5)^2 + (X_2 - 0.5)^2 > 0.15$ and $X_1 \leq 0.5$; and to class "blue" otherwise.

   (a) Generate a training set of $n = 200$ observations. (You will want to use the R function runif.) Plot the training set. Make sure that the axes are properly labeled, and that the observations are colored according to their class label.
   The code below generates the training data.

```
set.seed(NULL)
set.seed(99)
# size of training data
n1 = 200

get_label <- function(x1, x2){
  # function that generates the label given x1, x2
  d = (x1-0.5)^2+(x2-0.5)^2
  if ( d>0.15 && x1>0.5 ){
```

```
    return ("red")
  }
  else if ( d>0.15 && x1<=0.5 ) {
    return ("green")
  }
  else {
    return ("blue")
  }
}

generate_df_2 <- function(){
  # function that generates training data in Q6
  x1 = runif(n1)
  x2 = runif(n1)

  X_df = data.frame(x1=x1, x2=x2, label=mapply(get_label, x1, x2))

  return (X_df)
}

X_df = generate_df_2()

# plot a)
ggplot(X_df, aes(x1, x2)) +
  geom_point(aes(color=label), size=1) + xlim(0, 1) + ylim(0, 1) +
  scale_color_manual(values=c('blue', 'green', 'red')) + coord_fixed()
```

Plotting the training data, we get the plot shown below.



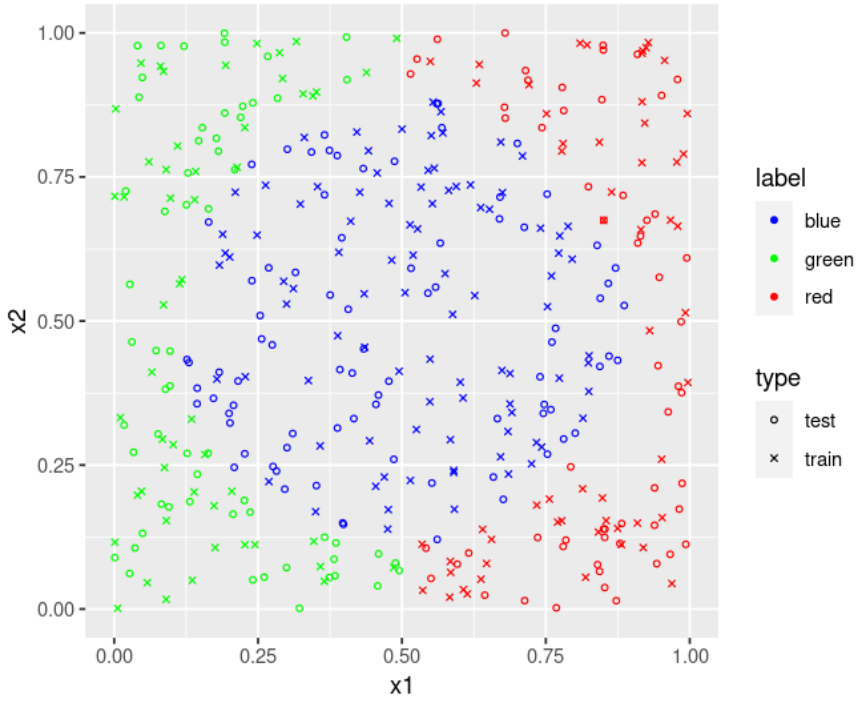
Plot of training data

(b) Now generate a test set consisting of another 200 observations. On a single plot, display both the training and test set, using one symbol to indicate training observations (e.g. circles) and another symbol to indicate the test observations (e.g. squares). Make sure that the axes are properly labeled, that the symbols for training and test observations are explained in a legend, and that the observations are colored according to their class label. The code below plots the training and test data.

```
test_df = generate_df_2()

# add type field to denote train/test data
X_df["type"] = rep("train", n1)
test_df["type"] = rep("test", n1)

# combine train and test data in one data frame
X_df = rbind(X_df, test_df)

# plot b)
ggplot(X_df, aes(x1, x2, shape=type, color=label)) +
  geom_point(size=1) +
  xlim(0, 1) + ylim(0, 1) +
  scale_color_manual(values=c('blue', 'green', 'red')) +
```

```
  scale_shape_manual(values = c(1, 4)) + coord_fixed()
```

Plot below shows the training and test data.



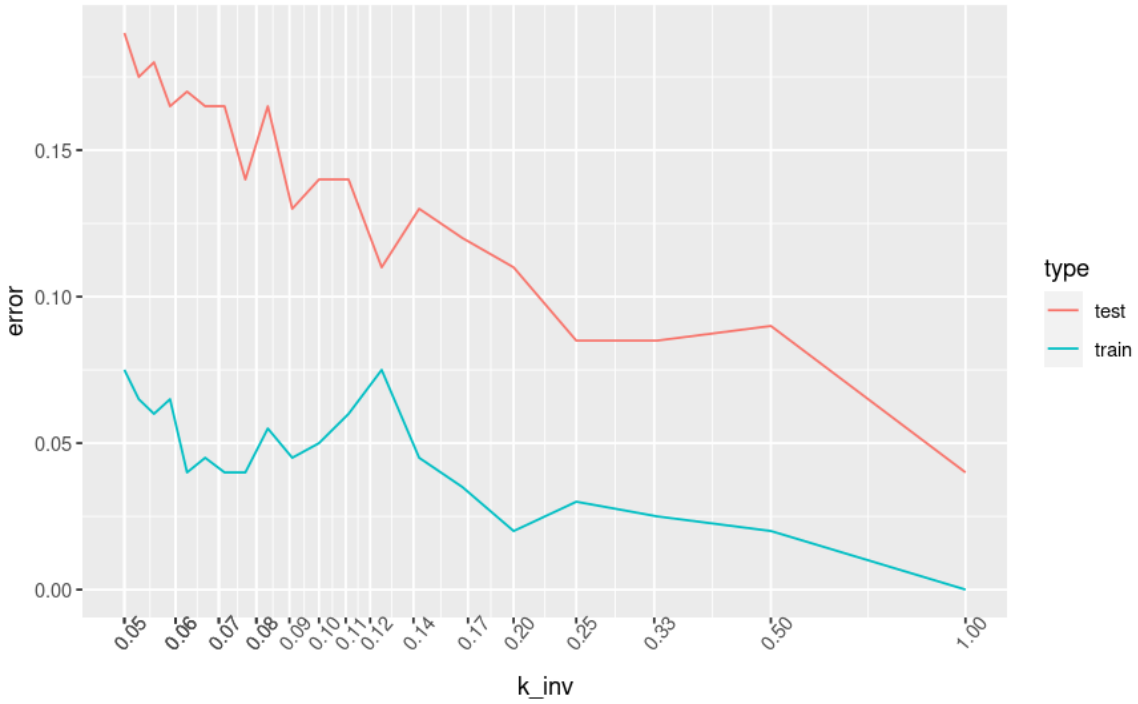
Plot of training and test data

(c) Using the knn function in the library class, fit a k-nearest neighbors model on the training set, for a range of values of k from 1 to 50. Make a plot that displays the value of $1/k$ on the x-axis, and classification error (both training error and test error) on the y-axis. Make sure all axes and curves are properly labeled. Explain your results.
The code to plot train and test error vs 1/k is given below.

```
# convert categorical variable to numeric
X_df["y"] = unclass(X_df$label)

# get train and test data
train_df = X_df[X_df$type=="train",]
test_df = X_df[X_df$type=="test",]

K = 20
train_acc = rep(0, K)
test_acc = rep(0, K)
k_list = 1:K
temparray <- rep(0, (K*nrow(test_df)))
```

```
temparray = as.numeric(temparray)
cl_test_arr <- array(temparray, dim = c(K, nrow(test_df)))
cl_train_arr <- array(temparray, dim = c(K, nrow(test_df)))

# run knn for different values of k
for(k in 1:K) {
  # knn classification on test data
  cl_test <- knn(train = train_df[, 1:2], test = test_df[, 1:2], cl = train_df$y, k =
  cl_test_arr[k,] = as.integer(cl_test) - 1
  test_acc[k] = mean(cl_test == test_df$y)
  # knn classification on train data
  cl_train <- knn(train = train_df[, 1:2], test = train_df[, 1:2], cl = train_df$y, k
  cl_train_arr[k,] = as.integer(cl_train) - 1
  train_acc[k] = mean(cl_train == train_df$y)
}

# create data frame to plot train, test errors

plot_df = rbind(data.frame(error = 1-train_acc, type="train", k_inv=1/k_list),
      data.frame(error = 1-test_acc, type="test", k_inv=1/k_list))

# plot (c)
ggplot(plot_df, aes(k_inv, error, group=type, col=type)) +
  geom_line() + scale_x_continuous(trans = 'log10', breaks = round(1/k_list, 2)) +
  theme(axis.text.x = element_text(angle = 50))
```

Plot below shows the training and test error vs 1/k.



Train and test error vs k_inv

Training error is 0 when the model is most flexible i.e., $k = 1$. Both train and test error decreases as the model flexibility increases which suggests that the model doesn't suffer from high variance but instead has high bias which can be addressed by using an even more flexible model.

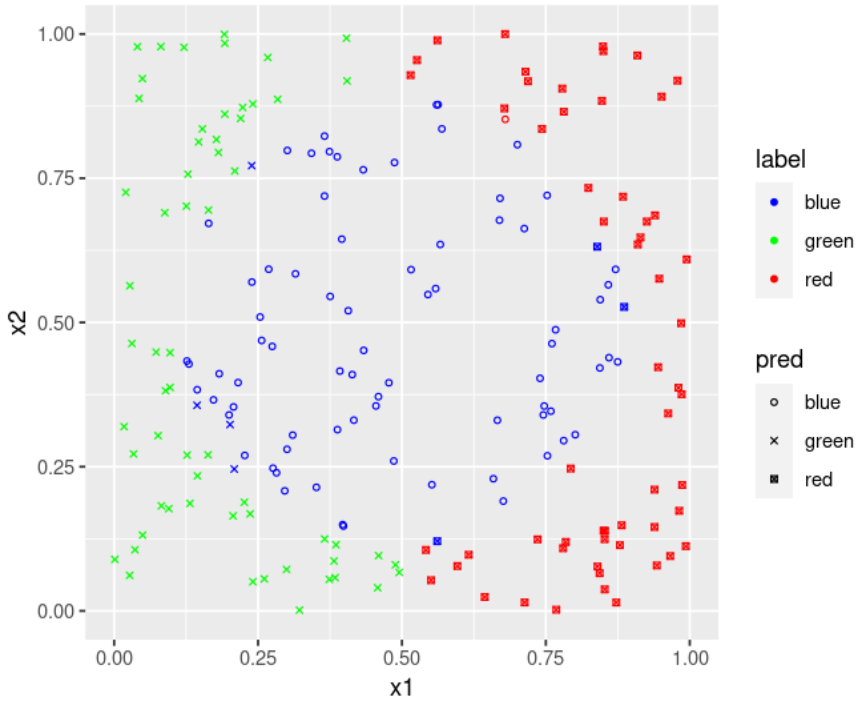
(d) For the value of k that resulted in the smallest test error in part (c) above, make a plot displaying the test observations as well as their true and predicted class labels. Make sure that all axes and points are clearly labeled.

Test error is the smallest when $k = 1$. The code below plots the true test label vs the predictions made by KNN for $k = 1$.

```
label_from_level = function(level) {
  # returns label given the corresponding level
  label = levels(cl_test)[level+1]
  return (label)
}


k = 1

# get predictions
test_df["pred"] = cl_test_arr[k,]
test_df["pred"] = lapply(test_df["pred"], label_from_level)
```

```r
# plot d)
ggplot(test_df, aes(x1, x2, shape=pred, color=label)) +
  geom_point(size=1) +
  xlim(0, 1) + ylim(0, 1) +
  scale_color_manual(values=c('blue', 'green', 'red')) +
  scale_shape_manual(values = c(1, 4, 7)) + coord_fixed()
```

Plot below shows the true label and predicted labels on the test data when $k = 1$.



True and predicted labels on the test set when $k = 1$

(e) In this example, what is the Bayes error rate? Justify your answer, and explain how it relates to your findings in (c) and (d).

In this example the Bayes error rate is exactly 0%. This is because we know that the data is generated using a determinisitic rule based on its coordinates (x, y) that is given as the rule in the question definition. Given any point (x, y), we know exactly which label it will map to due to this deterministic rule and the Bayes classifier that we construct would thus be able to predict the class with 100% accuracy. This also relates to our finding in (c) where we saw that the model suffers from high bias and not high variance and we saw the performance improve as we increased the model flexibility. A sufficiently flexible model would learn the deterministic rule defined in the problem statement given we have enough data. In (d) we see that the most flexible model makes the lowest error as it makes error

only near the decision boundary whereas points far from the decision boundary don't get misclassified as it is very likely to be surrounded by points from the same class. Reducing k also helps in this regard as the chance of encountering points from the other class increases as you increase k which increases the chance of misclassification.

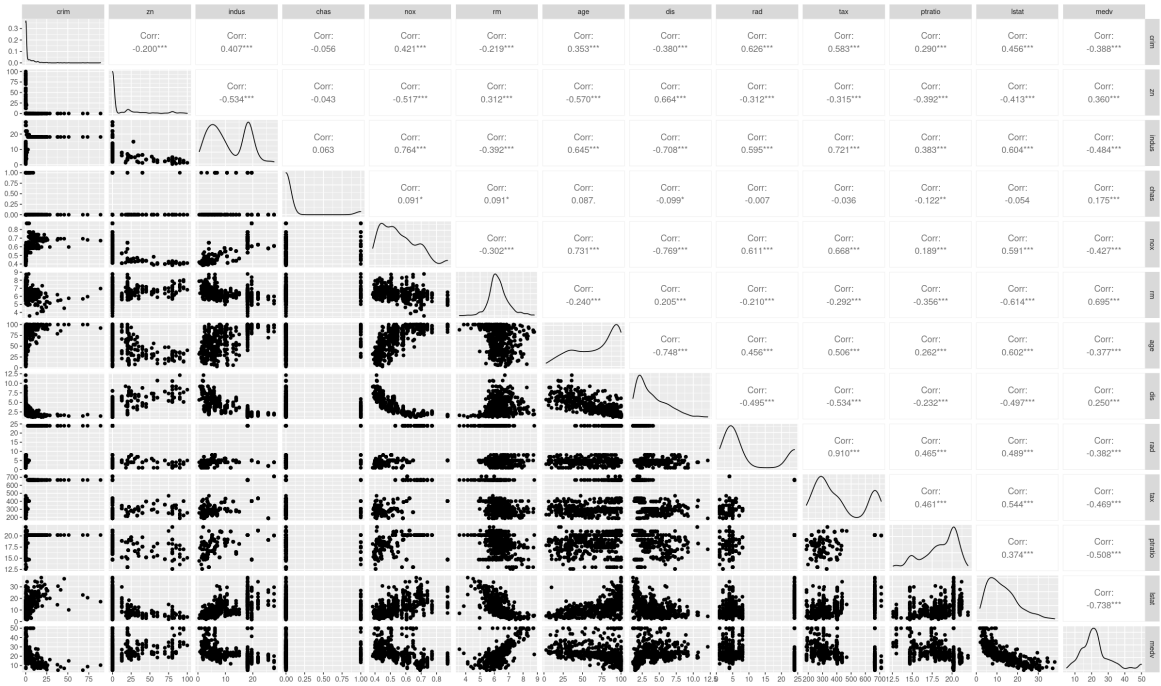
7. This exercise involves the Boston housing data set, which is part of the ISLR2 library.

   (a) How many rows are in this data set? How many columns? What do the rows and columns represent?

   There are 506 rows and 13 columns in the dataset. Each row corresponds to the features associated with a particular suburb and each column corresponds to one of the 13 features. Code used for this section is given below.

```
c(nrow(Boston), ncol(Boston))
```

   (b) Make some pairwise scatterplots of the predictors (columns) in this data set. Describe your findings.



pairwise scatter plot of predictors

The following combinations of features are positively correlated:
nox and indus, nox and age, rm and medv, age and lstat, tax and rad
The following combinations of features are negatively correlated:
rm and lstat, medv and lstat, dis and nox, age and dis, dis and indus
Code used for this section is given below.

```
ggpairs(Boston)
```

(c) Are any of the predictors associated with per capita crime rate? If so, explain the relationship.
'tax'(property tax rates) and 'rad'(accessibility to radial highways) are positively correlated with per capita crime rates.

(d) Do any of the suburbs of Boston appear to have particularly high crime rates? Tax rates? Pupil-teacher ratios? Comment on the range of each predictor.
Some suburbs have particularly high crime rates and tax rates (maximum values are much higher than the median). Whereas the highest pupil-teacher ratio isn't too different from the median value.
Per capita crime rates ranges from 0 to 89.
Tax rate per $10,000 ranges from 187 to 711.
Pupil-teacher ratio ranges from 12.6 to 22.
Code used for this section is given below.

```
summary(Boston["crim"])
summary(Boston["tax"])
summary(Boston["ptratio"])
```

(e) How many of the suburbs in this data set bound the Charles river?
35 suburbs in this data set bound the Charles river.
Code used for this section is given below.

```
sum(Boston["chas"])
```

(f) What are the mean and standard deviation of the pupil-teacher ratio among the towns in this data set?
Mean is 18.45 and standard deviation is 2.16.
Code used for this section is given below.

```
c(mean(Boston$ptratio), sd(Boston$ptratio))
```

(g) Which suburb of Boston has highest median value of owner-occupied homes? What are the values of the other predictors for that suburb, and how do those values compare to the overall ranges for those predictors? Comment on your findings.
16 suburbs of Boston have the highest median value of owner-occupied homes of $50,000. For these suburbs we see higher than median values of per capita crime rate, average number of rooms per dwelling and lower than median values of lower status of the population (percent).
Code used for this section is given below.

```
summary(Boston)
Boston[Boston["medv"]==50,]
```

(h) In this data set, how many of the suburbs average more than six rooms per dwelling? More than eight rooms per dwelling? Comment on the suburbs that average more than eight rooms per dwelling.

333 suburbs average more than six rooms per dwelling.

13 suburbs average more than eight rooms per dwelling.

These suburbs also have higher than average per capita crime rates, higher median value of owner occupied homes and lower than average lstat (lower status of the population in percentage)

Code used for this section is given below.

```
summary(Boston)
nrow(Boston[Boston["rm"]>6,])
nrow(Boston[Boston["rm"]>8,])
Boston[Boston["rm"]>8,]
```