Actuarial Data Analytics I – Subject Notes

Omar Amin

Contents

1 Overview of Statistical Learning and Data Analytics

1.1 What is Statistical Learning?

Statistical learning refers to *"a set of tools for making sense of complex datasets"*. In actuarial data analytics, these tools help us build models and algorithms to understand data and make predictions. Statistical learning encompasses both **supervised learning** and **unsupervised learning**:

- **Supervised learning**: We have input variables (features) $X_1, X_2, \ldots, X_p$ and an output variable Y. The goal is to predict Y or understand its relationship with X. If Y is quantitative (numeric), it is a **regression** problem; if Y is qualitative (categorical), it is a **classification** problem.
- **Unsupervised learning**: We only have features $X_1, \ldots, X_p$ and no explicit Y. The goal is to find patterns or groupings in the data without known responses. Examples include discovering clusters or underlying structure.

In supervised learning, the aim can be either **prediction** (accurately predicting unseen outcomes Y) or **inference** (understanding the effect of each X_j on Y). We often split data into a **training set** and a **test set** to evaluate model performance on new data. A key concept is the **bias-variance trade-off**: as model flexibility increases, its bias tends to decrease but variance tends to increase. An optimal model achieves a good balance, minimizing the expected test error.

1.2 Bias-Variance Trade-off and Model Accuracy

A more restrictive model: more interpretable, easier for the inference goal. lasso > linear models > GAM, trees > bagging, boosting. A more flexible model: better prediction accuracy, more complicated so harder to understand how each predictor behaves in the model. bagging, boosting > GAM, trees > linear models > lasso.

The **training error** (error on the data used to fit the model) typically decreases as model complexity increases, but the **test error** (error on new data) follows a U-shape pattern. Initially, increasing flexibility reduces bias faster than it increases variance, so test error decreases; beyond a point, variance dominates and test error rises. This is why an overly complex model can overfit the training data and perform poorly on new data. In general higher flexibility $\implies$ higher variance and less bias.

We quantify prediction error as mean squared error for regression or mis-classification rate for classification. The lowest possible error is achieved by the theoretical **Bayes classifier** or function $f(x)$ (which we usually don't know). Our goal is to approximate this with a model $\hat{f}(x)$. The irreducible error (noise) sets a lower bound on what error we can achieve.

Key definitions:

- **Bias**: error from erroneous assumptions in the learning algorithm. High bias means model is too simple to capture the true $f(x)$.
- **Variance**: error from sensitivity to small fluctuations in the training set. High variance means model is too complex (over fitting noise).
- **Mean Squared Error (MSE)** decomposition (for regression): $\mathbb{E}[(Y-\hat{f}(X))^2] = (\text{Bias})^2 + \text{Variance} + \text{Irreducible Error}$. We seek to minimise bias+variance. $MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{f(\boldsymbol{x}_i)})^2$.

2 Linear/Multiple Linear Regression

2.1 Simple Linear Regression Theory

Linear regression is a **supervised learning** method for predicting a quantitative response Y based on one or more predictors $X_1, \ldots, X_p$. In **simple linear regression**, we have a single predictor X. The model assumes an approximate linear relationship:

$$\hat{Y} = \beta_0 + \beta_1 X,$$

where β_0 is the intercept and β_1 is the slope. β_1 represents the expected change in Y for a one-unit increase in X. The model is usually fit by **ordinary least squares (OLS)**, which finds $\hat{\beta}_0, \hat{\beta}_1$ that minimize the sum of squared residuals:

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_i)^2.$$

The solutions are:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}, \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x},$$

where $\bar{x}$ and $\bar{y}$ are the sample means. Key formulas:

- **Predicted value:** $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$.
- **Residual:** $e_i = y_i - \hat{y}_i$.
- **Coefficient standard errors**: $\text{SE}(\hat{\beta}_1) = \sqrt{\frac{\sigma^2}{\sum (x_i - \bar{x})^2}}$, and similarly for $\text{SE}(\hat{\beta}_0) = \sigma \sqrt{\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2}}$ (where σ^2 is estimated error variance).
- **R-squared (R^2) and RSE**: The proportion of variance in Y explained by X: $R^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$, where $\text{TSS} = \sum (y_i - \bar{y})^2$. Also, Residual Standard Error (RSE) $= \sqrt{RSS/(n-2)}$. The 95% CI for $\hat{\beta}_i$ is $\hat{\beta}_i \pm 1.96 \times \text{SE}(\hat{\beta}_i)$, $i = 0, 1$.

Purpose and Description: Linear regression models the relationship between X and Y with a line. It's easy to interpret: β_1 tells the direction and magnitude of effect of X on Y. We can use it for prediction and to quantify relationships (with confidence intervals and hypothesis tests for β_1).

Pros:
Simple, interpretable, fast to fit, inference tools available (t-tests, etc.). Works well if true relationship is roughly linear and error terms are homoscedastic (equal variance).
Cons:
Only captures linear trends (unless extended), sensitive to outliers (which can drastically affect the slope), and performance degrades if relationship is non-linear or if there are important interaction effects not included. Violations of assumptions (e.g. non-constant variance, correlated errors) can invalidate inference.

2.2 Simple Linear Regression in R

```
library(MASS)
library(ISLR)

lm.fit=lm(medv~lstat,data=Boston)
summary(lm.fit)
coef(lm.fit)
# RSS (three different approaches)
```

```r
deviance(lm.fit)
sum(resid(lm.fit)^2)
sum((medv - predict(lm.fit, Boston)) ^ 2)
# Training MSE
mean((medv - predict(lm.fit, Boston)) ^ 2)
# CI for coefficients
confint(lm.fit)
# confidence interval
predict(lm.fit,data.frame(lstat=(c(5,10,15))), interval="confidence")
# prediction interval
predict(lm.fit,data.frame(lstat=(c(5,10,15))), interval="prediction")

# Graph linear relationship
plot(lstat,medv)
abline(lm.fit,lwd=3,col="red")
```

Explanation: We load the `MASS` library and the `Boston` dataset. The function `lm(medv ~ lstat, data=Boston)` fits a linear model $\text{medv} = \beta_0 + \beta_1 \text{lstat} + \epsilon$. The result `lm_fit` contains the fitted coefficients and other information. `summary(lm_fit)` prints the model summary, including estimates $\hat{\beta}_0, \hat{\beta}_1$, their standard errors, t-statistics and p-values (testing $H_0 : \beta_j = 0$), R^2, etc. For example, output might show:

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 34.5538    0.5627   61.42   <2e-16 ***
lstat       -0.9505    0.0387  -24.57   <2e-16 ***
```

This indicates $\hat{\beta}_0 \approx 34.55$ and $\hat{\beta}_1 \approx -0.95$. So the fitted equation is $\widehat{\text{medv}} = 34.55 - 0.95 \times \text{lstat}$. The negative slope implies that higher `lstat` (more low-status population) is associated with lower median home value. The p-value is $< 2e-16$, indicating the relationship is statistically significant.

We can also obtain the coefficients directly with `coef(lm_fit)` or confidence intervals for them using `confint(lm_fit)`.

2.3 Multiple Linear Regression Theory

Multiple linear regression generalizes to p predictors. The model:

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \varepsilon,$$

fitted by least squares minimizing $\text{RSS} = \sum_{i=1}^{n} (y_i - \hat{\beta}_0 - \hat{\beta}_1 x_{i1} + ... + \hat{\beta}_p x_{ip})^2$. Interpretation of coefficients: β_j is the expected change in Y for a one-unit increase in X_j, holding all other predictors fixed.

Important considerations:

- **t-tests**: We test $H_0 : \beta_j = 0$ for each coefficient (given others in model). A small p-value indicates predictor X_j contributes significantly (after accounting for others).
- **F-test**: A global test whether at least one $\beta_j \neq 0$. It compares the full model to the intercept-only model (answers if atleast one of the predictors are useful in predicting the response).
- **Dummy variables**: Qualitative predictors are incorporated via dummy (indicator) variables. For example, a categorical variable with k classes is encoded with $k - 1$ dummy binary variables.

- **Interactions**: We can include interaction terms (e.g. $X_1 \times X_2$) to allow the effect of one predictor to depend on another.
- **Model fit**: R^2 still indicates proportion of variance explained (it always increases when adding predictors). Adjusted R^2 penalizes adding useless predictors.

2.3.1 Important questions to answer with MLR

1. Is at least one of the predictors $X_1, X_2, ..., X_p$ useful in predicting the response?
   (a)

Pros:
Captures linear effects of multiple variables, enabling control for confounders. Inference allows testing each predictor's effect. The model is easy to implement and interpret (coefficients as marginal effects).
Cons:
Still assumes linear form, which may be too restrictive. Sensitive to multicollinearity (high correlation among predictors can inflate standard errors and make coefficient estimates unstable). Outliers or high-leverage points can distort the fit. If p is large relative to n, overfitting can occur.

2.4 Multiple Linear Regression in R

```
# ****************************Multiple Linear Regression**************************
lm.fit=lm(medv~lstat+age,data=Boston)
summary(lm.fit)
# fit using all predictors
lm.fit=lm(medv~.,data=Boston)
summary(lm.fit)
library(car)
vif(lm.fit) # variance inflation factor
# fit using all predictors except "age"
lm.fit1=lm(medv~.-age,data=Boston)
summary(lm.fit1)
# an alternative method of excluding one predictor
lm.fit1=update(lm.fit, ~.-age)

# ********************************Interaction Terms*********************************
summary(lm(medv~lstat*age,data=Boston))

# ******************Non-linear Transformations of the Predictors******************
lm.fit2=lm(medv~lstat+I(lstat^2))
summary(lm.fit2)
lm.fit=lm(medv~lstat)
anova(lm.fit,lm.fit2)
par(mfrow=c(2,2))
plot(lm.fit2)
lm.fit5=lm(medv~poly(lstat,5))
summary(lm.fit5)
summary(lm(medv~log(rm),data=Boston))

# ********************************Qualitative Predictors****************************
fix(Carseats)
names(Carseats)
lm.fit=lm(Sales~.+Income:Advertising+Price:Age,data=Carseats)
summary(lm.fit)
attach(Carseats)
```

```
contrasts(ShelveLoc)
```

Diagnostics: We can examine diagnostic plots for linear regression:

```
par(mfrow=c(2,2))
plot(lm_fit_full)
```

This produces residual plots, Q-Q plot for residuals, scale-location plot, and leverage plot. We look for non-linear patterns (indicating model mis-specification), non-constant variance, outliers, or high leverage points.

2.5 Polynomial Regression and Model Flexibility

Often the linearity assumption is too restrictive. One way to model a non-linear relationship is to include transformations of predictors. **Polynomial regression** adds polynomial terms. For example, a quadratic model:

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \varepsilon,$$

can capture curvature. Higher-degree polynomials (cubic, quartic, etc.) provide more flexibility.

In R, we can use `poly()` or the $I()$ function to include polynomial terms. For example:

```
lm_poly4 <- lm(medv ~ poly(lstat, 4), data=Boston) # or
lm_poly4_raw <- lm(medv ~ lstat + I(lstat^2) + I(lstat^3) + I(lstat^4), data=Boston)
```

includes powers of `lstat` explicitly (using $I()$ to inhibit formula interpretation). Polynomial terms are just one form of basis functions; others include piecewise polynomials and splines.

Including polynomial terms increases model flexibility (reduces bias) but also risk of over fitting (increased variance). One can use cross-validation (see below) to select the polynomial degree that optimises test error.

2.6 Other Considerations in the Regression Model

2.6.1 Qualitative predictors

For predictors with only two levels, we can define a new dummy variable:

$$X' = \begin{cases} 1 & \text{if } X \text{ is female} \\ 0 & \text{if } X \text{ is male} \end{cases}$$

and use X' as a predictor in the regression equation. We have:

$$Y = \beta_0 + \beta_1 X' + \varepsilon \implies y_i = \begin{cases} \beta_0 + \beta_1 + \varepsilon_i & \text{if } i\text{th person is female} \\ \beta_0 + \varepsilon_i & \text{if } i\text{th person is male} \end{cases}$$

So β_0 is the average of Y given they are male and $\beta_0 + \beta_1$ is the average of Y given they are female. Can fit this in R and check p-value to see if its a statistically significant change or not.

For qualitative predictors with more than two levels, we define additional dummy variables to include the extra levels. In general, m levels will need $m-1$ dummy variables.

- $X'_i = 1$ & $X'_j = 0,\ j = 1, ..., m-1,\ j \neq i$, corresponds to the original predictor taking level $i,\ i = 1, ..., m-1$;
- $X'_i = 0,\ i = 1, ..., m-1$, corresponds to the mth level;
- For each observation, at most one of these dummy variables can take value 1.

We have:

$$
\begin{aligned}
& Y = \beta_0 + \beta_1 X_1' + ... + \beta_{m-1} X_{m-1}' + \varepsilon \\
\implies & y_i = \begin{cases} \beta_0 + \beta_1 + \varepsilon_i & \text{if observation } i \text{ is at level 1} \\ \vdots & \vdots \\ \beta_0 + \beta_{m-1} + \varepsilon_i & \text{if observation } i \text{ is at level } m-1 \\ \beta_0 + \varepsilon_i & \text{if observation } i \text{ is at level } m \end{cases}
\end{aligned}
$$

The coding rule is arbitrary.

2.6.2 Extension of the linear model

Extending the linear regression model we can have 2-variable regression model,

$$
\begin{aligned}
Y &= \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + \varepsilon \\
&= \beta_0 + (\beta_1 + \beta_3 X_2) X_1 + \beta_2 X_2 + \varepsilon, \; X_2 \text{ impacts how much } X_1 \text{ will impact } Y
\end{aligned}
$$

The model above does not only have two main effects, but also includes an interaction effect, where on unit increase in X_1 will not always produce the same amount of increase in Y. Instead, the amount of increase is a function of X_2, and vice versa.

Can check statistical significance of the interaction term by regressing with it in R then checking resulting p-value of that predictor.

Interaction effect between quantitative and qualitative variables:
Suppose we are considering the folllowing multiple linear model:

$$
\begin{aligned}
\text{balance}_i &= \beta_0 + \beta_1 \times \text{income}_i + \beta_2 \times I_{\{\text{person } i \text{ is a student}\}} + \varepsilon_i \\
&\approx \beta_1 \times \text{income}_i + \begin{cases} \beta_0 + \beta_2 & \text{if } i\text{th person is a student} \\ \beta_0 & \text{if } i\text{th person is not a student} \end{cases}
\end{aligned}
$$

Note that we are fitting two parallel lines to the data, one for students and one for non-students. These two lines have different intercepts. There is a potential limitation in the model, since a change in incoe may have very different effect on the credit card balance of a student versus a non-student.

The above limitation can be addressed by adding an interaction variable, i.e. income $\times I_{\{\text{person } i \text{ is a student}\}}$.

$$
\begin{aligned}
\text{balance}_i &= \beta_0 + \beta_1 \times \text{income}_i + \beta_2 \times I_{\{\text{person } i \text{ is a student}\}} + \beta_3 \times \text{income}_i \times I_{\{\text{person } i \text{ is a student}\}} + \varepsilon_i \\
&\approx \begin{cases} (\beta_0 + \beta_2) + (\beta_1 + \beta_3) \times \text{income}_i & \text{if } i \text{ is a student} \\ \beta_0 + \beta_1 \times \text{income}_i & \text{if } i \text{ is not a student} \end{cases}
\end{aligned}
$$

Here we are still fitting two different regression lines for the students and non-students. But those lines have different intercepts as well as different slopes.

2.6.3 Potential problems

- Non-linearity of the response-predictor relationship

  - The residual plots can be used to identify potential non-linearity. Ideally, the residual plot should show no discernible pattern.
- Correlation of error terms
  - The linear regression models have an important assumption that all error terms are uncorrelated.
  - If this was not the case, then the estimated standard errors would tend to underestimate the true standard errors.
  - As a result the confidence and prediction interval we would obtain are narrower than they should be
  - Example with correlated error terms include data with repetitive entries, time-series data, and etc.
- Non-constant variance of error terms
  - Another important assumption of the linear regression model is that the error terms have constant variance, i.e. $\text{Var}(\varepsilon_i) = \sigma^2$.
  - The standard errors, confidence intervals, and hypothesis test associated with the linear models rely on this assumption.
  - However, it is often the case that the variance of the error terms are non-constant, so called ***heteroscedasticity***. The variances may increase with the value of the response. A possible solution is to transform the response Y using a concave function such as $\sqrt{Y}$ or $\log(Y)$.
    * For residual plot (fitted values vs residuals) if funnel shape indicates heteroscedasticity.
- Outliers
  - An outlier is a point for which y_i is far from the value predicted by the model. Outliers may be caused by various reasons, e.g. incorrect recording of an observation during data collection process.
  - Residual plots can be used to identify outliers. They usually have unusually high residual values. However, in practice it can be hard to judge whether a particular is truly an outlier or not.
    * Studentized residuals (ε_i divided by estimated s.e.) can give better indications than normal residual plots. Just need to check whether the absolute value > 3 or not.
- High-leverage points
  - High-leverage observations have an unusual value of x_i.
  - These points can have significant impacts on the estimated regression line. This is a concern as any problems with these points may invalidate the entire fit.
  - So it is important to identify high leverage observations.
    1. Simple linear regression: predictor value is outside the normal range of observations. $h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{j=1}^{n}(x_j - \bar{x})^2}$.
    2. Multiple linear regression: we can compute the leverage statistic h_i and to see whether it is $>> (p+1)/n$. If it is the case, then the ith observation could have been high leverage.
- Collinearity
  - Two or more predictors are closely related to one another.
  - Collinearity can cause problems in the regression context, as it can be difficult to separate out the individual effects of collinear variables on the response.
  - Collinearity reduces the accuracy of the estimates of the regression coefficients, it causes the s.e. for $\hat{\beta}_j$ to grow. Consequently, collinearity results in a decline in the t-statistic. So we may fail to reject $H_0 : \beta_j = 0$. It means the power of the hypothesis test is reduced by collinearity.
  - A way to detect collinearity is to look at correlation matrix of the predictors. An

element with large absolute value in this matrix indicates a potential collinearity issue.
  - However, *multi-collinearity* can't always be detected using a correlation matrix, instead we calculate teh *variance inflation factor* (VIF), which is $\text{VIF}(\hat{\beta}_j) = \frac{1}{1-R^2_{x_j|x_{-j}}}$, where $R^2_{x_j|x_{-j}}$ is the R^2 from a regression of X_j onto all of the other predictors.
  - A VIF that > 5 or 10 indicates a problematic amount of collinearity.

2.7 Parametric vs Non-parametric Methods

Parametric methods:

- + Have explicit forms and easy to fit (usually small number of parameters to estimate
- + Easier to explain the results and to test model significance
- − Based on strong model assumptions on the form of $f(X)$ that could be wrong
- − Usually less predicting power than non-parametric methods
- General rule, parametric methods tend to outperform non-parametric methods when there is a small number of observations per predictor (relative sample size)

Non-parametric methods:

- + No need to make assumptions on the form of $f(X)$
- + Usually more predicting power than parametric methods
- − Hard to explain model fitting results and to test model significance

2.8 Additional Graphical and Numerical Summarises

```
library(ISLR)
attach(Auto)
plot(cylinders, mpg)
cylinders=as.factor(cylinders)
plot(cylinders, mpg)
plot(cylinders, mpg, col="red")
plot(cylinders, mpg, col="red", varwidth=T)
plot(cylinders, mpg, col="red", varwidth=T,horizontal=T)
plot(cylinders, mpg, col="red", varwidth=T, xlab="cylinders", ylab="MPG")
hist(mpg,col=2)
hist(mpg,col=2,breaks=15)
pairs(Auto) #pairwise scatterplots
Auto$name=as.factor(Auto$name)
pairs(Auto)
pairs(~ mpg + displacement + horsepower + weight + acceleration, Auto)
#pairwise scatterplots of chosen variables
plot(horsepower,mpg)
identify(horsepower,mpg,name)
```

3 Resampling Methods

Resampling methods like cross-validation help determine model complexity. They do not create new models but provide a way to assess a given model's performance on unseen data.

3.1 Validation Set Approach

Randomly dividing the available set of observations into two parts, a *training set* and a *validation set*. The model is fit on the training set, and the fitted model is used to predict the response for the observations in the validation set.

Potential drawbacks:

- The validation estimate of the test error rate can be highly variable.
- Only a subset of the observations are used to fit the model. Therefore, the validation set error rate may tend to overestimate the test error rate for the model fit on the entire data set.

3.1.1 Validation set approach in R

```
library(ISLR)
attach(Auto)

# First way of dividing data
set.seed(1) # change seed for different train/test sets
train=sample(392,196) #generate training index subset
lm.fit=lm(mpg~horsepower,data=Auto,subset=train)
mean((mpg-predict(lm.fit,Auto))[-train]^2) # test MSE
mean((mpg[-train]-predict(lm.fit,Auto[-train,]))^2) # alternative
# quadratic regression
lm.fit2=lm(mpg~poly(horsepower,2),data=Auto,subset=train)
mean((mpg-predict(lm.fit2,Auto))[-train]^2)
# polynomial regression
lm.fit3=lm(mpg~poly(horsepower,3),data=Auto,subset=train)
mean((mpg-predict(lm.fit3,Auto))[-train]^2)

#Second way of dividing data
set.seed(1)
sample<-sample(c(TRUE, FALSE), nrow(Auto), replace=TRUE, prob=c(0.5,0.5))
train <- Auto[sample, ] #train is a training data set
test <- Auto[!sample, ] #test is a test data set
```

3.2 Leave-One-Out Cross Validation (LOOCV)

Involves splitting the set of observations into two parts, but with the validation set only containing a single observation.

1. Using the observation (x_1, y_1) as the validation set and $\{(x_2, y_2), ..., (x_n, y_n)\}$ as the training set to fit a model that generates a prediction $\hat{y}_1$ by x_1. Then $MSE_1 = (y_1 - \hat{y}_1)^2$.
2. Using observation (x_2, y_2) as the validation set and $\{(x_1, y_1), (x_3, y_3), ..., (x_n, y_n)\}$ as the training set to fit a model that generates a prediction $\hat{y}_2$ by x_2. Then $MSE_2 = (y_2 - \hat{y}_2)^2$.
3. Repeat this by selecting every remaining observation as the validation observation and obtain $MSE_3, ..., MSE_n$.

4. The LOOCV estiamte for the test MSE is then $CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n} MSE_i$. If n is large, this process is very time consuming.

Advantages of LOOCV:

- LOOCV has much less bias than the validation set approach, since when fitting the method the training sets almost cover the whole data set. So, LOOCV tends not to overestimate the test error rate.
- Validation set approach will yield different results when applied repeatedly. Performing LOOCV multiple times will always yield the same results, so no randomness.
- For least squares linear or polynomial regression, there is a computational shortcut. $CV_{(n)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{y_i-\hat{y}_i}{1-h_i}\right)^2$. h_i is the leverage statistic. In simple linear regression: $h_i = \frac{1}{n} + \frac{(x_i-\bar{x})^2}{\sum_{j=1}^{n}(x_j-\bar{x})^2}$.

3.2.1 LOOCV in R

```
# Automatically done through glm() and cv.glm()
# Use glm() to fit linear regression
glm.fit=glm(mpg~horsepower,data=Auto)
coef(glm.fit)
lm.fit=lm(mpg~horsepower,data=Auto)
coef(lm.fit)

library(boot)
glm.fit=glm(mpg~horsepower,data=Auto)
# cv.glm() without specifying K value does LOOCV
cv.err=cv.glm(Auto,glm.fit)
# the first value in 'delta' is average MSE of LOOCV;
# the second value is average MSE of LOOCV with bias correction
cv.err$delta

# Finding LOOCV error with different polynomial degrees
cv.error=rep(0,5)
for (i in 1:5){
  glm.fit=glm(mpg~poly(horsepower,i),data=Auto)
  cv.error[i]=cv.glm(Auto,glm.fit)$delta[1]
}
cv.error
```

3.3 K-Fold Cross-Validation

Alternative to LOOCV. LOOCV is special case of this when $k = n$. Steps:

1. Randomly dividing the set of observations into k groups, or folds, of approximately equal size.
2. First fold is treated as a validation set, and the method is fit on the remaining $k-1$ folds. Then find MSE_1 using the validation set.
3. Repeat step 1 and 2 by choosing different folds as the validation set. Calculate $MSE_2, ..., MSE_k$.
4. $CV_{(k)} = \frac{1}{k}\sum_{i=1}^{k} MSE_i$.

3.3.1 Bias-variance trade-off

Bias:
According to the amount of data used in fitting. Clearly, validation set approach has the highest

bias, and LOOCV has the lowest.

Variance:
k-fold CV tends to have the lower variance than LOOCV if $k < n$. Since LOOCV is averaging outputs of n highly correlated fitted models, while k-fold CV is averaging k less correlated outputs.

Overall, k-fold CV tends to give more accurate estimate of the test error rate than does LOOCV.

3.3.2 K-fold CV in R

```
set.seed(17)
cv.error.10=rep(0,10)
for (i in 1:10){
  glm.fit=glm(mpg~poly(horsepower,i),data=Auto)
  cv.error.10[i]=cv.glm(Auto,glm.fit,K=10)$delta[1] # set K=10 does 10-fold CV
}
cv.error.10
```

3.4 The Bootstrap

Used to quantify the uncertainty associated with a given estimator or statistical learning method. To assess accuracy of an estimator by sampling with replacement from the data and recalculating the estimator many times. The bootstrap can estimate the standard error of almost any statistic by emulating the process of getting new data from the population.

3.4.1 Bootstrap in R

```
# Bootstrap solution for the lecture example
?Portfolio # data set used
alpha.fn=function(data,index){
  X=data$X[index]
  Y=data$Y[index]
  return((var(Y)-cov(X,Y))/(var(X)+var(Y)-2*cov(X,Y)))
}
alpha.fn(Portfolio,1:100)
set.seed(1)
alpha.fn(Portfolio,sample(100,100,replace=T))
library(boot) # for boot function
boot(Portfolio,alpha.fn,R=1000) # data, function, repeats
boot(Portfolio,alpha.fn,R=10000)

# Estimating the Accuracy of a Linear Regression Model

# Define a statistic of interest, i.e. the linear regression coefficient estimates.
boot.fn=function(data,index){
  return(coef(lm(mpg~horsepower,data=data,subset=index)))}
# The original fit
boot.fn(Auto,1:392)

set.seed(1)
boot.fn(Auto,sample(392,392,replace=T)) # first bootstrap sample
boot.fn(Auto,sample(392,392,replace=T)) # second bootstrap sample
# Use boot() to run 1000 bootstrap calculations
boot(Auto,boot.fn,1000)
```

```r
summary(lm(mpg~horsepower,data=Auto))$coef

# Use bootstrap method to estimate the test MSE
boot.fn1=function(data,index){
  lm.fit=lm(mpg~horsepower,data=data,subset=index)
  return(mean((data$mpg - predict(lm.fit, data))[-index] ^ 2))}
boot.fn1(Auto,sample(392,392,replace=T))

set.seed(1)
boots.error.10=rep(0,10)
for(i in 1:10){
  boots.error.10[i]=boot.fn1(Auto,sample(392,392,replace=T))
}
boots.error.10


# Estimating the Accuracy of a quadratic Regression Model
boot.fn=function(data,index)
  return(coefficients(lm(mpg~horsepower+I(horsepower^2),data=data,subset=index)))
set.seed(1)
boot(Auto,boot.fn,2000)
summary(lm(mpg~horsepower+I(horsepower^2),data=Auto))$coef
```

4 Linear Model Selection and Regularisation

4.1 Linear Model Selection

So far we fit the standard linear model using the least squares method:

$$Y = \beta_0 + \beta_1 X_1 + ... + \beta_p X_p + \varepsilon$$

- *Good* when the true relationship between the response and the predictors is approximately linear, or if $n >> p$.
- *Bad* if n is not much larger than p (or $n < p$), the least squares method will under-perform.
- *Good/Bad* since in general a linear model is easy to interpret. However, when there are many predictors, some of them may not be correlated with the response (so we need to figure out which ones to remove which takes time).

4.2 Selecting Best Subset of Predictor

4.2.1 Best subset selection

Steps:

1. Let M_0 denote the *null* model (no predictors). This model simply predicts the sample mean for each observation
2. For $k = 1, 2, ..., p$:
   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors;
   (b) Pick the model with the smallest RSS (or largest R^2) among these $\binom{p}{k}$ models, and name it M_k.
3. Select the best model among $M_0, ..., M_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.
   - *Note:* In total there is 2^p models to work with. We choose the best model out of teh $p + 1$ local ones, we need to use appropriate criteria. RSS and R^2 cannot be used directly as M_p is always the best one to choose.

4.2.2 Best subset selection in R

```
library(ISLR)
?Hitters # dataset used

# Find out whether Salary has missing values and clean the data
sum(is.na(Hitters$Salary))
Hitters=na.omit(Hitters)

# Best Subset Selection
library(leaps)
# Build best subsets with up to 8 predictors (default option)
regfit.full=regsubsets(Salary~.,Hitters)
summary(regfit.full)
# Build best subsets with up to 19 predictors
regfit.full=regsubsets(Salary~.,data=Hitters,nvmax=19)
summary(regfit.full)
reg.summary=summary(regfit.full)
names(reg.summary)
reg.summary$rsq

# Plotting line graphs and best point
par(mfrow=c(2,2))
plot(reg.summary$rss,xlab="Number of Variables",ylab="RSS",type="l")
```

```r
plot(reg.summary$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="l")
# Find out the case with maximal adjusted R^2
which.max(reg.summary$adjr2)
# Draw points in the latest plot
points(11,reg.summary$adjr2[11], col="red",cex=2,pch=20)
plot(reg.summary$cp,xlab="Number of Variables",ylab="Cp",type='l')
which.min(reg.summary$cp)
points(10,reg.summary$cp[10],col="red",cex=2,pch=20)
plot(reg.summary$bic,xlab="Number of Variables",ylab="BIC",type='l')
which.min(reg.summary$bic)
points(which.min(reg.summary$bic),
    reg.summary$bic[which.min(reg.summary$bic)],col="red",cex=2,pch=20) # a better way

# The built-in plot() in regsubsets() gives weird visual graph
par(mfrow=c(1,1))
plot(regfit.full,scale="r2")
plot(regfit.full,scale="adjr2")
plot(regfit.full,scale="Cp")
plot(regfit.full,scale="bic") # best model has 6 predictors
coef(regfit.full,6)
```

4.2.3 Forward stepwise selection

Steps:

1. Let M_0 denote the *null* model, which contains no predictors.
2. For $k = 0, 2, ..., p-1$:
   (a) Consider all $p-k$ models that augment the predictors in M_k with one additional predictor;
   (b) Pick the model with the smallest RSS (or largest R^2) among these $p-k$ models, and anme it M_{k+1}.
3. Select the best model from among $M_0, ..., M_p$ using cross-validation prediction error, C_p (AIC), BIC, or adjusted R^2.
   - Algorithm works when $n < p$, however, it can only produce a best out of sub-models $M_0, ..., M_{n-1}$.

4.2.4 Forward stepwise selection in R

```r
regfit.fwd=regsubsets(Salary~.,data=Hitters,nvmax=19,method="forward")
summary(regfit.fwd) # *'s say use that predictor
coef(regfit.fwd,7) # could be diff to best subset or backward
```

4.2.5 Backward stepwise selection ($n > p$)

Steps:

1. Let M_p denote the *full* model, which contains all p predictors.
2. For $k = p, p-1, ..., 1$:
   (a) Consider all k models that contain all one of the predictors in M_k, for a total of $k-1$ predictors;
   (b) Pick the model with the smallest RSS (or largest R^2) among these k models, and name it M_{k-1}.
3. Select the best model from among $M_0, ..., M_p$ using cross-validation prediction error, C_p (AIC), BIC, or adjusted R^2.

- The stepwise method takes many less models into consideration $(1 + \frac{p(p+1)}{2})$ possibilities. Both methods are NOT guaranteed to find the *best* subset model.

4.2.6 Backward stepwise selection in R

```
regfit.bwd=regsubsets(Salary~.,data=Hitters,nvmax=19,method="backward")
summary(regfit.bwd) # *'s say use that predictor
coef(regfit.bwd,7) # could be diff to best subset or forward
```

4.2.7 A number of model selection criteria

We can use the following approaches to estimate the test error:

- **Direct approaches**:
  1. Validation set approach, or cross-validation approach which have been introduced earlier.
- **Indirect approaches**:
  1. $\boldsymbol{C_p}$: for a fitted least squares model containing d predictors, the C_p estimate for test MSE is $C_p = \frac{1}{n}(RSS + 2d\hat{\sigma}^2)$.
     - C_p is an unbiased estimate of the test MSE, so we choose the model with the lowest C_p value.
     - $\hat{\sigma}^2$ is an estimate of $\text{Var}(\varepsilon)$, and $2d\hat{\sigma}^2$ can be interpreted as a penalty term added to the training RSS, which can address the issue that the training error tends to underestimate the test error. This penalty term increases as d increases.
  2. **Akaike information criterion ($\boldsymbol{AIC}$)**: defined for a large class of models fit by the MLE method. In our case, for normal distributed ε, MLE and least squares are the same. $AIC = \frac{1}{n\hat{\sigma}^2}(RSS + 2d\hat{\sigma}^2) \propto C_p$.
     - In general, we choose the model that has the lowest AIC value.
  3. **Bayesian information criteria ($\boldsymbol{BIC}$)**: similar to C_p too. $BIC = \frac{1}{n\hat{\sigma}^2}(RSS + \log(n)d\hat{\sigma}^2)$
     - In general, we choose the model that has the lowest BIC value.
  4. **Adjusted $\boldsymbol{R^2}$**: for a fitted least squares model containing d predictors, the adjusted R^2 statistic is calculated as: adjusted $R^2 = 1 - \frac{RSS/(n-d-1)}{TSS/(n-1)}$.
     - A large value of adjusted R^2 indicates a model with a small test error.
     - $\frac{RSS}{n-d-1}$ is not a monotone function. This adjusted R^2 pays a price for the inclusion of unnecessary predictors in the model.

4.2.8 Choosing among models in R

```
# Create training set and test test
set.seed(1)
train=sample(c(TRUE,FALSE), nrow(Hitters),rep=TRUE)
test=(!train)
regfit.best=regsubsets(Salary~.,data=Hitters[train,],nvmax=19)
coef3=coef(regfit.best,id=3) # gets 3 best predictors coefficients


# direct method of finding the best model
test.mat=model.matrix(Salary~.,data=Hitters[test,])
val.errors=rep(NA,19)
for(i in 1:19){
  coefi=coef(regfit.best,id=i)
  pred=test.mat[,names(coefi)]%*%coefi
  val.errors[i]=mean((Hitters$Salary[test]-pred)^2)
```

```
}
val.errors
which.min(val.errors)
coef(regfit.best,which.min(val.errors)) # gets coefficients of number
# which gives smallest error also depends on seed used


# Create the function of prediction and find the best model
methods(predict) # show models predict() covers
####  VERY IMPORTANT FUNCTION ####
predict.regsubsets=function(object,newdata,id,...){
#... allows for other arguments to be passed into the function
  form=as.formula(object$call[[2]])
  mat=model.matrix(form,newdata)
  coefi=coef(object,id=id)
  xvars=names(coefi)
  mat[,xvars]%*%coefi
}
###################################

val.errors=rep(NA,19)
for(i in 1:19){
  pred=predict.regsubsets(regfit.best,Hitters[test,],id=i) #can also use predict()
  val.errors[i]=mean((Hitters$Salary[test]-pred)^2)
}
val.errors
which.min(val.errors)
coef(regfit.best,which.min(val.errors))

# Obtain the best subset model using the full data and CV selected id
regfit.best=regsubsets(Salary~.,data=Hitters,nvmax=19)
coef(regfit.best,which.min(val.errors))

# k-fold CV
k=10
set.seed(1)
folds=sample(1:k,nrow(Hitters),replace=TRUE)
cv.errors=matrix(NA,k,19, dimnames=list(NULL, paste(1:19)))
# NA means no data, NULL means no row names

for(j in 1:k){
  best.fit=regsubsets(Salary~.,data=Hitters[folds!=j,],nvmax=19)
  for(i in 1:19){
    pred=predict(best.fit,Hitters[folds==j,],id=i)
    cv.errors[j,i]=mean( (Hitters$Salary[folds==j]-pred)^2)
  }
}
mean.cv.errors=apply(cv.errors,2,mean)
mean.cv.errors
par(mfrow=c(1,1))
plot(mean.cv.errors,type='b')

# Obtain the best subset model using the full data and CV selected id
reg.best=regsubsets(Salary~.,data=Hitters, nvmax=19)
coef(reg.best,10) # 10 derived from lowest point in mean.cv.errors plot
```

4.3 Shrinkage Methods (Ridge/Lasso Regression)

We want to shrink the regression coefficients towards zero. These are the two best methods.

4.3.1 Ridge regression

Recall the least squares fitting procedure estimate $\hat{\beta}_0, ..., \hat{\beta}_p$ minimise:

$$RSS = \sum_{i=1}^{n} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + ... + \hat{\beta}_p x_{ip}))^2$$

Ridge regression coefficient estimates $\hat{\beta}_i^R,\ i = 0, ..., p$, are the values that minimise:

$$\sum_{i=1}^{n}(y_i - \hat{\beta}_0^R - \sum_{j=1}^{p} \hat{\beta}_j^R x_{ij})^2 + \lambda \sum_{j=1}^{p} (\hat{\beta}_j^R)^2 = RSS + \lambda \sum_{j=1}^{p} (\hat{\beta}_j^R)^2$$

where λ is a tuning parameter, to be determined separately.

The second term in the above objective function is the shrinkage penalty, which is small when $\hat{\beta}_i^R$ are close to 0. It has the effect of shrinking the estimates of β_j towards 0. The tuning parameter λ serves to control the relative impact of the RSS and the shrinkage penalty term.

- When $\lambda = 0$: ridge regression will produce the least squares estiamtes;
- When $\lambda \to \infty$: ridge regression coefficient estimates will approach zero.
- The ridge regression will produce a different set of coefficient estimates, $\hat{\beta}_\lambda^R$, for each value of λ.
- Let $||\beta||_2 = \sqrt{\sum_{j=1}^{p} \beta_j^2}$ denote teh l_2 norm of vector β, which measures the distance of β from 0. Then $||\hat{\beta}_\lambda^R||_2$ will always decrease as λ increases, and $||\hat{\beta}_\lambda^R||_2 / ||\hat{\beta}^R||_2$ ranges from 1 ($\lambda = 0$) to 0 ($\lambda = \infty$).
- Different from the least squares method, the term $\hat{\beta}_{j,\lambda}^R x_{ij}$ in the object function not only depends on λ, but also depends on the scaling of X_j. It may even depend on the scaling of other predictors. So it is best to apply ridge regression after standardising the predictors using the following formula, so that they are on the same scale: $\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \bar{x}_j)^2}}$.

Advantages of Ridge Regression Ridge regression's advantage over least squares is rooted in the bias-variance trade-off.

- As λ increases, the flexibility of the ridge regression fit decreases, leading to decreased variance but increased bias.
- If p is close to n or $p > n$, even when the relationship between the response and the predictors is close to linear, the least squares estimates will have high variance or even no solution. The ridge regression can still perform well by trading off a small increase in bias for a large decrease in variance. So the ridge regression works best when the least squares estimate have high variance.
- ridge regression also has substantial computational advantages over the best subset selection as for each fixed value of λ, ridge regression only fits a single model.

Disadvantages of Ridge Regression: It includes all p predictors in the final model (this disadvantage is overcome by the lasso model below).

4.3.2 Ridge regression in R

```r
library(ISLR)
library(glmnet) # FOR RIDGE
?Hitters # data set used

# Find out whether Salary has missing values and clean the data
sum(is.na(Hitters$Salary))
Hitters=na.omit(Hitters)

# Create data matrix which is required by the glmnet() function
# the [,-1] option is to exclude the intercept column in the x matrix
x=model.matrix(Salary~.,Hitters)[,-1]
y=Hitters$Salary

# generate 100 different values of the tuning parameter
grid=10^seq(10,-2,length=100)
ridge.mod=glmnet(x,y,alpha=0,lambda=grid) # Fit ridge regression
plot(ridge.mod)
# coef() stores all ridge regression parameter estimates in each column
dim(coef(ridge.mod)) # predictors x number of lambda values
ridge.mod$lambda[50] # extracts the 50th lambda value
coef(ridge.mod)[,50] # shows coefficients of the 50th lambda value

# The squared root of l2 norm value
sqrt(sum(coef(ridge.mod)[-1,50]^2)) # = 6.36, as 50 get smaller l2 gets smaller
ridge.mod$lambda[1] # largest lambda value so heaviest shrinkage of coefficients
sqrt(sum(coef(ridge.mod)[-1,1]^2)) # approx = 0

# Calculate ridge regression parameter estimates for a new lambda
predict(ridge.mod,s=50,type="coefficients")[1:20,] # lambda = 50, NOT 50th lambda from above
sqrt(sum(predict(ridge.mod,s=50,type="coefficients")[1:20,]^2)) # l2 norm value

# train/test set and MSE
set.seed(1)
train=sample(1:nrow(x), nrow(x)/2)
test=(-train)
y.test=y[test]
# Fit ridge regression using training set
ridge.mod=glmnet(x[train,],y[train],alpha=0,lambda=grid, thresh=1e-12)
# Predict by ridge regression when lambda=4
ridge.pred=predict(ridge.mod,s=4,newx=x[test,])
mean((ridge.pred-y.test)^2) # MSE, can try diff lambda by changing s above

# Comparison between least squares and ridge regression
lm(y~x, subset=train)
predict(ridge.mod,s=0,exact=T,type="coefficients",x=x[train,],y=y[train])[1:20,]
# gives exact same results since lambda=0

# choosing lambda by cross-validation
set.seed(1)
cv.out=cv.glmnet(x[train,],y[train],alpha=0) # uses 10-fold CV
plot(cv.out) # shows MSE against log(lambda)
bestlam=cv.out$lambda.min
ridge.pred=predict(ridge.mod,s=bestlam,newx=x[test,])
mean((ridge.pred-y.test)^2) # MSE
# fit the best case using the full data
out=glmnet(x,y,alpha=0)
predict(out,type="coefficients",s=bestlam)[1:20,]
```

4.3.3 Lasso regression

The lasso coefficient estimates $\hat{\beta}_i^L,\ i = 0, ..., p$ are the values that minimise:

$$\sum_{i=1}^{n}(y_i - \hat{\beta}_0^L - \sum_{j=1}^{p}\hat{\beta}_j^L x_{ij})^2 + \lambda\sum_{j=1}^{p}|\hat{\beta}_j^L| = RSS + \lambda\sum_{j=1}^{p}|\hat{\beta}_j^L|$$

In the above objective function, the second term is the lasso penalty term, which is the l_1 norm of the coefficient vector $\hat{\beta}^L$.

The lasso also shrinks the coefficient estimates towards zero, but the l_1 penalty has the effect of forcing sum of the coefficient estimates to be exactly 0 when λ is large enough. Therefore, the lasso performs variable selection, and the models generated from the lasso are generally easier to interpret.

We say that the lasso yields sparse models – that is, models that involve only a subset of the variables. As in ridge regression, selecting a good value of λ for the lasso is critical. The cross-validation approach is suitable for the job.

4.3.4 Ridge and Lasso in R

```
lasso.mod=glmnet(x[train,],y[train],alpha=1,lambda=grid) # alpha=1 for lasso
plot(lasso.mod)

# choosing lambda by cross-validation
set.seed(1)
cv.out=cv.glmnet(x[train,],y[train],alpha=1)
plot(cv.out)
bestlam=cv.out$lambda.min
lasso.pred=predict(lasso.mod,s=bestlam,newx=x[test,])
mean((lasso.pred-y.test)^2)

# fit the best case using the full data
out=glmnet(x,y,alpha=1,lambda=grid)
lasso.coef=predict(out,type="coefficients",s=bestlam)[1:20,]
lasso.coef # showing all and even zero coefficients
# showing the non-zero coefficients
lasso.coef[lasso.coef!=0]
```

We can compare models' test performance via CV or a separate test set. Generally, ridge is better if many predictors all have small contributions (many "weak" signals), whereas lasso is good if only a few predictors are truly important and others are noise.

4.4 Dimension Reduction Methods (PCA, PCR, PLS)

4.4.1 Dimension reduction methods - overview theory

Transform the predictors and then fit a least squares model using the transformed variables.

Let $Z_1, ..., Z_M,\ M < p$, denote M linear combinations of the original p predictors, and:

$$Z_m = \sum_{j=1}^{p}\phi_{jm}X_j \tag{1}$$

For some constants $\phi_{1m}, ..., \phi_{pm},\ m = 1, ..., M$. We can then fit the linear regression model using least squares:

$$y_i = \theta_0 + \sum_{m=1}^{M} \theta_m z_{im} + \varepsilon_i \ i = 1, ..., n \tag{2}$$

If the constants $\phi_{1m}, ..., \phi_{pm}$ in (1), $m = 1, ..., M$ are chosen wisely, then the above dimension reduction approaches can often outperform normal least squares regression.

Notice that from (1), we have:

$$\begin{aligned}\sum_{m=1}^{M} \theta_m z_{im} &= \sum_{m=1}^{M} \theta_m \sum_{j=1}^{p} \phi_{jm} X_{ij} \\ &= \sum_{j=1}^{p} \sum_{m=1}^{M} \theta_m \phi_{jm} X_{ij} = \sum_{j=1}^{p} \beta_j X_{ij}\end{aligned}$$

So one can see that (2) is a special case of the original linear regression model with coefficients β satisfy the above relationship.

When p is large relative to n, selecting a value of $M << p$ can significantly reduce the variance of the fitted coefficients. If $M = p$ and all Z_m are linearly independent, then no dimension reduction occurs.

Dimension reduction methods have two steps:
1. Obtain transformed predictors $Z_1, ..., Z_M$.
2. Fit the model (2).

We introduce two methods for generating transformed predictors $Z_1, ..., Z_m$: *PCA* and *PLS*.

4.4.2 Principle components analysis (PCA)

Generates normalised linear combinations of the correlated variables. These linear combinations provide a new set of axes in the direction of the maximum variability. Can be used for dimension reduction, and transformation of a set of dependent variables into a new set of independent variables.

Let $\boldsymbol{X} = (X_1, ..., X_p)$ be a random vector (not necessarily normal) with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. The first principal component (PC), Z_1 is defined as the normalised linear combination of $X_1, ..., X_p$ with maximum variance, i.e.

$$Z_1 = \phi_{11} X_1 + \phi_{21} X_2 + ... + \phi_{p1} X_p$$

With $\phi_{11}^2 + ... + \phi_{p1}^2 = 1$ and $\mathrm{Var}(Z_1)$ is higher than the variance of any other linear combination of $X_1, ..., X_p$.

Similarly, the second PC, Z_2, is a normalised linear combination with second highest variance and independent of Z_1. In general, the m-th PC Z_m is given by, for $m = 1, ..., p$:

$$Z_m = \phi_{1m} X_1 + \phi_{2m} X_2 + ... + \phi_{pm} X_p = \boldsymbol{X} \boldsymbol{\phi_m}^\top$$

where $\boldsymbol{\phi_m} = (\phi_{1m}, ..., \phi_{pm})$, $\boldsymbol{\phi_m} \boldsymbol{\phi_m}^\top = 1$, $\mathrm{Var}(Z_m) = \boldsymbol{\phi_m} \boldsymbol{\Sigma} \boldsymbol{\phi_m}^\top$, $\mathrm{Cov}(Z_i, Z_j) = 0\ \forall i \neq j$, and:

$$\mathrm{Var}(Z_1) \geq \mathrm{Var}(Z_2) \geq ... \geq \mathrm{Var}(Z_p)$$

Theorem. Let $(\xi_1, \boldsymbol{e}_1), (\xi_2, \boldsymbol{e}_2), ..., (\xi_p, \boldsymbol{e}_p$ be teh eigenvalue-eigenvector pairs of Σ such that $\xi_1 \geq ... \geq \xi_p$, then the m-th PC is given by for $m = 1, ..., p$:

$$Z_m = \boldsymbol{X}\boldsymbol{e}_{\boldsymbol{m}}^\top$$

and $\text{Var}(Z_m) = \boldsymbol{e_m}\boldsymbol{\Sigma}_{\boldsymbol{m}}^\top = \xi_m,\ \text{Cov}(Z_i, Z_j) = \boldsymbol{e_i}\boldsymbol{\Sigma}\boldsymbol{e_j}^\top = 0,\ i \neq j.$

Remarks:

- Two or more ξ_m's values may be equal in some cases, however, these eigenvalues would have multiple eigenvectors hence p different PCs for any given $\boldsymbol{\Sigma}$.
- The total variance is $T = \sum_{j=1}^p \text{Var}(X_j) = \text{Trace}(\boldsymbol{\Sigma}) = \sum_{m=1}^p \xi_m$. Then the proportion of variance due to the m-the PC is ξ_m/T. In most cases, the first few PCs contain large percentages of the total variance and the remaining PCs are insignificant.
- If $\boldsymbol{X}$ contain independent variables, then the PCs are the original set of variables.

Sample PCA:
When $\boldsymbol{\Sigma}$ is unknown, we cannot obtain the principal components (PCs) directly. Instead we can estimate the PCs by the sample PCs. The sample PCs are defined as a set of mutually independent, normalised linear combinations of original variables with maximum sample variance.

Assume the data $\boldsymbol{X_1}, ..., \boldsymbol{X_n}$ represent n independent observations from some p-dimensional population with unknown mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$. We estimate:

- $\boldsymbol{\mu}$ by sample mean vector $\bar{\boldsymbol{X}}$
- $\boldsymbol{\Sigma}$ by sample covariance matrix: $\boldsymbol{S}_n = \frac{1}{n-1}\sum_{j=1}^n (\boldsymbol{x}_j - \bar{\boldsymbol{x}})^\top (\boldsymbol{x}_j - \bar{\boldsymbol{x}})$

The sample PCs can be obtained using normalised eigenvectors of $\boldsymbol{S}_n$.

PCI using correlation matrix:
When some of the individual variables in $\boldsymbol{X}$ have significantly larger variances than the other variables, they will dominate the first few principle components. To overcome this limiation, we introduce principal components using correlation matrix instead of covariance matrix.

Let $\boldsymbol{\rho}$ be correlation matrix of the random vector $\boldsymbol{X}$ and Z_i be the standardised X_i, that is $Z_i = \frac{X_i - \mu_i}{\sigma_i},\ i = 1, ..., p.$

Then $\boldsymbol{\rho} = \text{Cov}(\boldsymbol{Z})$, where $\boldsymbol{Z} = (Z_1, ..., Z_p)$. Let $\delta_1, ..., \delta_p$ be the eigenvalues and $\boldsymbol{\omega}_1, ..., \boldsymbol{\omega}_p$ be the corresponding eigenvectors of $\boldsymbol{\rho}$. Then the Pcs of $\boldsymbol{Z}$ are given by:

$$U_j = \boldsymbol{Z}\boldsymbol{\omega}_{\boldsymbol{j}}^\top,\ j = 1, ..., p$$

4.4.3 Principle Component Regression (PCR)

PCR approach involves constructing the first M PCs and then using them as the predictors in a linear regression model that is fit using least squares.

- If the assumption underlying PCR holds, then fitting a least squares model to $Z_1, ..., Z_M$ will lead to better results than fitting a least squares model to $X_1, ..., X_p$ due to the reduced model over fitting.
- Even though PCR provides a simple way to perform regression using $M < p$ predictors, it is NOT a feature selection method, since all original variables are used in constructing the PCs. So PCR is more related to ridge regression than to a lasso.
- In PCR, the number of PCs, M, cn be chosen by cross-validation
- When performing PCR, we usually recommend standardising each predictor using: $\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^n (x_{ij} - \bar{x}_j)^2}}$.

4.4.4 PCR in R

```
library(pls)
set.seed(2)
pcr.fit=pcr(Salary~., data=Hitters,scale=TRUE,validation="CV")
summary(pcr.fit)
validationplot(pcr.fit,val.type="MSEP")

# using training and test sets
# Set up train and test split
set.seed(1)
train=sample(1:nrow(x), nrow(x)/2)
test=(-train)
y.test=y[test]

set.seed(1)
pcr.fit=pcr(Salary~., data=Hitters, subset=train, scale=TRUE, validation="CV")
# scale=TRUE scales predictors, validation="CV" does 10-fold CV
validationplot(pcr.fit,val.type="MSEP") # visual of the MSEP help choose optimal components
summary(pcr.fit)
pcr.pred=predict(pcr.fit,x[test,],ncomp=5)
mean((pcr.pred-y.test)^2) # MSE for 5 components
pcr.pred=predict(pcr.fit,x[test,],ncomp=7)
mean((pcr.pred-y.test)^2) # MSE for 7 components

# using whole data set
set.seed(1)
pcr.fit=pcr(Salary~., data=Hitters, scale=TRUE, validation="CV")
validationplot(pcr.fit,val.type="MSEP")
pcr.fit=pcr(y~x,scale=TRUE,ncomp=5) # use 5 PCs since lowest test MSE from above
summary(pcr.fit)
```

4.4.5 Partial Least Squares (PLS)

PCR indetifies the PCs in an *unsupervised* way, since the response is not involved in the process. Now we introduce *supervised* alternative to PCR, PLS.

PLS is also a dimension reduction method, which:

- First identifies a new set of features $Z_1, ..., Z_M$ that are linear combinations of the original variables, and
- then fits a linear model by least squares using the M new variables.

The PLS approach attempts to find directions that help explain both the response and the predictors.

Overview of PLS Computation:

1. Firstly, after standardising the p predictors, PLS computes the first direction Z_1 by setting each ϕ_{ji} in (1) equal to the coefficient from the simple linear regression of Y to X_j, which is proportional to the correlation between Y and X_j.
2. To identify the second PLS direction, we first adjust each of the preictors for Z_1, by regressing each predictor on Z_1 and taking residuals, which form the *orthogonalized* data. This data contain the remaining information that have not been explained by Z_1. Repeat step 1 using this data to generate Z_2.
3. Repeat step 2 to generate the remaining PLS directions, $Z_3, ..., Z_M$.

Similar to PCR, the number M in the PLS is a tuning parameter that is typically chosen by

cross-validation. In practice it often performs no better than ridge regression or PCR.

Pros:

- PCR/PLS handle multicollinearity well by constructing orthogonal components.
- Can reduce noise by dropping low-variance directions (PCR) or uncorrelated directions (PLS).
- Especially useful when p is almost as large or larger than n.

Cons:

- Components may be hard to interpret (each is a mix of original variables).
- If the relationship between Y and X is not aligned with top variance components, PCR may miss it.
- Need to choose number of components M (via CV).

4.4.6 PLS in R

```
# Set up train and test split
set.seed(1)
train=sample(1:nrow(x), nrow(x)/2)
test=(-train)
y.test=y[test]

library(pls)
set.seed(1)
pls.fit=plsr(Salary~., data=Hitters,subset=train, scale=TRUE, validation="CV")
# scale=TRUE scales predictors, validation="CV" does 10-fold CV
summary(pls.fit)
validationplot(pls.fit,val.type="MSEP")
pls.pred=predict(pls.fit,x[test,],ncomp=2)
mean((pls.pred-y.test)^2) # MSE for 1 component
pls.pred=predict(pls.fit,x[test,],ncomp=1)
mean((pls.pred-y.test)^2) # MSE for 2 components

# using whole data set use 2 components since gives lowest test MSE
pls.fit=plsr(Salary~., data=Hitters,scale=TRUE,ncomp=2)
summary(pls.fit)
```

5 Non-linear Models

Types:

- Polynomial regression: extra predictors which raise original predictors to a power
- Step functions: cut the range of a variable into K regions in order to produce a qualitative varaiable
- Regression splines: divide the range of X into K distinct regions. Within each region, a polynomial function is fit to the data.
- Smoothing splines: result from minimising a RSS subject to a smoothness penalty
- Local regression: regions of a spline that are allowed to overlap (in a smooth way)
- Generalised additive models (GAMs): extend the models above to deal with multiple predictors

5.1 Polynomial Regression

Defined by:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + ... + \beta_d x_i^d + \varepsilon_i$$

Coefficients estimated by using least squares approach because it can be treated as a linear model with predictors $x, x^2, ..., x^d$.

Calculating CI of $\hat{y}$: For a given value x_0, $\hat{y}_0 = \hat{\beta}_0 + \sum_{j=1}^{d} \hat{\beta}_j x_0^j$. Let $\hat{\boldsymbol{C}}$ denote the $(d+1) \times (d+1)$ covariance matrix of the $\hat{\beta}_j$ and let $\ell_0 = (1, x_0, x_0^2, ..., x_0^d)$. Then $\text{Var}(\hat{y}_0) = \ell_0 \hat{\boldsymbol{C}} \ell_0^\top$. Then the 95% CI for $\hat{y}_0$ is: $\hat{y}_0 \pm 2\sqrt{\text{Var}(\hat{y}_0)}$

5.1.1 Polynomial regression in R

```
# Dataset
library(ISLR)
attach(Wage)

# 1. Orthogonal polynomial: each predictor is a linear of combination
# of age, age^2, age^3, and age^4
fit=lm(wage~poly(age,4),data=Wage)
coef(summary(fit))
# 2. normal polynomial: predictors are age, age^2, age^3, age^4.
fit2=lm(wage~poly(age,4,raw=T),data=Wage)
coef(summary(fit2))
# 3. Using I()
fit2a=lm(wage~age+I(age^2)+I(age^3)+I(age^4),data=Wage)
coef(fit2a)
# 4. Using cbind()
fit2b=lm(wage~cbind(age,age^2,age^3,age^4),data=Wage)
coef(fit2b)

# Creating a grid of age
agelims=range(age)
age.grid=seq(from=agelims[1],to=agelims[2])

# Predictions using model 1
preds=predict(fit,newdata=list(age=age.grid),se=TRUE)
# Predictions using model 2
preds2=predict(fit2,newdata=list(age=age.grid),se=TRUE)
max(abs(preds$fit-preds2$fit)) #maximal difference between the two predictions
# approx 0 showing that they are basically the same eventhough using diff. methods
```

```r
# CI of predictied values
se.bands=cbind(preds$fit+2*preds$se.fit,preds$fit-2*preds$se.fit)

# Draw plots
par(mfrow=c(1,2),mar=c(4.5,4.5,1,1),oma=c(0,0,4,0)) #mar and oma both set margins
plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
title("Degree-4 Polynomial",outer=T) # overall title
lines(age.grid,preds$fit,lwd=2,col="blue") # lwd: line width
matlines(age.grid,se.bands,lwd=1,col="blue",lty=3) # lty: line type

# Analysis of variance (F-test)
# H0: Model fit.i is sufficient to explain the data <-> H1: Model fit.i+1 is required.
# Model fit.i is nested in fit.i+1
fit.1=lm(wage~age,data=Wage)
fit.2=lm(wage~poly(age,2),data=Wage)
fit.3=lm(wage~poly(age,3),data=Wage)
fit.4=lm(wage~poly(age,4),data=Wage)
fit.5=lm(wage~poly(age,5),data=Wage)
anova(fit.1,fit.2,fit.3,fit.4,fit.5)
# Analysis of Variance Table
# Model 1: wage ~ age
# Model 2: wage ~ poly(age, 2)
# Model 3: wage ~ poly(age, 3)
# Model 4: wage ~ poly(age, 4)
# Model 5: wage ~ poly(age, 5)
#   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
# 1   2998 5022216
# 2   2997 4793430  1    228786 143.5931 < 2.2e-16 *** # adds significance => non-linear
# 3   2996 4777674  1     15756   9.8888  0.001679 **  # still adds statistical significance
# 4   2995 4771604  1      6070   3.8098  0.051046 .
# 5   2994 4770322  1      1283   0.8050  0.369682
# Since cubic still adds statistical signifiance probably best to use it

# ANOVA can also be used to test more complex models
fit.1=lm(wage~education+age,data=Wage)
fit.2=lm(wage~education+poly(age,2),data=Wage)
fit.3=lm(wage~education+poly(age,3),data=Wage)
anova(fit.1,fit.2,fit.3)

# Choose best degree of polynomial by CV
# cv.glm() does LOOCV error, without specifying K value
# fit glm -> find LOOCV error with cv.glm() -> output error rate
library(boot)
glmfit.1=glm(wage~age,data=Wage)
cv.err1=cv.glm(Wage,glmfit.1)
cv.err1$delta[1]
glmfit.2=glm(wage~poly(age,2),data=Wage)
cv.err2=cv.glm(Wage,glmfit.2)
cv.err2$delta[1]
glmfit.3=glm(wage~poly(age,3),data=Wage)
cv.err3=cv.glm(Wage,glmfit.3)
cv.err3$delta[1]
glmfit.4=glm(wage~poly(age,4),data=Wage)
cv.err4=cv.glm(Wage,glmfit.4)
cv.err4$delta[1]
glmfit.5=glm(wage~poly(age,5),data=Wage)
cv.err5=cv.glm(Wage,glmfit.5)
cv.err5$delta[1]
```

Polynomial logistic regression in R

```
fit=glm(I(wage>250)~poly(age,4),data=Wage,family=binomial)
preds=predict(fit,newdata=list(age=age.grid),se=T)
pfit=exp(preds$fit)/(1+exp(preds$fit))
se.bands.logit = cbind(preds$fit+2*preds$se.fit, preds$fit-2*preds$se.fit)
se.bands = exp(se.bands.logit)/(1+exp(se.bands.logit))
preds=predict(fit,newdata=list(age=age.grid),type="response",se=T)

# Plot example
plot(age,I(wage>250),xlim=agelims,type="n",ylim=c(0,.2))
points(jitter(age), I((wage>250)/5),cex=.5,pch="|",col="darkgrey")
lines(age.grid,pfit,lwd=2, col="blue")
matlines(age.grid,se.bands,lwd=1,col="blue",lty=3)
```

5.2 Step Functions

Break X into ranges using step functions

- Create cutpoints $c_1, ..., c_K$ in range of X, and construct $K+1$ new variables. At most one of C_i can be 0, also $\sum_{i=0}^{K} C_i(x) = 1$:

$$\begin{aligned} C_0(x) &= I(X < c_1) \\ &\vdots \\ C_{K-1}(x) &= I(c_{K-1} \leq X < c_k) \\ C_K(x) &= I(X \geq c_K) \end{aligned}$$

- Use least squares to fit a linear model using $C_1(X), ...C_K(X)$ as predictors. *Note* that $C_0(X)$ is not used as it is redundant with the intercept being in the model.
- When $X < c_1$, all predictors are 0 so β_0 is the mean value of Y for $X < c_1$
- β_j represents the average increase in the response for X in $[c_j, c_{j+1})$ relative to $X < c_1$
- In general, the step function may not describe the underlying trends properly, unless there are natural breakpoints in the predcitors

5.2.1 Step function in R

```
table(cut(age,4))
# (17.9,33.5]   (33.5,49]   (49,64.5] (64.5,80.1]
#         750        1399         779          72
fit=lm(wage~cut(age,4),data=Wage)
coef(summary(fit))
```

5.3 Basis Functions

Let a family of functions be applied to a predictor X: $b_1(X), ..., b_K(x)$. We now fit $y_i = \beta_0+\beta_1 b_1(x_i)+...+\beta_k b_k(x_i)+\varepsilon_i$. Use least squares to estimate the unknown regression coefficients. If $b_j(x) = x^j \implies$ polynomial regression, or if $b_j(x) = I(c_j \leq x < c_{j+1}) \implies$ piecewise-constant regression.

5.3.1 Spline basis representation example (cubic spline)

A cubic spline with K knots. One way of representing a cubic spline is to start off with $\beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3$ and then add one *truncated power basis* function per knot which is defined

as:

$$h(x,\xi) = (x-\xi)_+^3 = \begin{cases} (x-\xi)^2 & \text{if } x > \xi \\ 0 & \text{otherwise} \end{cases}$$

Where ξ is a knot. So in order to fit a cubic spline to the data set with K knots, we perform least squares to the following model:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \beta_4 h(x_i, \xi_1) + ... + \beta_{K+3} h(x_i, \xi_K) + \varepsilon_i$$

Where $\xi_1, ..., \xi_K$ are the K knots. We need to estimate a total of $K+4$ regression coefficients. It means fitting a cubic spline with K knots uses $K+4$ degrees of freedom.

5.3.2 Basis functions in R

```r
library(splines) # used for bs (b-spline basis)
fit=lm(wage~bs(age,knots=c(25,40,60)),data=Wage) # default degree=3 in bs can change
pred=predict(fit,newdata=list(age=age.grid),se=T)

# Plotting spline with 95% confidence interval
plot(age,wage,col="gray")
lines(age.grid,pred$fit,lwd=2)
lines(age.grid,pred$fit+2*pred$se,lty="dashed") # Upper bound
lines(age.grid,pred$fit-2*pred$se,lty="dashed") # Lower bound

dim(bs(age,knots=c(25,40,60))) # bs() generates the B-spline basis matrix
dim(bs(age,df=6)) # df=6 gives three knots
attr(bs(age,df=6),"knots") # get knots attributes of the matrix bs
```

5.4 Regression Splines

5.4.1 Piecewise polynomials

A piecewise cubic polynomial fits $y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \varepsilon_i$. The points where coefficients change are called knots. E.g. A single knot at point c:

$$y_i = \begin{cases} \beta_{01} + \beta_{11} x_i + \beta_{21} x_i^2 + \beta_{31} x_i^3 + \varepsilon_i & \text{if } x_i < c \\ \beta_{02} + \beta_{12} x_i + \beta_{22} x_i^2 + \beta_{32} x_i^3 + \varepsilon_i & \text{if } x_i \geq c \end{cases}$$

Using more knots in the models leads to a more flexible piecewise polynomial.

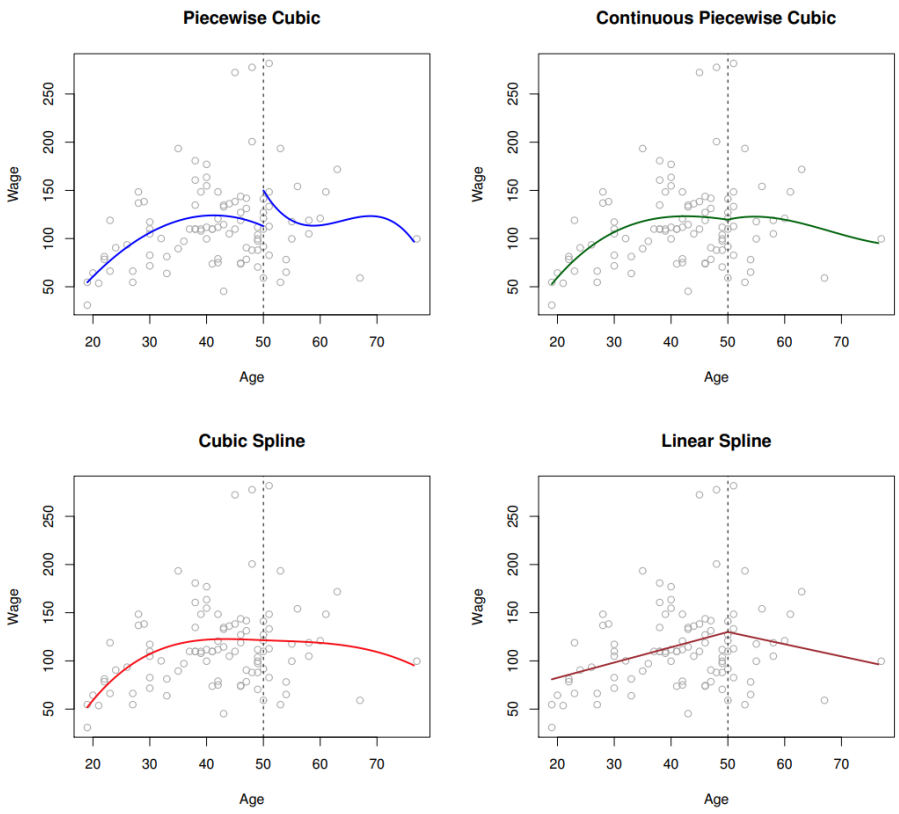


FIGURE 7.3. *Various piecewise polynomials are fit to a subset of the* `Wage` *data, with a knot at* `age=50`. Top Left: *The cubic polynomials are unconstrained.* Top Right: *The cubic polynomials are constrained to be continuous at* `age=50`. Bottom Left: *The cubic polynomials are constrained to be continuous, and to have continuous first and second derivatives.* Bottom Right: *A linear spline is shown, which is constrained to be continuous.*

5.4.2 Constraints and splines

- The fitted curve must be continuous
- The fitted curve should be very smooth
  - For cubic spline: the first and second derivatives of the piecewise cubic function need to be continuous at each know
  - For d-degree spline: the continuity in derivatives is required for up to degree $d-1$ at each knot
- Each constraint we impose on the piecewise polynomials reduce on degree of freedom
- A *natural spline* is a regression spline with linear boundary constraints: the spline function is required to be linear in the region where X is the smaller than the smallest know, or larger than the largest knot

5.4.3 Choosing the number and location of knots

1. Place more knots in places where we feel the function might vary most rapidly (place fewer knots where it seems more stable)
2. Place knots in a uniform fashion. Firstly, specify the desired degrees of freedom, and then have software place the corresponding number of knots at uniform quantiles of the data.

CV to choose optimal number of knots

1. Remove a portion of the data (e.g. 10%, then fit a spline with a certain number of knots to the remaining data
2. Use fitted spline to make predictions for the held-out data
3. Repeat steps 1-2 until each observation has been held-out once, and then compute the overall cross-validated RSS
4. Repeat steps 1-3 for different number of knots K
5. Value of K that gives the smallest RSS is the optimal one

Regression splines vs polynomial regression

- To increase flexibility, polynomial regression must use a high degree, while splines can simply increase number of knots but keeping the degree fixed. The latter generally produces more stable estimates
- Splines also allow us to adjust the position of the knots to create more flexibility. The polynomial regression doesn't have this feature

5.4.4 Splines in R

```
library(splines) # used for ns (natural cubic spline)
fit2=lm(wage~ns(age,df=4),data=Wage)
pred2=predict(fit2,newdata=list(age=age.grid),se=T)

# Another way of draw plots with CIs
plot(age,wage,col="gray")
lines(age.grid,pred2$fit,lwd=2)
lines(age.grid,pred2$fit+2*pred2$se,lty="dashed")# Upper bound
lines(age.grid,pred2$fit-2*pred2$se,lty="dashed")# Lower bound

dim(ns(age,df=4)) # df=4 gives three knots for ns()
attr(ns(age,df=4),"knots") # get knots attributes of the matrix ns
```

5.5 Smoothing Splines

Find some function $g(x)$ that can fit the observed data well. We need to consider both adherence and smoothness.

Adherence measured by the $RSS = \sum_{i=1}^{n} (y_i - g(x_i))^2$, and the smoothness can be measured by $\int g''(t)^2 \, dt$. Since there is a trade-off between adherence and smoothness, we construct the following objective function (with $\lambda \geq 0$): $\sum_{i=1}^{n}(y_i - g(x_i))^2 + \lambda \int g''(t)^2 \, dt$ (like *loss* + *penalty*).

Some remarks:

- $\lambda = 0$: smoothness term has no effect. g will be very jumpy and will exactly interpolate the training observations
- $\lambda \to \infty$, g will be perfectly smooth
- So λ controls the bias-variance trade-off of the smoothing spline
- Smoothing spline can be shown to be a nutral cubic spline with knots at the unique values of $x_1, ..., x_n$

The effective degrees of freedom:

- λ controls the smoothness of the spline, and hence the effective degrees of freedom
- When λ increases from 0 to ∞, the effective degrees of freedom, denoted by df_λ, decrease from n to 2
- df_λ is a measure of the flexibility of the smoothing spline

Choosing the smoothing parameter λ: Find a value of λ that makes the CV RSS as small as possible. LOOCV can also be computed for smoothing splines, using the following formula $RSS_{cv}(\lambda) = \sum_{i=1}^{n} \left[\frac{y_i - \hat{g}_\lambda(x_i)}{1-\{\boldsymbol{S}_\lambda\}_{ii}}\right]^2$, where $\hat{\boldsymbol{g}}_\lambda = \boldsymbol{S}_\lambda \boldsymbol{y}$

5.5.1 Smoothing splines in R

```
fit=smooth.spline(age,wage,df=16) # lambda determined by df=16
fit2=smooth.spline(age,wage,cv=TRUE) # lambda determined by CV
fit2$df # df under CV

# Plotting example
plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
title("Smoothing Spline")
lines(fit,col="red",lwd=2)
lines(fit2,col="blue",lwd=2)
legend("topright",legend=c("16 DF","6.8 DF"),col=c("red","blue"),lty=1,lwd=2,cex=.8)
```

5.6 Local Regression

Local regression at $X = x_0$:

1. Gather the fraction $s = \frac{k}{n}$ of training points whose x_i are closest to x_0
2. Assign a weight $K_{i0} = \omega \cdot \frac{x_i - x_0}{h(x_0)}$ to each point in this neighbourhood, where $h(x_0)$ is the bandwidth, the distance from x_0 to its kth closest x_i
3. Fit a weighted least squares regression of the y_i on the x_i using the aforementioned weights, by finding $\hat{\beta}_0$ and $\hat{\beta}_1$ that minimise $\sum_{i=1}^{n} K_{i0}(y_i - \beta_0 - \beta_1 x_i)^2$
4. The fitted value at x_0 is given by $\hat{f}(x_0) = \hat{\beta}_0 + \hat{\beta}_1 x_0$

s controls the flexibility of the non-linear fit. The smaller the s, the more local and wiggly will be our fit; alternatively, a very large s will lead to a global fit to the data using all of the training observations. CV can be used to choose s.

Why Local Regression?

- Adapts well to bias problems at boundaries and in regions of high curvature
- Easy to understand and interpret
- Methods have been developed that provide fast computation for one or more independent variables
- Very simple, can be used to work for different distributional assumptions
- A local model enables derivation of response adaptive methods for span value and polynomial order selection in a straightforward manner

How to choose polynomial degree?: The choice is a bias-variance trade off. A higher degree will produce a less biased, but more variable estimate than a lower degree one.

How to choose the weight function? Want to consider a weight function $\omega(u)$ that are peaked at $u = 0$ and that decay smoothly to 0 as u increases. A smooth weight function results in a smoother estimate. We also want a weight function that is nonzero only on a bounded interval, helping with computation speed.

5.6.1 Local regression in R

```r
fit=loess(wage~age,span=.2,data=Wage) # 20% of the neighborhood
fit2=loess(wage~age,span=.5,data=Wage) # 50% of the neighborhood
# Large span (s) => smoother curve. Span determines the proportion of the data
# used in each localised regression.

# Plotting example
plot(age,wage,xlim=agelims,cex=.5,col="darkgrey")
title("Local Regression")
lines(age.grid,predict(fit,data.frame(age=age.grid)),col="red",lwd=2)
lines(age.grid,predict(fit2,data.frame(age=age.grid)),col="blue",lwd=2)
legend("topright",legend=c("Span=0.2","Span=0.5"),col=c("red","blue"),lty=1,lwd=2,cex=.8)
```

5.7 Generalised Linear Models (GLMs)

- *Probability distribution*: Instead of assuming normal distribution, GLMs work with distributions in the exponential family (e.g. normal, Poisson, binomial, and gamma)
- *Model for the mean*: The mean is a linear function of the predictors, $\boldsymbol{X}$, $E[Y] = f(\boldsymbol{X})$. In GLMs some smooth monotonic transformation of the mean is a linear function of $\boldsymbol{X}$. $g(E[Y]) = f(\boldsymbol{X})$.

Why GLMs:

- Easy to estimate standard errors, constructing CIs, testing, model selection and other statistical features
- GLMs are vary common in many areas of statistics, so they are constantly improving and we can draw upong developments from many industries
- There is standard software for fitting GLMs
- Although there is standard software companies create their own specialised GLMs so there is space to innovate

The exponential family and the link function:

GLMs involve distributions that can be expressed in exponential form: $f(y|\theta,\phi) = e^{\frac{y\theta - b(\theta)}{\phi} + c(y,\theta)}$, θ is the canonical parameter, and ϕ is the dispersion parameter (that plays a role in defining the variance of y).

The canonical parameter θ is a function of μ, denoted by $\theta(\mu)$, the canconical link function:

- Normal: $\theta(\mu) = \mu$
- Poisson: $\theta(\mu) = \ln(\mu)$
- Exponential, gamma: $\theta(\mu) = -\mu^{-1}$
- Categorical, binomial, multinomial: $\theta(\mu) = \ln\left(\frac{\mu}{1-\mu}\right)$

5.8 Generalised Additive Models (GAMs)

GAMs extend multiple linear regression by allowing each variable to have its own non-linear effect, then adding them up:

$$y_i = \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + ... + f_p(x_{ip}) + \varepsilon_i$$

Could use and combine any non-linear methods such as local regression, polynomial regression, splines, or any combination of the approaches discussed. Also could use qualitative variables using dummy variables.

5.8.1 Pros and Cons

Pros

- GAM allow non-linear f_j to each X_j and does it automatically, so we don't need to manually try out many different transformations.
- Non-linear fit could make more accurate predictions.
- Since model is additive, we can still examine the effect of each X_j on Y individually while holding all the other variables fixed. Hence GAMs provid useful inference.
- Smoothness of function f_j for X_j can be summarised via degrees of freedom.

Cons

- Restricted to be additive. Imporatnt interactions may be missed, however we can add interaction terms to the GAM model by including additional predictors of the form $X_j \times X_k$. In addition we can add low-dimensional interaction functions of the form $f_{jk}(X_j, X_k)$; such terms can be fit using two-dimensional smoothers such as local regression.

5.8.2 GAMs in R

Using gam library:

```
library(gam)
gam1=lm(wage~ns(year,4)+ns(age,5)+education,data=Wage) # ns() natural cubic spline
gam.m3=gam(wage~s(year,4)+s(age,5)+education,data=Wage) # s() smoothing splines
par(mfrow=c(1,3))
plot(gam.m3, se=TRUE, col="blue") # same as plot.Gam()
plot.Gam(gam1, se=TRUE, col="red") # upper case plot.Gam for gam library

gam.m1=gam(wage~s(age,5)+education,data=Wage)
gam.m2=gam(wage~year+s(age,5)+education,data=Wage)
anova(gam.m1,gam.m2,gam.m3,test="F")
summary(gam.m3) #only age shows significant non-linearity

# Predictions
preds=predict(gam.m2,newdata=Wage)
mean((preds-Wage$wage)^2) # training MSE
```

Logistic regression GAM using gam

```
gam.lo=gam(wage~s(year,df=4)+lo(age,span=0.7)+education,data=Wage)
plot.Gam(gam.lo, se=TRUE, col="green")
gam.lo.i=gam(wage~lo(year,age,span=0.5)+education,data=Wage)
```

Using mgcv library:

```
library(mgcv)
gam.new=mgcv::gam(wage~year+s(age)+education,data=Wage,method="REML")
summary(gam.new)
plot.gam(gam.new, se=TRUE, col="purple") # lower case plot.gam for mgcv library
```

Logistic regression GAM using mgcv

```
gam.lr=gam(I(wage>250)~year+s(age,df=5)+education,family=binomial,data=Wage)
par(mfrow=c(1,3))
plot(gam.lr,se=T,col="green")
table(education,I(wage>250))
gam.lr.s=gam(I(wage>250)~year+s(age,df=5)+education,
    family=binomial,data=Wage,subset=(education!="1. < HS Grad"))
plot(gam.lr.s,se=T,col="green")
```

6 Logistic Regression and Bayes Theorem

However, the key new topic in Week 6 was **Logistic Regression**, a classification method.

6.1 Classification Methods (Introduction)

Types:

1. Logistic regression
2. Linear discriminant analysis (LDA)
3. Quadratic discriminant analysis (QDA)
4. Naive Bayes
5. K-nearest neighbours
6. Classification trees
7. Support vector machines

Used to capture a qualitative variable instead of a quantitative one. Why can't we encode a variable using linear regression past a binary response? (a) A regression method cannot accommodate a qualitative response with more than two classes (as order of the classes matters and how different they are to each other matters in regression), (b) A regression method will not provide meaningful estimates of $Pr(Y|X)$, even with just two classes.

6.2 Logistic Regression Theory

For a binary outcome Y (two classes, often coded as 0/1 for convenience), logistic regression models the **probability** of $Y = 1$ as a function of predictors X. We write:

$$p(X) = P(Y = 1 \mid X) = \frac{\exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p)}{1 + \exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p)}.$$

This is the **logistic function** mapping $\mathbb{R}$ to $(0, 1)$. It ensures predicted probabilities stay between 0 and 1, unlike a linear model.

The logistic model: $p(X) = \frac{e^{\beta_0+\beta_1 X}}{1+e^{\beta_0+\beta_1 X}}$ and the multiple logistic regression: $p(\boldsymbol{X}) = \frac{e^{\beta_0+\beta_1 X_1+\ldots+\beta_p X_p}}{1+e^{\beta_0+\beta_1 X_1+\ldots+\beta_p X_p}}$

Taking odds:

$$\frac{p(X)}{1 - p(X)} = \exp(\beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p) \in (0, \infty),$$

If odds close to 0 $\implies$ very low probability of X and odds close to ∞ $\implies$ very high probability of X.
So the **log-odds** (logit) is linear:

$$\log \frac{p(X)}{1 - p(X)} = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p.$$

Interpretation: β_j now is the change in log-odds for a one unit increase in X_j, holding others fixed. Equivalently, e^{β_j} is the multiplicative change in the odds for a one-unit increase in X_j. For example, if $\beta_j = 0.7$, then odds are $e^{0.7} \approx 2.01$ times greater for a one-unit increase in X_j (meaning p roughly doubles, if in mid-range).
Logistic regression is fit by **maximum likelihood** rather than least squares (since it's not a linear model in Y). The likelihood for independent observations is

$$L(\beta_0, \boldsymbol{\beta}) = \prod_{i:y_i=1} p(x_i) \prod_{j:y_j=0} [1 - p(x_j)],$$

and we find $\hat{\beta}$ that maximize this (often via iterative algorithms).

The output in R often includes:

- Coefficients (values of β_j), measures relationship of each predictor with the response
- Standard error ($\text{SE}(\beta_j)$)
- z-statistics ($\frac{\hat{\beta}_j}{\text{SE}(\hat{\beta}_j)}$)
  - A large (absolute) value of the z-statistic indicates evidence against the null hypothesis $H_0: \ B_j = 0$ (meaning there is a meaningful relationship between the predictor X_j and Y)
- P-values. If p is tiny, we can reject $H_0 \implies$ there is a meaningful relationship between X_j and Y

We measure model fit by **deviance** or by classification performance metrics, since R^2 isn't meaningful here. One can compute a pseudo-R^2 or just measure accuracy.

Prediction rule: To make a classification, we predict class 1 if $\hat{p}(X) > 0.5$ (or another cutoff). 0.5 is the default threshold for equal costs. Sometimes we adjust the threshold for specific sensitivity or specificity needs (e.g., predict positive if probability > 0.2 to catch more positives at cost of more false alarms, as shown by altering threshold in slides for LDA; similar logic applies to logistic).

Making predictions: Once you have coefficients, we can compute the probability of event X (use probability of default as the example). E.g. if credit card balance $X = \$1,000$, $\hat{p}(X) = \frac{e^{\hat{\beta}_0+\hat{\beta}_1 X}}{1+e^{\hat{\beta}_0+\hat{\beta}_1 X}} = \frac{e^{-10.6513+0.0055\times 1000}}{1+e^{-10.6513+0.0055\times 1000}} = 0.00576$

Qualitative predictors (dummy variable encoding): Can fit a model and the predictor can be 0 if some condition is satisfied, and 1 if not, then can fit a simple logistic model based on that.

Pros:

- Outputs probabilities, which is more informative than just class labels (especially in insurance, you might use the probabilities for decision).
- Well-calibrated (if model is correct, predicted probabilities tend to reflect true frequencies).
- Inference is available (Wald tests via z-values).
- Can naturally extend to multiple classes (via multinomial logistic, not covered deeply but concept exists).

Cons:

- Assumes a linear log-odds relationship. If the true decision boundary is highly non-linear, logistic (without polynomial terms or interactions) will mis-specify.
- It's parametric; if parametric form is wrong, could be biased.
- In practice, if classes are not well separated, parameter estimates have large uncertainty (though still optimal in max likelihood sense).

6.3 Logistic Regression in R

```
library(ISLR)
?Smarket # info on dataset used

# Fitting a logistic regression (no test/train set)
glm.fits=glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
             data=Smarket,family=binomial) # binomial for LR
summary(glm.fits)
```

```
#  Coefficients:
#               Estimate Std. Error z value Pr(>|z|)
#  (Intercept) -0.126000   0.240736  -0.523    0.601
# Lag1        -0.073074   0.050167  -1.457    0.145
# Lag2        -0.042301   0.050086  -0.845    0.398
# Lag3         0.011085   0.049939   0.222    0.824
# Lag4         0.009359   0.049974   0.187    0.851
# Lag5         0.010313   0.049511   0.208    0.835
# Volume       0.135441   0.158360   0.855    0.392
# (Dispersion parameter for binomial family taken to be 1)
#    Null deviance: 1731.2  on 1249  degrees of freedom
#  Residual deviance: 1727.6  on 1243  degrees of freedom
# AIC: 1741.6
# Number of Fisher Scoring iterations: 3
coef(glm.fits)
summary(glm.fits)$coef
summary(glm.fits)$coef[,4] # Showing the associated p-values only
glm.probs=predict(glm.fits,type="response") # type="response" returns P(Y=1|X)
contrasts(Direction)
glm.pred=rep("Down",1250) # creating 1250 Down elements
glm.pred[glm.probs>.5]="Up" # transforming those elements whose prob.>0.5 to Up
table(glm.pred,Direction) # The confusion matrix
mean(glm.pred==Direction) # rate of correct predictions, i.e. 1-training error rate
mean(glm.pred!=Direction) # training error rate

# Fit the logistic regression using training set
train=(Year<2005)
Smarket.2005=Smarket[!train,] # selecting observations in 2005
dim(Smarket.2005)
Direction.2005=Direction[!train]
glm.fits=glm(Direction~Lag1+Lag2+Lag3+Lag4+Lag5+Volume,
             data=Smarket,family=binomial,subset=train) # subset=train for train data only
# Predict using test set
glm.probs=predict(glm.fits,Smarket.2005,type="response")
glm.pred=rep("Down",252)
glm.pred[glm.probs>.5]="Up"
table(glm.pred,Direction.2005)
mean(glm.pred==Direction.2005) #rate of correct predictions
mean(glm.pred!=Direction.2005) #test error rate
sum(glm.pred==Direction.2005 & glm.pred=="Down")
#Sensitivity
sum(glm.pred==Direction.2005 & glm.pred=="Up")/sum(Direction.2005=="Up")

# model improvement by removing weakest predictors
glm.fits=glm(Direction~Lag1+Lag2,data=Smarket,family=binomial,subset=train)
glm.probs=predict(glm.fits,Smarket.2005,type="response")
glm.pred=rep("Down",252)
glm.pred[glm.probs>.5]="Up"
table(glm.pred,Direction.2005)
mean(glm.pred==Direction.2005)
#Sensitivity
sum(glm.pred==Direction.2005 & glm.pred=="Up")/sum(Direction.2005=="Up")
# predict using new observations x1=(1.2, 1.1) and x2=(1.5, -0.8)
predict(glm.fits,newdata=data.frame(Lag1=c(1.2,1.1),Lag2=c(1.5,-0.8)),type="response")
```

6.4 Bayes' theorem for classification:

Suppose we wish to classify an observation into one of K classes ($K \geq 2$).

Let π_k denote the overall or *prior* probability that a randomly chosen observation comes from class k.

Let $f_k(x) = \mathbb{P}(X = x|Y = k)$ be the density function of X for an observation that comes from the kth class:

- $f_k(x)$ is relatively large if there is a high prob. that an observation in the kth class $X \approx x$;
- $f_k(x)$ is relatively small if it is unlikely that an observation in the kth class has $X \approx x$.

Then Bayes' theorem states that:

$$p_k(x) = \mathbb{P}(Y = k|X = x) = \frac{\pi_k f_k(x)}{\sum_{I=1}^{K} \pi_I f_I(x)}$$

We can estimate π_k and $f_k(x)$ seperately and then plug in those estimatees into the equation above to obtain an estimate of $p_k(x)$. *LDA*, *QDA* and *naive Bayes* all use different estimates of $f_k(x)$.

7 Linear/Quadratic Discriminant Analysis and KNN

7.1 LDA Theory

7.1.1 LDA for $p = 1$:

We only have 1 predictor, $f_k(x) = \frac{1}{\sqrt{2\pi}\sigma_k} e^{-\frac{(x-\mu_k)^2}{2\sigma_k^2}}$, assume that $\sigma_1^2 = ... = \sigma_k^2 = \sigma^2$. Therefore:

$$p_k(x) = \frac{\pi_k e^{-\frac{(x-\mu_k)^2}{2\sigma^2}}}{\sum_{I=1}^{K} \pi_I e^{-\frac{(x-\mu_k)^2}{2\sigma^2}}}$$

How to classify an observation $\boldsymbol{X = x}$: We take $\delta_k(x) = \ln(p_k(x))$ and try to maximise it. Re-arrange terms to get $\delta_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$.

E.g. If $K = 2$ and $\pi_1 = 0.5$. Find the condition that x needs to satisfy such that the Bayes classifier assigns observation $X = x$ to class 1:

$$\text{For } \mu_1 > \mu_2 :$$
$$\delta_1(x) > \delta_2(x) \Leftrightarrow x(\mu_1 - \mu_2) - \frac{1}{2}(\mu_1^2 - \mu_2^2) > 0$$
$$\Leftrightarrow x > \frac{\mu_1 + \mu_2}{2} \Leftrightarrow Y = 1$$

Clearly, if $\mu_1 < \mu_2$, then $x < \frac{\mu_1+\mu_2}{2}$. The point $x = \frac{\mu_1+\mu_2}{2}$ is the Bayes decision boundary.

Estimations used in LDA: LDA estimates parameters then plugs them into $\delta_k(x)$. Assume n is the total #observations in the training data set, and n_k is the number of training observations in the kth class.

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_k$$
$$\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^{K} \sum_{i:y_i=k} (x_i - \hat{\mu_k})^2$$
$$\hat{\pi}_k = \frac{n_k}{n}$$
$$\implies \hat{\delta}_k(x) = x \cdot \frac{\hat{\mu}_k}{\hat{\sigma}^2} - \frac{\hat{\mu}_k^2}{2\hat{\sigma}^2} + \log(\hat{\pi}_k), \text{ linear discriminant function of } x$$

The LDA classifier results from assuming that the observations within each class come from a normal distribution with a class-specific mean vector and common variance σ^2.

7.1.2 LDA for $p > 1$:

We have p-predictors, let $\boldsymbol{X} \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $\text{Cov}(\boldsymbol{X}) = \boldsymbol{\Sigma}$, $\implies$ the pdf:

$$f(\boldsymbol{x}) = \frac{1}{(2\pi)^{p/2}\sqrt{|\boldsymbol{\Sigma}|}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})\boldsymbol{\Sigma}^2(\boldsymbol{x}-\boldsymbol{\mu})^\top}$$

$$\delta_k(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k^\top - \frac{1}{2}\boldsymbol{\mu}_k^\top + \log(\pi_k)$$

Then we can estimate the unknown parameters like before, to classify a new observation $\boldsymbol{X} = \boldsymbol{x}$, LDA plugs these estimates into $\delta_k(\boldsymbol{x})$ and classifies to the class with the largest $\delta_k(\boldsymbol{x})$.

A binary classification problem example: $10,000$ observations in *Default* training data set, where 333 individuals defaulted. Perform LDA on the training data and the training error rate is 2.75%, some remarks:

- The training error rate being low doesn't mean the test error will be low as well, over fitting is a potential issue, in particular when p is close to n.
- Since only 3.33% of individuals in training data defaulted, the *null* classifier, which always predicts no defaulting, will achieve an training error rate of 3.33%, not much higher than the LDA training error rate.
- What does the credit card company really care about?

Confusion matrix:

		True default status		
		No	Yes	Total
Predicted default status	No	9644	252	9896
	Yes	23	81	104
	Total	9667	333	10000

TABLE 4.4. *A confusion matrix compares the LDA predictions to the true default statuses for the* 10,000 *training observations in the* `Default` *data set. Elements on the diagonal of the matrix represent individuals whose default statuses were correctly predicted, while off-diagonal elements represent individuals that were misclassified. LDA made incorrect predictions for* 23 *individuals who did not default and for* 252 *individuals who did default.*

- Training error rate is $(252 + 23)/10000 = 2.75\%$.
- Assigning not defaulted individuals to *default* error: $23/9667 = 0.24\%$.
- Assigning defaulted individuals to *no default* error: $252/333 = 75.7\%$.

Is the LDA classifier a satisfactory one?

- *Sensitivity* of the LDA classifier: $81/333 = 24.3\%$; very poor!
- *Specificity* of the LDA classifer: $9664/9667 = 99.8\%$; Great!
- Poor sensitivity so not a satisfactory classifier, since credit card company would usually care more about not incorrectly classifying an individual who will default as no default.

Can we modify the LDA to better meet the credit card company's need?

- Yes. We can modify the threshold used in the LDA approach to achieve the goal. The default threshold is 0.5, i.e. we assign an observation to the *default* class if $\mathbb{P}(\text{default} = \text{Yes}|\boldsymbol{X} = \boldsymbol{x}) > 0.5$.
- If threshold is now 20% sensitivity is better, specificity is still great which is really good although the overall error rate increases a little bit.

Threshold vs. Error Rate Curve:

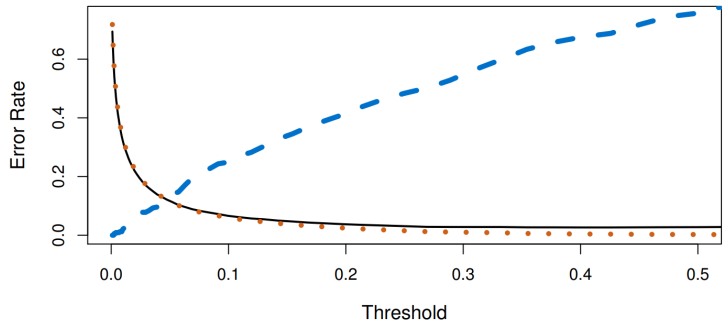


FIGURE 4.7. *For the* `Default` *data set, error rates are shown as a function of the threshold value for the posterior probability that is used to perform the assignment. The black solid line displays the overall error rate. The blue dashed line represents the fraction of defaulting customers that are incorrectly classified, and the orange dotted line indicates the fraction of errors among the non-defaulting customers.*

ROC (Receiver Operating Characteristics) Curve:
How to read: the overall performance of a classifier, summarised over all possible thresholds, is given by the *area under the (ROC) curve* (AUC). An ideal ROC curve will hug the top left corner, so the larger the AUC the better the classifier.

It plots the true positive rate against the false positive rate at various threshold settings associated with the classification algorithm.

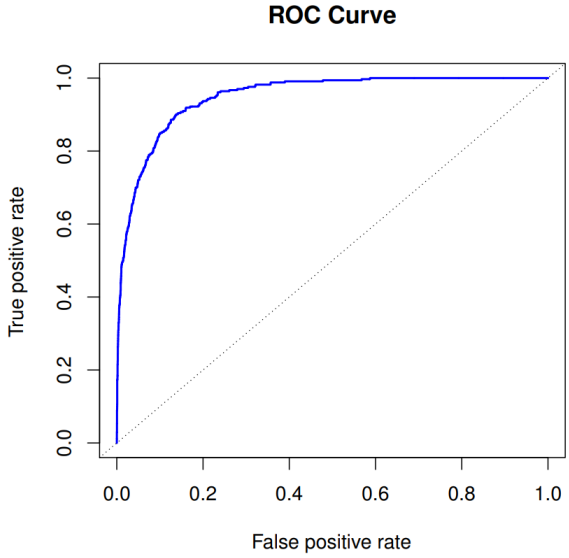


FIGURE 4.8. *A ROC curve for the LDA classifier on the* `Default` *data. It traces out two types of error as we vary the threshold value for the posterior probability of default. The actual thresholds are not shown. The true positive rate is the sensitivity: the fraction of defaulters that are correctly identified, using a given threshold value. The false positive rate is 1-specificity: the fraction of non-defaulters that we classify incorrectly as defaulters, using that same threshold value. The ideal ROC curve hugs the top left corner, indicating a high true positive rate and a low false positive rate. The dotted line represents the "no information" classifier; this is what we would expect if student status and credit card balance are not associated with probability of default.*

7.1.3 ROC in R

```r
# Load necessary libraries
library(mlbench)       # For dataset
library(caret)         # For data splitting
library(ROCR)          # For ROC and AUC

# Load the dataset
data(PimaIndiansDiabetes)
df <- PimaIndiansDiabetes

# Split into training and testing sets
set.seed(123)
trainIndex <- createDataPartition(df$diabetes, p = 0.7, list = FALSE)
trainData <- df[trainIndex, ]
testData <- df[-trainIndex, ]

# Train logistic regression model
model <- glm(diabetes ~ ., data = trainData, family = binomial)

# Predict probabilities on the test set
probabilities <- predict(model, newdata = testData, type = "response")

# Create a prediction object
pred <- prediction(probabilities, testData$diabetes)
```

```r
# Performance metrics (ROC curve and AUC)
perf <- performance(pred, "tpr", "fpr")
auc <- performance(pred, "auc")@y.values[[1]]

# Plot the ROC curve
plot(perf, main = paste("ROC Curve (AUC =", round(auc, 3), ")"), col = "blue", lwd = 2)
abline(a = 0, b = 1, lty = 2, col = "gray")
```

7.2 Confusion Matrices and Measures for Classification

		True class		
		− or Null	+ or Non-null	Total
Predicted class	− or Null	True Neg. (TN)	False Neg. (FN)	N^*
	+ or Non-null	False Pos. (FP)	True Pos. (TP)	P^*
	Total	N	P	

TABLE 4.6. *Possible results when applying a classifier or diagnostic test to a population.*

Name	Definition	Synonyms
False Pos. rate	FP/N	Type I error, 1−Specificity
True Pos. rate	TP/P	1−Type II error, power, sensitivity, recall
Pos. Pred. value	TP/P^*	Precision, 1−false discovery proportion
Neg. Pred. value	TN/N^*	

TABLE 4.7. *Important measures for classification and diagnostic testing, derived from quantities in Table 4.6.*

7.3 Quadratic Discriminant Analysis (QDA):

Assumes that an observation from the kth class is one of the form $\boldsymbol{X} \sim N(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, so unique covariance matrices for each class. Therefore we can derive:

$$\begin{aligned}\delta_x(\boldsymbol{x}) &= -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)\boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}-\boldsymbol{\mu_k})^\top - \frac{1}{2}\log|\boldsymbol{\Sigma}_k| + \log(\pi_k) \\ &= -\frac{1}{2}\boldsymbol{x}\boldsymbol{\Sigma}_k^{-1}\boldsymbol{x}^\top + \boldsymbol{x}\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k^\top - \frac{1}{2}\boldsymbol{\mu}_k\boldsymbol{\Sigma}_k^{-1}\boldsymbol{\mu}_k^\top - \frac{1}{2}\log|\boldsymbol{\Sigma}_k| + \log(\pi_k)\end{aligned}$$

Then we plug in estimates for $\boldsymbol{\Sigma}_k$ and $\boldsymbol{\mu}_k$ to assign an observation to the class for which this quantity is the largest, $\boldsymbol{x}$ appears as a quadratic function in the equation above so that's why its a QDA.

7.3.1 When to use QDA or LDA?

The answer depends on the bias-variance trade-off:

- LDA is much less flexible classifier than QDA, which lead to lower variance. Since, for p predictors, QDA estimates $K\frac{p(p+1)}{2}$ params. and LDA estaimtes $\frac{p(p+1)}{2}$ params.
- If LDA's assumption on common covariance matrix is not appropriate, then it will result in higher bias
- LDA tends to be a better bet than QDA if there are relatively fewer training observations and so reducing variance is crucial.
- QDA is recommended if the training set is very large, or if the assumption of common covariance matrix for the K classes is clearly untenable (not that great).

7.4 LDA in R

```
library(ISLR)
attach(Smarket)
train=(Year<2005)
Smarket.2005=Smarket[!train,] #selecting observations in 2005
Direction.2005=Direction[!train]

library(MASS)
lda.fit=lda(Direction~Lag1+Lag2,data=Smarket,subset=train)
lda.fit
# Call:
# lda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
# Prior probabilities of groups:
#     Down       Up
# 0.491984 0.508016
# Group means:
#             Lag1        Lag2
# Down  0.04279022  0.03389409
# Up   -0.03954635 -0.03132544
# Coefficients of linear discriminants:
#             LD1
# Lag1 -0.6420190
# Lag2 -0.5135293
plot(lda.fit)
lda.pred=predict(lda.fit, Smarket.2005)
names(lda.pred) # "class", "posterior", "x"
lda.pred$posterior[1:10,] #estimated posterior probabilities
lda.pred$class[1:10] #class predictions
lda.class=lda.pred$class
confusion <- table(lda.class,Direction.2005) # Confusion matrix
mean(lda.class==Direction.2005) # True rate
(confusion[1,1]+confusion[2,2])/length(lda.class) # Manual true rate
#recreate predictions using a threshold 0.5
sum(lda.pred$posterior[,1]>=0.5) # Pr(y=down|x)>=0.5
sum(lda.pred$posterior[,1]<.5) # Pr(y=down|x)<0.5
#change threshold
sum(lda.pred$posterior[,1]>0.49)
```

7.5 QDA in R

```
par(mfrow=c(2,1))
hist(Lag1)
hist(Lag2)
par(mfrow=c(1,1))
qda.fit=qda(Direction~Lag1+Lag2,data=Smarket,subset=train)
qda.fit
# Call:
# qda(Direction ~ Lag1 + Lag2, data = Smarket, subset = train)
# Prior probabilities of groups:
#     Down       Up
# 0.491984 0.508016
# Group means:
#             Lag1        Lag2
# Down  0.04279022  0.03389409
# Up   -0.03954635 -0.03132544
qda.class=predict(qda.fit,Smarket.2005)$class # predict then get the classes
```

```r
table(qda.class,Direction.2005) # Confusion matrix
mean(qda.class==Direction.2005) # True rate
```

7.6 K-Nearest Neighbors (KNN) Theory

KNN is a non-parametric classification method: To predict class for an observation x_0, find the K training observations closest to x_0 (in Euclidean distance, typically). Then have them vote (majority class among those K neighbors). The estimated probability for class j is the fraction of the K neighbors that belong to class j:

$$\Pr(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j),$$

where N_0 is the index set of the K nearest neighbors of x_0. Then assign the class with highest estimated probability.

7.6.1 Choosing a K Value

- KNN is very flexible: as $K \to n$, it becomes very smooth (predict majority of entire training set = always most frequent class, high bias, low variance).
- As $K \to 1$, it can get very wiggly (each point is basically its own neighborhood, can overfit noise, low bias, high variance).
- We choose K by cross-validation typically.

7.6.2 Pros an cons of KNN approach

- **Pros:**
  1. The algorithm is simple and easy to implement.
  2. There is no need to build a model, tune several parameters, or make additional assumptions (like shape of decision boundary).
  3. The algorithm is versatile. It can be used for classification and regression.
  4. Effective if n is very large (lots of data) and relationship is hard to parametric form.
- **Cons:**
  1. No explicit model, so no interpretability beyond "these neighbors vote".
  2. Doesn't give confidence intervals or the like, though one can derive some asymptotic analysis if needed.
  3. Slow if n is large (need to compute distances to all points for each prediction, though can use indexing structures to speed up).
  4. Curse of dimensionality. KNN works poorly if p is large, because distance in high dimension becomes less meaningful (all points become far). Requires feature scaling or selection and often needs a lot of data.

7.7 KNN in R

```r
library(class) # library for KNN
train.X=cbind(Lag1,Lag2)[train,]
test.X=cbind(Lag1,Lag2)[!train,]
train.Direction=Direction[train] # get class of train data (in Smarket)
set.seed(5) # ties as nearest neighbors will be randomly broken
knn.pred=knn(train.X,test.X,train.Direction,k=58) # use k=58
table(knn.pred,Direction.2005) # Confusion matrix
mean(knn.pred==Direction.2005) # True rate
```

```r
knn.rp = rep(0, 101) # test a bunch of k's
for(i in 1:101){
  set.seed(5)
  knn.pred=knn(train.X,test.X,train.Direction,k=i)
  knn.rp[i]=mean(knn.pred==Direction.2005)
}
which.max(knn.rp) # k=58 is the best (highest true rate)

# how to use knn.cv() to select best K
knn.rp = rep(0, 101)
for(i in 1:101){
  set.seed(5)
  knn.pred.cv=knn.cv(train.X,train.Direction,k=i,p=1) # Leave-p-out CV
  knn.rp[i]=mean(knn.pred.cv==train.Direction)
}
k.cv=which.max(knn.rp) # gives k=33 which isn't as great as above
```

7.8 Plotting results of LDA/QDA/KNN in R

```r
# Draw the decision boundaries
#create new data
np = 100
mydata = Smarket[train,]
nd.x = seq(from = min(mydata$Lag1), to = max(mydata$Lag1), length.out = np)
nd.y = seq(from = min(mydata$Lag2), to = max(mydata$Lag2), length.out = np)
nd = expand.grid(Lag1 = nd.x, Lag2 = nd.y)
# Run LDA, QDA, KNN and predict using new data
prd.lda = as.numeric(predict(lda.fit, newdata = nd)$class)
prd.qda = as.numeric(predict(qda.fit, newdata = nd)$class)
set.seed(1)
prd.knn = as.numeric(knn(train.X,nd,train.Direction,k=55)) # k takes odd values
# LDA graph
plot(mydata[, 2:3], col = mydata$Direction)
points(nd, cex = 0.05, col = prd.lda)
contour(x = nd.x, y = nd.y, z = matrix(prd.lda, nrow = np, ncol = np),
        levels = c(1, 2), add = TRUE, drawlabels = FALSE)
#QDA graph
plot(mydata[, 2:3], col = mydata$Direction)
points(nd, cex = 0.05, col = prd.qda)
contour(x = nd.x, y = nd.y, z = matrix(prd.qda, nrow = np, ncol = np),
        levels = c(1, 2), add = TRUE, drawlabels = FALSE)
#KNN graph
plot(mydata[, 2:3], col = mydata$Direction)
points(nd, cex = 0.05, col = prd.knn)
contour(x = nd.x, y = nd.y, z = matrix(prd.knn, nrow = np, ncol = np),
        levels = c(1, 2), add = TRUE, drawlabels = FALSE)
```

7.9 Comparison of Logistic regression and LDA

Both methods produce a linear decision boundary, the only difference between the two approaches is that β_0 and β_1 are estimated by MLE, whereas c_0 and c_1 are computed using the estaimtes mean and variance from a normal distribution (also holds for $p > 1$).

Proof: logistic regression $\log(\frac{p_1(x)}{1-p_1(x)}) = \beta_0 + \beta_1 x$ and LDA $\log(\frac{p_1(x)}{1-p_1(x)}) = \log(\frac{p_1(x)}{p_2(x)}) = \log\left(\frac{\pi_1 e^{-\frac{(x-\mu_1)^2}{2\sigma^2}}}{\pi_2 e^{\frac{(x-\mu_2)^2}{2\sigma^2}}}\right) =$

$\log(\pi_1) - \log(\pi_2) + \frac{\mu_2^2+\mu_1^2}{2\sigma^2} + \frac{\mu_1-\mu_2}{\sigma^2}x = c_0 + c_1 x$

They do not always produce the similar results, since LDA requires a normal assumption with a common covariance matrix. Logistic regression can outperform LDA if this assumption is not met, and vice versa.

7.10 KNN, QDA vs Logistic regression and LDA

- KNN is completely non-parametric method, and no assumptions are needed about the shape of the decision boundary.
- When decision boundary is highly non-linear, KNN is likely to outperform LDA and logistic regression (as they will have higher bias)
- KNN doesn't tell us which predictors are important: no inference results for KNN (if n is sufficient lower bias but might have higher variance)
- QDA is a compromise with a quadratic decision boundary. Not as flexible as KNN but QDA can perform better in the presence of a limited training dataset, as it makes assumptions of form of decision boundary.

7.11 Six scenarios of classification comparison ($p = 2$)

1. 20 training observations in each of two classes. Observations were uncorrelated normal r.v.'s with a different mean in each class.
   - Suits LDA assumptions very well
   - Logistic regression performance similar to LDA, and QDA is slightly worse as its more flexible than LDA
   - KNN didn't perform well, as it didn't have much advantage on bias reduction in this linear decision boundary case
2. Same as S1, except that the two predictors had a correlation of -0.5
   - Similar to S1 performances
3. X_1 and X_2 from t-distribution, 50 observations per class. Decision boundary is still linear, and so fit into LR framework.
   - LR worked very well
   - Normal assumption are not met here, LDA worked slightly worse than LR
   - QDA method deteriorated since non-normality
   - KNN didn't perform well, however KNN-CV improved moderately
4. Normal dist. data, corr of 0.5 between predictors in the first class, and corr of -0.5 between predictors in the second class.
   - QDA assumption, and results in a quadratic decision boundary. So best method
5. Normal dist. with uncorrelated predictors. However, responses were samples from logistic function using X_1^2 and X_2^2, and $X_1 \times X_2$ as predictors. (Quadratic decision boundary)
   - QDA outperforms linear method
   - KNN-CV had improved performance as the data became more complex
6. Same as S5, responses were sampled from more complicated non-linear function.
   - KNN-CV method outperformed others
   - QDA couldn't adequately model the data
   - KNN-1 performed poorly (since level of smoothness not chosen correctly)

7.12 Summary of LDA vs QDA vs Logistic Regression vs KNN

- When true decision boundary is linear (or close to), LR and LDA tend to work better. If normality is absent, then LE outperforms LDA.

- When true decision boundary is quadratic or moderate non-linear, QDA would outperform LR and LDA
- KNN-CV is the most robust method. Performs relatively well in most cases. Outperforms when true decision boundary is very non-linear and the normality assumption is not met.
- KNN-1 is the worst performed approach in most cases.

8 Trees and Support Vector Machines

8.1 Decision Trees Theory (Classification Trees)

A **decision tree** is a hierarchical model that partitions the feature space into rectangles (or more complex regions) and fits a simple model (constant for regression, majority vote for classification) in each region.

How to construct the regions: Divide the predictor space into high-dimensional boxes. Goal is to find boxes $R_1, ..., R_j$ that minimise the RSS, given by: $\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$. Where $\hat{y}_{R_j}$ is the mean response for the training observations within the jth box. Then take *top-down* or *recursive binary splitting* (*greedy*) approach.

- *Top-down*: begin at top of tree, successively splits the predictor space; each split indicated by two new branches further down;
- *Greedy*: each step of tree-building process makes the best split at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some further step.

Recursive binary splitting:

1. Select X_j and the particular way of splitting its predictor space which leads to greatest possible reduction in RSS.
   - i Examine all predictors $X_1, ..., X_p$ and all possible ways of splitting the predictor space of each predictor
     - For any numerical predictor X_j and cutpoint s, we define: $R_1(j, s) = \{\boldsymbol{X} | X_j < s\}$ and $R_2(j, s) = \{\boldsymbol{X} | X_j \geq s\}$.
     - For any categorical predictor X_j and any subset of its classes s, we define: $R_1(j, s) = \{\boldsymbol{X} | X_j \in s\}$ and $R_2(j, s) = \{\boldsymbol{X} | x_j \notin s\}$.
   - ii Seek the value of j and s that minimise the expression: $\sum_{i:\boldsymbol{x} \in R_1(j,s)} (y_i - \hat{y}_{R_1(j,s)})^2 + \sum_{i:\boldsymbol{x} \in R_2(j,s)} (y_i - \hat{y}_{R_2(j,s)})^2$
2. Repeat the process, looking for the best predictor and best ways of splitting the data further so as to minimise the RSS within each of the resulting regions. Instead of splitting the entire predictor space, we split one of the two previously identified regions. As a result, we have on more region.
3. Then look to split one of the previously constructed regions, so as to minimise the RSS. This process continues until a stopped criterion is reached (e.g. no region contains more than 5 observations.
4. Output: $\hat{f}_{\text{tree}}(\boldsymbol{x}) = \sum_{m=1}^{M} \text{ave}(y_i | \boldsymbol{x}_i \in R_m) I_{\boldsymbol{x} \in R_m}$. Where M is the number of terminal nodes of the generated tree.

Remarks on binary recursive splitting:

- Finding the optimal split:
  - Is straightforward for continuous predictors since the data can be ordered in a natural way
  - Easy for binary predictors since there is only one possible split point
  - More complicated for predictors with more than two categories. Indeed, for a predictor with q unordered categories, there are $2^{q-1} - 1$ possible partitions of the q categories into two groups. So for large q, the computation can be very time consuming.
- Why binary splitting:

- Multi way splits fragment the data too quickly ⇒ leaving insufficient data at the next level down.
- Also, multi way splits can be achieved by a series of binary splits.

8.1.1 Tree Pruning

- Recursive binary splitting is likely to overfit the data, leading to poor test data performance, as the resulting tree might be too complex.
- A smaller tree with fewer splits (lower flexibility) might lead to lower variance and better interpretation at the cost of little bias.
- One alternative: build the tree only as the decrease in the RSS due to each split exceeds some threshold. Drawback: A seemingly worthless split early on in the tree might be followed by a very good split – a split that leads to a large reduction in RSS later on.
- A better alternative: to grow a very large tree T_0, and then prune it back in order to obtain a *subtree*.

Cost complexity pruning:

- Instead of considering every possible subtree, consider a sequence of trees indexed by a non-negative tuning parameter α
- For each α, there is a subtree $T \subset T_0$ such that: $\sum_{m=1}^{|T|} \sum_{i:x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T|$ is small as possible. $|T|$ is as number of terminal nodes of tree T, R_m is the box corresponding to the mth terminal node.

The effect of tuning parameter α in cost complexity pruning

- α controls trade-off between the subtree's complexity and its fit to the training data.
- When $\alpha = 0$, $T = T_0$.
- When α increases, the formula above tends to be minimised for a smaller subtree.
- Select α using a validation set or using cross-validation.

8.1.2 Building a regression tree – an algorithm

1. Use recursive binary splitting to grow a large tree on training data. Stop when each terminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. Average the following results for each value of α, and pick α to minimse the average error.
   (a) Repeat steps 1 and 2 on all but the kth fold of the training data.
   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α
4. Return the subtree from Step 2 that corresponds to the chosen value of α

8.1.3 RSS vs classification error rate

Can't use RSS. Use classification error rate, which is the fraction of the training observations in a region that do not belond to the most common class:

$$E = 1 - \max_k (\hat{p}_{mk})$$

Where $\hat{p}_{mk}$ is the proportion of training observations in the mth region are from the kth class.

Alternatives to E:

- *Gini index*: $G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$, measure of total variance across the K classes.
  - Small G if all $\hat{p}_{mk}$ are close to either 0 or 1. So small G indicates that a terminal node contains predominantly observations from a single class.
- *Entropy*: $D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}) \geq 0$.
  - Small D if all $\hat{p}_{mk}$ are close to 0 or 1. Small D indicates that the mth node is pure (a single class dominates the node).

When to use which classification error rate:

- When *building* a classification tree, either the *Gini index* or *entropy* can be used to evaluate the quality of a particular split. As they are more sensitive to node purity than the classification error rate.
- When *pruning* a classification tree, any measure can be used.
- if *prediction accuracy* of the final pruned tree is the goal, then the classification error rate is more preferable.

8.1.4 Advantages and disadvantages of trees

Advantages:

- Trees are easily explainable. They mirror human decision making.
- Can be explained graphically, and are easily interpreted even by a non-expert.
- Can easily handle qualitative predictors without the need for dummy variables.

Disadvantages:

- Do not have the same level of predictive accuracy as other regression and classification approaches.
- Can be very non-robust. A small change in the data can cause a large change in the final esimated tree.

8.2 Decision Trees in R

8.2.1 Fitting classification trees

```
library(tree) # for the trees
library(ISLR2) # for the dataset
attach(Carseats)
High <- factor(ifelse(Sales <= 8, "No", "Yes"))
Carseatsnew <- data.frame(Carseats, High) # Merge "High" with "Carseats"
tree.carseats=tree(High~.-Sales, data=Carseatsnew)
summary(tree.carseats)
# Classification tree:
# tree(formula = High ~ . - Sales, data = Carseatsnew)
# Variables actually used in tree construction:
# [1] "ShelveLoc" "Price" "Income" "CompPrice"
# [5] "Population" "Advertising" "Age" "US"
# Number of terminal nodes: 27
# Residual mean deviance: 0.4575 = 170.7 / 373
# Misclassification error rate: 0.09 = 36 / 400
plot(tree.carseats)
text(tree.carseats,pretty=0) # incl. category names for any qualitative predictions
tree.carseats # terminal node branches are given a *
# node), split, n, deviance , yval, (yprob)
# * denotes terminal node
```

```
# 1) root 400 541.5 No ( 0.590 0.410 )
# 2) ShelveLoc: Bad,Medium 315 390.6 No ( 0.689 0.311 )
# 4) Price < 92.5 46 56.53 Yes ( 0.304 0.696 )
# 8) Income < 57 10 12.22 No ( 0.700 0.300 )

########

# Predictions and confusion matrix
set.seed(2)
train=sample(1:nrow(Carseats_new), 200) #training subset
Carseats.test=Carseats_new[-train,] #test subset
High.test=High[-train]
tree.carseats=tree(High~.-Sales,Carseats_new,subset=train)
tree.pred=predict(tree.carseats,Carseats.test,type="class")
table(tree.pred,High.test)
#           High.test
# tree.pred No Yes
#       No 104 33
#       Yes 13 50
mean(tree.pred==High.test) #1-test error rate
# 0.77

########

# Perform CV to determine optimal level of tree complexity (cost complexity pruning)
set.seed(3)
#FUN=prune.misclass uses classification error rate to guide the CV
cv.carseats=cv.tree(tree.carseats,FUN=prune.misclass)
cv.carseats
# $size
# [1] 21 19 14  9  8  5  3  2  1
# $dev <- no. of CV errors. 8 terminal nodes = 75 CV errors (also k=2), so best.
# [1] 74 76 81 81 75 77 78 85 81
# $k
# [1] -Inf  0.0  1.0  1.4  2.0  3.0  4.0  9.0 18.0
# $method
# [1] "misclass"
# attr(,"class")
# [1] "prune"         "tree.sequence"
best_size <- cv.carseats$size[which.min(cv.carseats$dev)]
plot(cv.carseats$size,cv.carseats$dev,type="b") # plot error rate as a function of size
plot(cv.carseats$k,cv.carseats$dev,type = "b") # plot error rate as a function of k
# now prune tree to best size 8 (eight-node tree)
prune.carseats=prune.misclass(tree.carseats,best=8)
plot(prune.carseats)
text(prune.carseats,pretty=0)
tree.pred=predict(prune.carseats,Carseats.test,type="class")
table(tree.pred,High.test) # confusion table
mean(tree.pred==High.test) # correct rate
# OR can use k (cost-complexity parameter to prune)
prune.carseats1=prune.misclass(tree.carseats,k=2) #k=10 gives 2 nodes
summary(prune.carseats1)
plot(prune.carseats1)
text(prune.carseats1,pretty=1)
```

8.2.2 Fitting regression trees

```r
library(MASS)
set.seed(1)
train = sample(1:nrow(Boston), nrow(Boston)/2)
tree.boston=tree(medv~.,Boston,subset=train)
summary(tree.boston)
plot(tree.boston)
text(tree.boston,pretty=0)
cv.boston=cv.tree(tree.boston)
plot(cv.boston$size,cv.boston$dev,type="b")
prune.boston=prune.tree(tree.boston,best=5)
plot(prune.boston)
text(prune.boston,pretty=0)
yhat=predict(tree.boston,newdata=Boston[-train,])
boston.test=Boston[-train,"medv"] #response in test set
plot(yhat,boston.test)
abline(0,1)
mean((yhat-boston.test)^2)
```

8.3 Maximal Margin Classifier (Hyperplane Classification)

A hyperplane: In a $p-$dimensional space, a hyperplane is a flat affine subspace of the dimension $p-1$. Given by $\beta_0 + \beta_1 X_1 + ... + \beta_p X_p = 0$.

Classification using hyperplanes: For equation above, for point $X = (X_1, ..., X_p)^\top$, if it $= 0$ then X lies on hyperplane. If its > 0 its on one side of hyperplane, if its < 0 its on the other side.

Example with classes $y_i \in \{-1, 1\}$: If $\beta_0 + \beta_1 X_{i1} + ...\beta_p X_{ip} > 0$ if $y_i = 1$ and $\beta_0 + \beta_1 X_{i1} + ...\beta_p X_{ip} > 0$ if $y_i = -1$. Therefore a separating hyperplane has the property that $y_i(\beta_0 + \beta_1 X_{i1} + ... + \beta_p X_{ip}) > 0$, for $i = 1, ..., n$. Main **assumption** is that a separating hyperplane exists.

Confidence of classification of test observations: If test observation is x^*, let $f(x^*) = \beta_0 + \beta_1 X_1^* + ... + \beta p X_p^*$. If $f(x^*)$ is far from 0, x^* lies far from the hyperplane and we can be confident in our class assignment for x^*. Conversely, if $f(x^*)$ is close to 0, x^* is near the hyperplane and we are less certain above the class assignment for x^*.

The Maximal Margin Classifier: The separating hyperplane that is farthest from the training observations. Can also lead to over fitting when p is large. The maximal margin hyperplane depends directly on the support vectors, but not on the other observations which is very wasteful.

Construction of the Maximal Margin Classifier: If n training observations and associated class labels $y_1, ..., y_n \in \{-1, 1\}$. Maximal margin hyperplane is solution to the optimisation problem:

$$\max_{\beta_0, \beta_1, ..., \beta_p, M} M \tag{3}$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1 \tag{4}$$

$$y_i(\beta_0 + \beta_1 X_{i1} + ... + \beta_p X_{ip}) \geq M, \forall i = 1, ..., n \tag{5}$$

Worded explanation:

- (5) makes sure each observation will be on correct side of hyperplane (given $M > 0$)

- The constraint (4) adds meaning to (5), as this constraints means the perpendicular distance from the ith observation to the hyperplane is given by $y_i(\beta_0 + \beta_1 X_{i1} + ... + \beta_p X_{ip})$
- Also (4) and (5) ensure that each observation is on the correct side of the hyperplane and at least a distance M from the hyperplane (thats why we want to max M in (3)).

8.4 Support Vector Classifiers

Issues with maximal margin classifier: In many cases no separating hyperplane exists (no solution with $M > 0$) of the optimisation problem. If separating hyperplane exists it may not be desirable (as its extremely sensitive to a change in a single observation suggesting it may have over fit the data).

Soft Margin Classifier: (same as support vector classifier). Instead of seeking largest possible margin, now we allow some observations to be on the incorrect side of the margin, or even the incorrect side of the hyperplane.

Construction of Support Vector Classifier: Need to optimise the following problem:

$$\max_{\beta_0,\beta_1,...,\beta_p,\epsilon_1,...,\epsilon_n,M} M \tag{6}$$

$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1 \tag{7}$$

$$\epsilon_i \geq 0, \sum_{i=1}^{n} \epsilon_i \leq C \; (C \geq 0) \tag{8}$$

- M is width of margin, so we want to max this
- $\epsilon_1, ..., \epsilon_n$ are *slack variables* allowing observation to be on wrong side.
  - $\epsilon_i = 0$: ith observation is on the right side of the margin (default)
  - $0 < \epsilon < 1$: ith observation has violated the margin, but is on the right side of the hyperplane
  - $\epsilon > 1$: ith observation is on the wrong side of the hyperplane
- C is the budget for the amount that the margin can be violated by the n observations.
  - $C = 0$: no tolerance of violations to the margin
  - $C > 0$: no more than C observations can be on the wrong side of the hyperplane
  - As C increases, we become more tolerant of violations to the margin, and so the margin will widen.
  - Choose C using cross-validation

Role of C in bias-variance trade-off (opposite interpretation in R called *cost*!):
- When C is small (cost is large), we seek narrow margins that are rarely violated. Classifier that is highly fit to data, low bias but high variance.
- When C is larger (cost is small), margin is wider and we allow more violations. Easier to obtain a classifier, potentially more biased but may have lower variance.
- *In exam explain cost, rather then explaining C

8.5 Support Vector Machines

Enlarge the feature space to get a non-linear decision boundary. Can be done by using: higher-order polynomial terms, interaction terms of the form $X_j X_{j'}$ for $j \neq j'$, or other functions of the predictors rather than polynomials. Computations could become unmanageable if lots.

Inner product $\langle x_i, x_{i'} \rangle = \sum_{j=1}^{p} x_{ij} x_{i'j}$. Linear support vector classifier: $f(x) = \beta_0 + \sum_{i=1}^{n} \alpha_i \langle x, x_i \rangle$.

Generalised support vector classifier, with *Kernel* $K(x_i, x_{i'})$, and S be collection of indices of these support points: $f(x) = \beta_0 + \sum_{i \in S} K(x, x_i)$.

Types of Kernel functions:

- *Linear kernel*: $K(x_i, x_{i'}) = \langle x_i, x_{i'} \rangle$. Support vector classifier case with a linear decision boundary.
- *Polynomial kernel*: $K(x_i, x_{i'}) = (1 + \sum_{j=1}^{p} x_{ij} x_{i'j})^d$ for $d \geq 1$.
- *Radial kernel*: $K(x_i, x_{i'}) = e^{-\gamma \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2}$, $\gamma > 0$.
  - High γ ($\gamma = 10, 100$):
    - Tight decision boundary
    - Risk of over fitting
    - Each point influences only very close neighbours
  - Low γ ($\gamma = 0.01, 0.001$):
    - Each point influences a wider region
    - Smoother, more general decision boundary
    - Risk of under fitting

8.5.1 SVM with More than Two Classes

1. One-Versus-One Classification
   - Suppose there is $K > 2$ classes. Constructs $\binom{K}{2}$ SVM's, each of which compares a pair of classes
   - e.g. One SVM might compare the kth class, coded as $+1$, to the $k^{'}$th class, coded as -1
   - Classify a test observation using each of the $\binom{K}{2}$ classifiers, tally the number of times that the test observation is assigned to each of the K classes
   - Final classification is performed by assigning the test observation to the class to which it was most frequently assigned in these $\binom{K}{2}$ pairwise classifications.
2. One-Versus-All Classification
   - Suppose there is $K > 2$ classes. Fit K SVM's each time comparing one of the K classes to the remaining $K - 1$ classes
   - Let $\beta_{0k}, \beta_{1k}, ..., \beta_{pk}$ denote the result from fitting an SVM comparing the kth class (coded as $+1$) to the others (coded as -1)
   - Let x^* denote a test observation. Assign the observation to the class for which $\beta_{0k} + \beta_{1k} x_1^* + ... + \beta_{pk} x_p^*$ is the largest. Giving high level of confidence that the test observation belonds to the kth class rather than to any of the other classes.

8.5.2 Pros and Cons of SVM's

- **Pros:**
  - Works really well with a clear margin of separation
  - Effective in high dimensional spaces
  - Only uses a subset of training data in decision function (i.e. support vectors), so it is also memory efficient.
  - Robust to outliers
  - Can capture non-linear relationships
- **Cons:**
  - Doesn't perform well when large data set since training time is higher
  - Doesn't perform well when data set has noise (i.e. trarget classes are overlapping)
  - SVM doesn't directly provide probability estimates

– Hard to interpret (though one can examine support vectors)

8.6 SVMs in R

8.6.1 Support vector classifiers

```r
# Create data
set.seed(1)
x <- matrix(rnorm(20 * 2), ncol = 2)
y <- c(rep(-1, 10), rep(1, 10))
x[y == 1, ] <- x[y == 1, ] + 1
plot(x, col = (3 - y))
dat <- data.frame(x = x, y = as.factor(y))
library(e1071)
# Choices for kernel include "linear", "polynomial", "radial basis", and "sigmoid"
# cost: cost of constraints violation (default: 1)
svmfit.1 <- svm(y ~ ., data = dat, kernel = "linear",
    cost = 10, scale = FALSE)
plot(svmfit.1, dat)
# The index of the resulting support vectors in the data matrix.
svmfit.1$index
summary(svmfit.1)
# Visually graphing classes, points, and supporting vectors
make.grid = function(x, n = 75) {
    grange = apply(x, 2, range)
    x1 = seq(from = grange[1,1], to = grange[2,1], length = n)
    x2 = seq(from = grange[1,2], to = grange[2,2], length = n)
    expand.grid(x.1 = x1, x.2 = x2)
}
xgrid = make.grid(x)
ygrid = predict(svmfit.1, xgrid)
plot(xgrid, col = c("red","blue")[as.numeric(ygrid)], pch = 20, cex = .2)
points(x, col = y + 3, pch = 19)
points(x[svmfit.1$index,], pch = 5, cex = 2)

# To get back the linear coefficients in the hyperplane.
beta = drop(t(svmfit.1$coefs)%*%x[svmfit.1$index,])
beta0 = svmfit.1$rho

# Draw the linear decision boundary with the margins
abline(beta0 / beta[2], -beta[1] / beta[2])
abline((beta0 - 1) / beta[2], -beta[1] / beta[2], lty = 2)
abline((beta0 + 1) / beta[2], -beta[1] / beta[2], lty = 2)

# tune(): This generic function tunes hyperparameters of statistical methods
# by default uses 10-fold CV
set.seed(1)
tune.out <- tune(svm, y ~ ., data = dat, kernel = "linear",
    ranges = list(cost = c(0.001, 0.01, 0.1, 1, 5, 10, 100)))
summary(tune.out)
bestmod <- tune.out$best.model
summary(bestmod)
# Testing best model from tune()
xtest <- matrix(rnorm(20 * 2), ncol = 2)
ytest <- sample(c(-1, 1), 20, rep = TRUE)
xtest[ytest == 1, ] <- xtest[ytest == 1, ] + 1.5
testdat <- data.frame(x = xtest, y = as.factor(ytest))
ypred.best <- predict(bestmod, testdat)
```

```
table(predict = ypred.best, truth = testdat$y)
```

8.6.2 Support vector machines

```
library(e1071)
# Make data
set.seed(1)
x <- matrix(rnorm(200 * 2), ncol = 2)
x[1:100, ] <- x[1:100, ] + 2
x[101:150, ] <- x[101:150, ] - 2
y <- c(rep(1, 150), rep(2, 50))
dat <- data.frame(x = x, y = as.factor(y))
plot(x, col = y)
train <- sample(200, 100)
plot(x[train,], col = y[train])
# Fit "radial" kernel and sketch the regions
svmfit <- svm(y ~ ., data = dat[train, ], kernel = "radial",
              gamma = 1, cost = 1)
plot(svmfit, dat[train, ])

# Use tune for a range of cost and gamma
set.seed(1)
tune.out <- tune(svm, y ~ ., data = dat[train, ],
                 kernel = "radial",
                 ranges = list(
                   cost = c(0.1, 1, 10, 100, 1000),
                   gamma = c(0.5, 1, 2, 3, 4)
                 )
)
summary(tune.out)
# Construct the confusion matrix using the test data
table(
  true = dat[-train, "y"],
  pred = predict(
    tune.out$best.model, newdata = dat[-train, ]
  )
)
```

8.6.3 ROC curves

```
library(ROCR)
# Function for ROC PLOT
rocplot <- function(pred, truth, ...) {
  predob <- prediction(pred, truth)
  perf <- performance(predob, "tpr", "fpr")
  plot(perf, ...)
}

# Plot ROC
svmfit.opt <- svm(y ~ ., data = dat[train, ],
                  kernel = "radial", gamma = 2, cost = 1,
                  decision.values = T)
fitted <- attributes(
  predict(svmfit.opt, dat[train, ], decision.values = TRUE)
)$decision.values
# Training data and Test data ROC curves
```

```
par(mfrow = c(1, 2))
rocplot(-fitted, dat[train, "y"], main = "Training Data")
svmfit.flex <- svm(y ~ ., data = dat[train, ],
                   kernel = "radial", gamma = 50, cost = 1,
                   decision.values = T)
# Train data ROC curve
fitted <- attributes(
  predict(svmfit.flex, dat[train, ], decision.values = T)
)$decision.values
rocplot(-fitted, dat[train, "y"], add = T, col = "red")
# Test data ROC curves
fitted <- attributes(
  predict(svmfit.opt, dat[-train, ], decision.values = T)
)$decision.values
rocplot(-fitted, dat[-train, "y"], main = "Test Data")
fitted <- attributes(
  predict(svmfit.flex, dat[-train, ], decision.values = T)
)$decision.values
rocplot(-fitted, dat[-train, "y"], add = T, col = "red")
```

8.6.4 SVM with multiple classes

```
# Create data set
set.seed(1)
x <- rbind(x, matrix(rnorm(50 * 2), ncol = 2))
y <- c(y, rep(0, 50))
x[y == 0, 2] <- x[y == 0, 2] + 2
dat <- data.frame(x = x, y = as.factor(y))
par(mfrow = c(1, 1))
plot(x, col = (y + 1))
# Fitting radial kernel
svmfit <- svm(y ~ ., data = dat, kernel = "radial",
              cost = 1, gamma = 1)
plot(svmfit, dat)
summary(svmfit)
```

9 Unsupervised Learning Methods PCA and Clustering

An overview

We are not interested in prediction any more. Instead, our main goal is to discover interesting things about the measurements on $X_1, ..., X_p$.

- *PCA*: A tool used for data visualisation or data preprocessing before supervised techniques are applied.
- *Clustering*: A broad class of methods for discovering unknown subgroups in data.

The challenge of unsupervised learning

Unsupervised learning is often much more challenging than supervised learning.

- The exercise tends to be more subjective, and there is no simple goal for the analysis, such as the prediction of a response.
- Unsupervised learning is often performed as part of an *exploratory data analysis*.
- It can be hard to assess the results obtained from unsupervised learning methods, since there is no universally accepted mechanism for validating the results (like CV), because we simply do not know the true answer.

Examples of unsupervised learning applications

- A cancer researcher assesses gene expressions in 100 patients with breast cancer. They might look for subgroups among breast cancer samples, in order to obtain a better understanding of the disease.
- Online shopping site might try to identify groups of shoppers with similar browsing and purchase histories. Then the individual shopper can be preferentially shown the items in which they particular likely to be interested.
- A search engine might choose what search results to display to a particular individual based on the click histories of other individuals with similar search patterns.

9.1 Principal Component Analysis

Why principal components

If we want to visualise n observations with measurements on a set of p features, we could examine 2D scatter plots of the data but this gives us $\binom{p}{2} = p(p-1)/2$ scatter plots. Therefore, we would like a low-dimensional representation of the data that captures as much of the information as possible.

9.1.1 Interpretation for PCs

In general, the m-th PC Z_m is given by, for $m = 1, ..., p$:

$$Z_m = \phi_{1m}X_1 + \phi_{2m}X_2 + ... + \phi_{pm}X_p = \boldsymbol{X}\boldsymbol{\phi}_{\boldsymbol{m}}^{\top}$$

- Geometric interpretation for the first PC (Z_1). The *loading vector* $\boldsymbol{\phi_1}$ defines a direction in feature space along which the data vary the most. If we project the n data points $\boldsymbol{x_1}, .., \boldsymbol{x_n}$ onto this direction, the projected values are the *PC scores* $z_{11}, ..., z_{n1}$.
- The second PC Z_2 is the linear combination of $X_1, ..., X_p$ that has maximal variance out of all linear combinations that are uncorrelated with Z_1. i.e. we restrict the direction of $\boldsymbol{\phi_2}$ to be orthogonal (perpendicular) to the direction $\boldsymbol{\phi_1}$.

9.1.2 An example of PCA – USArrests

Setup:

- For each of the 50 states in the US, the dataset contains number of arrests per 100,000 residents for each of the three crimes: *Assault*, *Murder*, and *Rape*.
- We also record *UrbanPop* (% of the population in each state living in urban areas).
- The PC score vectors have length $n = 50$, and the PC loading vectors have length $p = 4$.
- PCA was performed after standardisation.
- The following graph represents the principal component scores and the loading vectors in a *single bi-plot* display.

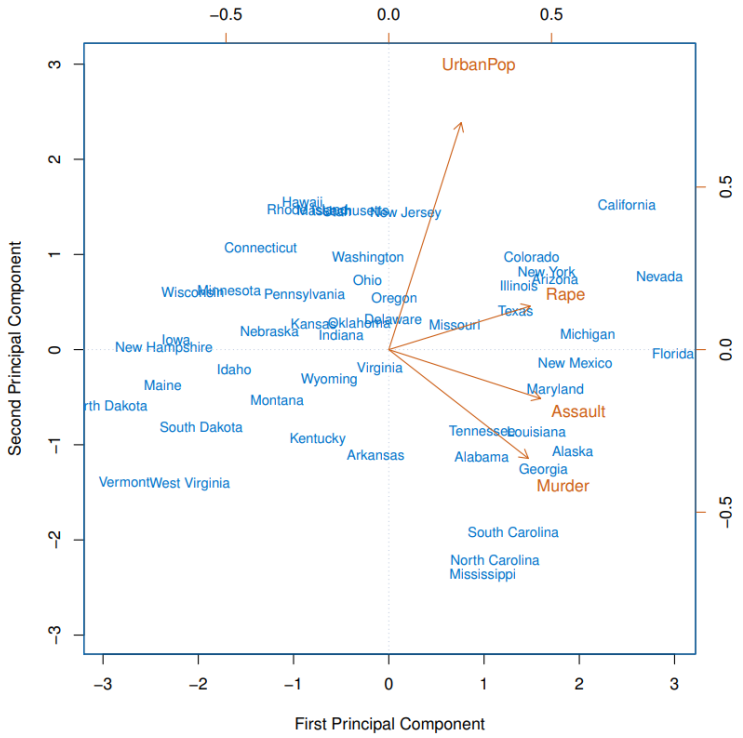


FIGURE 12.1. *The first two principal components for the* `USArrests` *data. The blue state names represent the scores for the first two principal components. The orange arrows indicate the first two principal component loading vectors (with axes on the top and right). For example, the loading for* `Rape` *on the first component is* 0.54*, and its loading on the second principal component* 0.17 *(the word* `Rape` *is centered at the point* $(0.54, 0.17)$*). This figure is known as a biplot, because it displays both the principal component scores and the principal component loadings.*

	PC1	PC2
`Murder`	0.5358995	−0.4181809
`Assault`	0.5831836	−0.1879856
`UrbanPop`	0.2781909	0.8728062
`Rape`	0.5434321	0.1673186

TABLE 12.1. *The principal component loading vectors,* ϕ_1 *and* ϕ_2*, for the* `USArrests` *data. These are also displayed in Figure 12.1.*

Main Findings:

- In the graph, we see that the first loading vector places approximately equal weight on *Assault*, *Murder*, and *Rape*, with must less weight on *UrbanPop*. Hence this component roughly corresponds to a measure of **overall rates of serious crime**.
- The second loading vector places most of its weight on *UrbanPop* and much less weight on the other three features. Hence, this component roughly corresponds to **the level of**

urbanisation of the state.

- Overall, the crime-related variables are located close to each other, and that the *UrbanPop* variable is far from the other three.
  - This indicates that the crime-related variables are correlated with each other- states with high murder rates tend to have high assault and rape rates - and that the *UrbanPop* variable is less correlated with the other three.

We can examine differences between the states via the two PC score vectors shown in the graphs.

- The loading vectors suggest that **states with large positive scores on the 1st PC**, such as California, Nevada and Florida, **have high crime rates**, while states like North Dakota, with negative scores on the 1st PC, have low crime rates.
- California also has **a high score on the second component, indicating a high level of urbanisation**, while the opposite is true for states like Mississippi.
- **States close to zero on both components**, such as Indiana, **have approximately average levels of both crime and urbanisation**. *Not a zero level.*

9.1.3 More on PCA (Scaling, Uniqueness, PVE, and number of PCs)

Scaling the variables

The variables are centered to have mean zero. Furthermore, the results from PCA depend on whether the variables have been individually scaled (as they are all multiped by a different constant).

E.g. in the example before the variables have different units: the crimes are the number of occurrences per 100,000 people, and *UrbanPop* is the percentage of the state's population that lives in that area. The four variables have variance *Murder*: 18.67, *Rape*: 87.73, *Assault*: 6945.16, and *UrbanPop*: 209.5.

Consequently, if we perform PCA on the unscalaed variables, then:

- $\boldsymbol{\phi_1}$ will have a very large loading for *Assault*;
- $\boldsymbol{\phi_2}$ will have a very large loading for *UrbanPop*.

We typically scale each variable to have s.d. of 1 before we perform PCA. In certain settings, if the variables are the same units, then we may not wish to scale the variables.

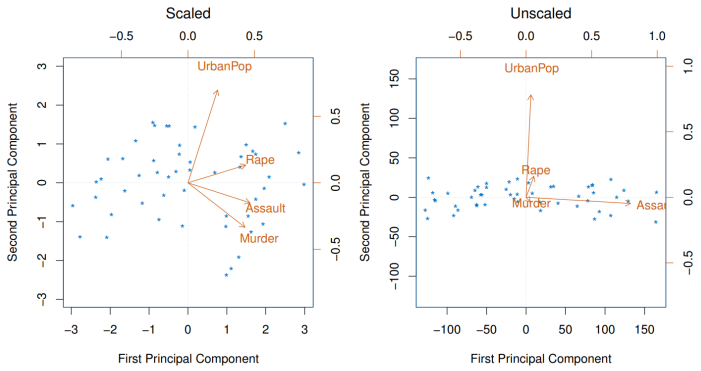


FIGURE 12.4. *Two principal component biplots for the* `USArrests` *data.* Left: *the same as Figure 12.1, with the variables scaled to have unit standard deviations.* Right: *principal components using unscaled data.* `Assault` *has by far the largest loading on the first principal component because it has the highest variance among the four variables. In general, scaling the variables to have standard deviation one is recommended.*

Uniqueness of the PCs

Each PC loading vector in unique, *up to a sign flip*. Meaning you can get the same PC loading vector, although the sign of those loading vectors may differ.

The signs may differ because each PC loading vector specifies a direction in p-dimensional space: flipping the sign has no effect as the direction does not change.

Similarly, the score vector are unique up to a sign flip. Since the variance of Z is the same as the variance of $-Z$.

Proportion of Variance Explained (PVE)

We are usually interested in PVE by each PC. The total variance present in the data (assuming all variables have mean centered at zero) is defined as:

$$\sum_{j=1}^{p} \text{Var}(X_j) = \sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2$$

and the variance explained by Z_m is:

$$\frac{1}{n} \sum_{i=1}^{n} z_{im}^2 = \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{jm} x_{ij} \right)^2$$

Thus, the PVE of Z_m is given by:

$$\frac{\frac{1}{n} \sum_{i=1}^{n} \left(\sum_{j=1}^{p} \phi_{jm} x_{ij} \right)^2}{\sum_{j=1}^{p} \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2}$$

Deciding the number of PCs to use

We would like ot use the **smallest** number of principal components required to get a **good** understanding of the data. There is no best number of PCs to use.

- An *ad hoc* approach: examine a *scree plot* of the data and look for an *elbow* in the scree plot.

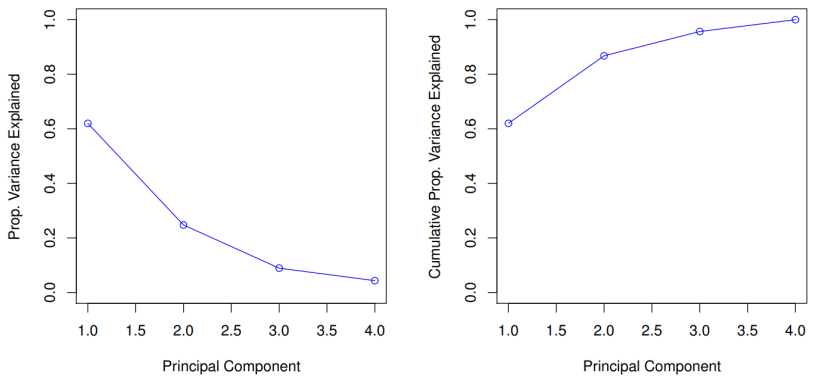


FIGURE 12.3. Left: *a scree plot depicting the proportion of variance explained by each of the four principal components in the* `USArrests` *data.* Right: *the cumulative proportion of variance explained by the four principal components in the* `USArrests` *data.*

- A *subjective approach*: look at first few PCs in order to find interesting patterns in the data. If no interesting patterns in first few PCs, then further PCs are unlikely to be on interest. Conversely, if the first few PCs are interesting, then we typically continue to look at subsequent PCs until no further interesting patterns are found.
- A *objective approach*: if we compute PCs for use in a supervised analysis, such as PCR, we can treat the number of PC score vectors to be used in the regression as a tuning parameter to be selected via cross-validation or a related approach.

9.1.4 PCA in R

```
states=row.names(USArrests)
names(USArrests)
apply(USArrests, 2, mean) # 2 for columns
apply(USArrests, 2, var) # 2 for columns
pr.out=prcomp(USArrests, scale=TRUE) # PCA with standardisation
names(pr.out)
pr.out$center # mean of original columns
pr.out$scale # s.d. of original columns
pr.out$rotation # PC loading vectors
dim(pr.out$x) # PC scores, i.e. transformed features
biplot(pr.out, scale=0)
pr.out$rotation=-pr.out$rotation # to be consistent with textbook figure
pr.out$x=-pr.out$x
biplot(pr.out, scale=0)
pr.out$sdev # s.d. of PCs
pr.var=pr.out$sdev^2 # variance of PCs
pr.var
pve=pr.var/sum(pr.var) # calculating PVE (proportion of var explained)
pve # first 2 PCs are good enough because explain greater than 80% of the variance
plot(pve, xlab="Principal Component", ylab="Proportion of Variance Explained",
     ylim=c(0,1),type="b") # elbow point is about 2 so good enough
plot(cumsum(pve), xlab="Principal Component",
    ylab="Cumulative Proportion of Variance Explained",
    ylim=c(0,1),type="b") # cumsum() calculates cumulative sum of elements
```

9.2 Clustering

We seek to partition the observations into distinct groups so that the observations within each group are homogeneous, while observations between groups are heterogeneous.

Clustering approaches:

- **K-means clustering**: partition the observations into a pre-specified number of clusters;
- **Hierarchical clustering**: generate a dendrogram to visually represent the observations. Allows us to view all the clusters obtained.

Two goals of clustering:

1. Cluster observations on the basis of the features in order to identify *subgroups among the observations*.
2. Cluster features on the basis of the observation in order to discover *subgroups among the features*.

9.2.1 K-means clustering

Must first specify number of clusters K; then the K-means algorithm will assign each observation to exactly one of the K clusters.

Let $C_1, ..., C_k$ denote sets containing the indices of the observation in each cluster. These sets satisfy two properties:

1. $C_1 \cup C_2 \cup ... \cup C_k = \{1, ..., n\}$. In other words, each observation belongs to at least one of the K clusters.
2. $C_k \cup C_{k'} = \varnothing \; \forall k \neq k'$. The clusters are non overlapping: no observations belongs to more than one cluster.

Idea behind K-means clustering

A good clustering is one for which the within-cluster variation ($W(C_k)$ for cluster C_k) is as small as possible.

$$W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2 = 2 \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2$$

. Where $|C_k|$ denotes the number of observations in C_k, and $\bar{x}_{kj} = \frac{1}{|C_k|} \sum_{i \in C_k} x_{ij}$.

hence we want to solve the problem:

$$\min_{C_1,...,C_k} \sum_{k=1}^{K} W(C_k)$$

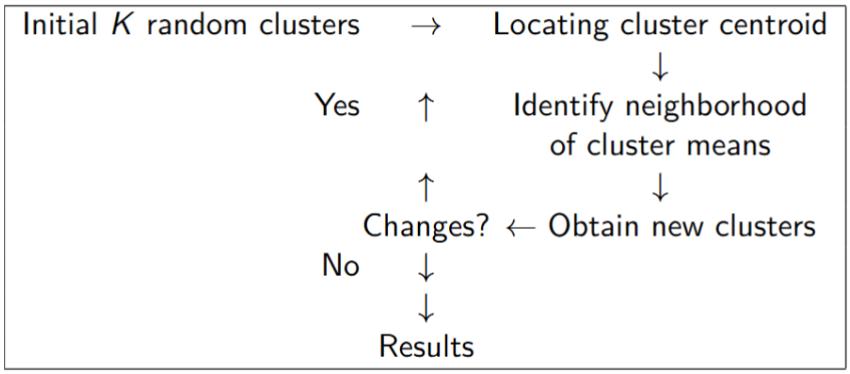
We want to partition the observations into K clusters such that the total within-cluster variation, summed over all K clusters, is as small as possible. We can combine the above into one minimisation problem:

$$\min_{C_1,...,C_k} \sum_{k=1}^{K} \sum_{i \in C_k} \sum_{j=1}^{p} (x_{ij} - \bar{x}_{kj})^2$$

9.2.2 K-Means clustering algorithm

1. Randomly assign a number, from 1 to K, to each of the observations. These serve as initial cluster assignments for the observations, denotes $C_1^{(1)}, ..., C_K^{(1)}$. Let $I = 1$.
2.
   (a) For $C_k^{(I)}$, $k = 1, ..., K$, compute the cluster *centroid* $\left(\bar{x}_{k1}^{(I)}, ..., \bar{x}_{kp}^{(I)}\right)$. i.e. the vector of the p feature means for the observations in $C_k^{(I)}$.
   (b) Assign each observation to the cluster whose centroid is closest (where teh closest is defined using Euclidean distance) to generate new clusters $C_1^{(I+1)}, ..., C_K^{(I+1)}$. If $C_1^{(I+1)}, ..., C_K^{(I+1)}$ are different from $C_1^{(1)}, ..., C_K^{(1)}$, then let $I = I + 1$ and repeat Step 2; otherwise, return $C_1^{(1)}, ..., C_K^{(1)}$ as the output.

The algorithm can be summarised in the following flow chart:



- Step 2(a) minimises the sum-of-squared deviations by updating the cluster means, $\bar{x}_{kj},\ j = 1, ..., p$.
- Step 2(b) minimises the sum-of-squared deviations by reallocating the observations, i.e. updating $\{i \in C_k\},\ k = 1, ..., K$.
- As we loop the algorithm, the clustering obtained will continually improve until the result no longer changes; the objective (minimisation) function will never increase.
- When the result no longer changes, a local optimum has been reached.
- The algorithm finds a local rather than a global optimum. So it depends on the initial (random) cluster obtained in step 1 of the algorithm. So its is important to run the algorithm multiple times from different random initial configurations. Then one selects the best solution, i.e. that for which the objective is the smallest.

K-means clustering algorithm visual example:

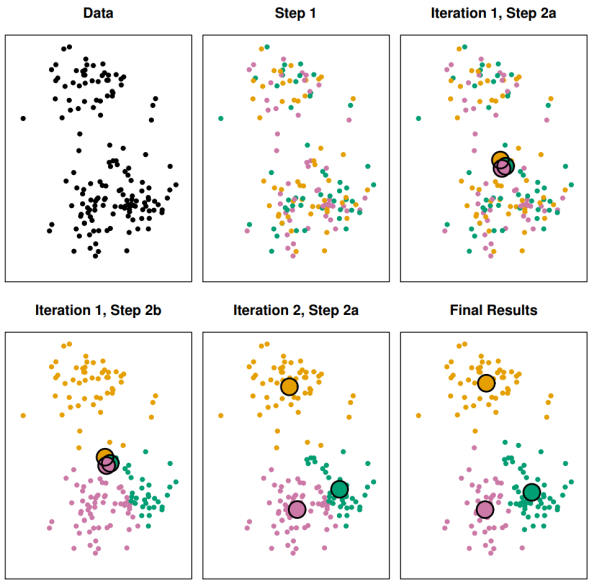


FIGURE 12.8. *The progress of the K-means algorithm on the example of Figure 12.7 with K=3.* Top left: *the observations are shown.* Top center: *in Step 1 of the algorithm, each observation is randomly assigned to a cluster.* Top right: *in Step 2(a), the cluster centroids are computed. These are shown as large colored disks. Initially the centroids are almost completely overlapping because the initial cluster assignments were chosen at random.* Bottom left: *in Step 2(b), each observation is assigned to the nearest centroid.* Bottom center: *Step 2(a) is once again performed, leading to new cluster centroids.* Bottom right: *the results obtained after ten iterations.*

K-means clustering different initial configurations visual example:

FIGURE 12.9. *K-means clustering performed six times on the data from Figure 12.7 with $K = 3$, each time with a different random assignment of the observations in Step 1 of the K-means algorithm. Above each plot is the value of the objective (12.17). Three different local optima were obtained, one of which resulted in a smaller value of the objective and provides better separation between the clusters. Those labeled in red all achieved the same best solution, with an objective value of 235.8.*

9.2.3 K-means clustering in R

```
set.seed(2)
x=matrix(rnorm(50*2), ncol=2)
x
plot(x)
x[1:25,1]=x[1:25,1]+3
x[1:25,2]=x[1:25,2]-4
plot(x)
km.out=kmeans(x,2,nstart=20) #20 random sets were chosen and report the best one
km.out$cluster
plot(x, col=(km.out$cluster+1), main="K-Means Clustering Results with K=2",
     xlab="", ylab="", pch=20, cex=2)

#Try different random sets
set.seed(4)
for(i in 1:4){
km.out=kmeans(x,3,nstart=i)
print(km.out$tot.withinss)
plot(x, col=(km.out$cluster+1), main="K-Means Clustering Results with K=3",
     xlab="", ylab="", pch=20, cex=2)
}

km.out=kmeans(x,3,nstart=20)
km.out$tot.withinss # total within cluster sum of squares
plot(x, col=(km.out$cluster+1), main="K-Means Clustering Results with K=3",
     xlab="", ylab="", pch=20, cex=2)
```

9.2.4 Hierarchical clustering

- Disadvantage of K-means clustering is that you have to pre-specify the number of clusters K. *Hierarchical clustering does not commit to a particular choice of* K.

- Hierarchical clustering has an advantage over K-means is that is results in a tree based representation of the observations, a *dendrogram*.
- *Bottom-up* or *agglomerative clustering* is the most common type of hierarchical clustering, and refers to the fact that the dendrogram is built starting from the leaves and combining clusters up to the trunk.

The clusters depend on what level you make the cut at in the dendrogram:

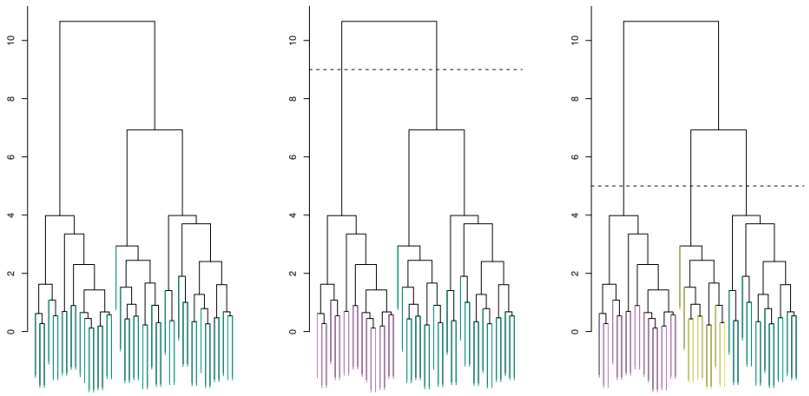


FIGURE 12.11. Left: *dendrogram obtained from hierarchically clustering the data from Figure 12.10 with complete linkage and Euclidean distance.* Center: *the dendrogram from the left-hand panel, cut at a height of nine (indicated by the dashed line). This cut results in two distinct clusters, shown in different colors.* Right: *the dendrogram from the left-hand panel, now cut at a height of five. This cut results in three distinct clusters, shown in different colors. Note that the colors were not used in clustering, but are simply used for display purposes in this figure.*

A visual example of how to interpret a dendrogram based on euclidean distance in a two-dimensional space:

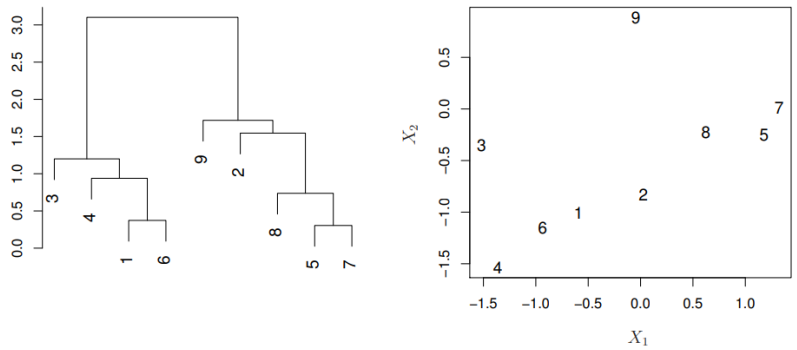


FIGURE 12.12. *An illustration of how to properly interpret a dendrogram with nine observations in two-dimensional space.* Left: *a dendrogram generated using Euclidean distance and complete linkage. Observations* 5 *and* 7 *are quite similar to each other, as are observations* 1 *and* 6. *However, observation* 9 *is* no more similar to *observation* 2 *than it is to observations* 8, 5, *and* 7, *even though observations* 9 *and* 2 *are close together in terms of horizontal distance. This is because observations* 2, 8, 5, *and* 7 *all fuse with observation* 9 *at the same height, approximately* 1.8. Right: *the raw data used to generate the dendrogram can be used to confirm that indeed, observation* 9 *is no more similar to observation* 2 *than it is to observations* 8, 5, *and* 7.

How to obtain clusters using a dendrogram

We make a horizontal cut across the dendrogram, and the distinct sets of observations beneath the cut can be interpreted as clusters. The diagram above shows how 1, 2, or 3 clusters are made in a dendrogram with different levels of cuts.

What hierarchical refers to

The clusters obtained by cutting the dendrogam at a given height are necessarily nested within the clusters obtained by cutting the dendrogram at any greater height.

Assumption of hierarchical is unrealistic

e.g. there is a group of people with a $50 - 50$ split of males and females, evenly split among Americans, Japanese, and French. In this case, the true clusters are not nested.

Due to situations like this, hierarchical clustering can sometimes yield worse (i.e. less accurate) results than K-means clustering for a given number of clusters.

9.2.5 Hierarchical clustering algorithm

1. Being with n observations and a measure (such as Euclidean distance) of all the $\binom{n}{2} = n(n-1)/2$ pairwise dissimilarities. Treat each observation as its own cluster.
2. For $i = n, n-1, ..., 2$:
   (a) Examine all pairwise inter-cluster dissimilarities among the i clusters and identify the pair of clusters that are least dissimilar (most similar). Fuse these two clusters. The dissimilarity between these two clusters indicate the height in the dendrogram at which the fusion should be placed.
   (b) Compute the new pairwise inter-cluster dissimilarities among the $i - 1$ remaining clusters.

An illustrating example of a dendrogram continued:

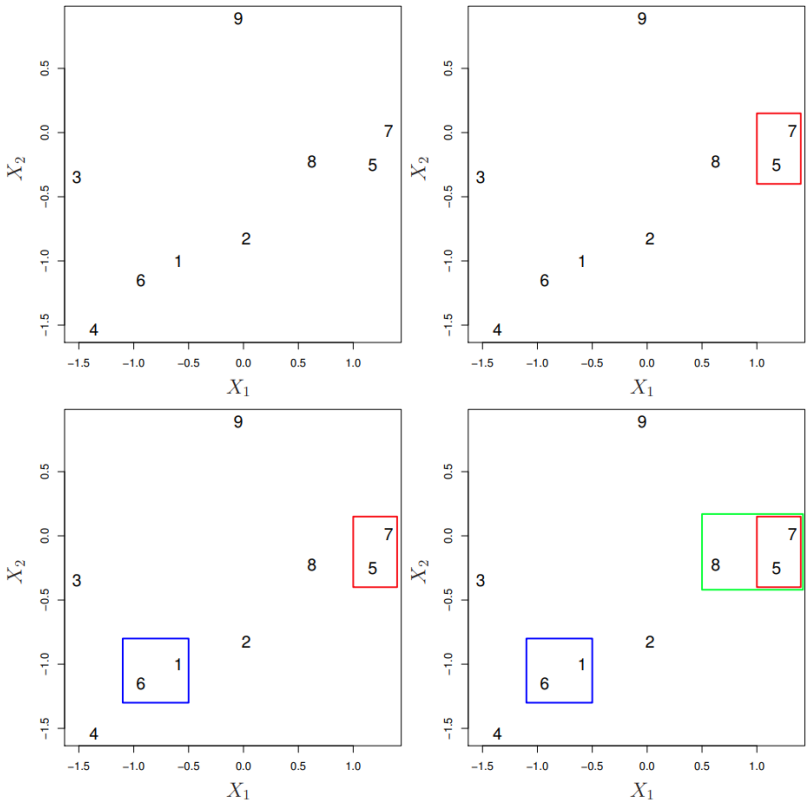


FIGURE 12.13. *An illustration of the first few steps of the hierarchical clustering algorithm, using the data from Figure 12.12, with complete linkage and Euclidean distance.* Top Left: *initially, there are nine distinct clusters,* $\{1\}, \{2\}, \ldots, \{9\}$. Top Right: *the two clusters that are closest together,* $\{5\}$ *and* $\{7\}$*, are fused into a single cluster.* Bottom Left: *the two clusters that are closest together,* $\{6\}$ *and* $\{1\}$*, are fused into a single cluster.* Bottom Right: *the two clusters that are closest together using* complete linkage, $\{8\}$ *and the cluster* $\{5, 7\}$*, are fused into a single cluster.*

Dissimilarity between two clusters

One issue with the algorithm. How do we define the dissimilarity between two clusters if one or both of the clusters contains multiple observations? The concept of dissimilarity between a pair of observations needs to be extended to a pair of groups of observations.

This extension is achieved by developing the notion of *linkage*, which defines the dissimilarity between two groups of observations.

Most common used types of linkage

Centroid is not assessable

Linkage	*Description*
Complete	Maximal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *largest* of these dissimilarities.
Single	Minimal intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *smallest* of these dissimilarities. Single linkage can result in extended, trailing clusters in which single observations are fused one-at-a-time.
Average	Mean intercluster dissimilarity. Compute all pairwise dissimilarities between the observations in cluster A and the observations in cluster B, and record the *average* of these dissimilarities.
Centroid	Dissimilarity between the centroid for cluster A (a mean vector of length p) and the centroid for cluster B. Centroid linkage can result in undesirable *inversions*.

TABLE 12.3. *A summary of the four most commonly-used types of linkage in hierarchical clustering.*

Clustering by different types of linkage

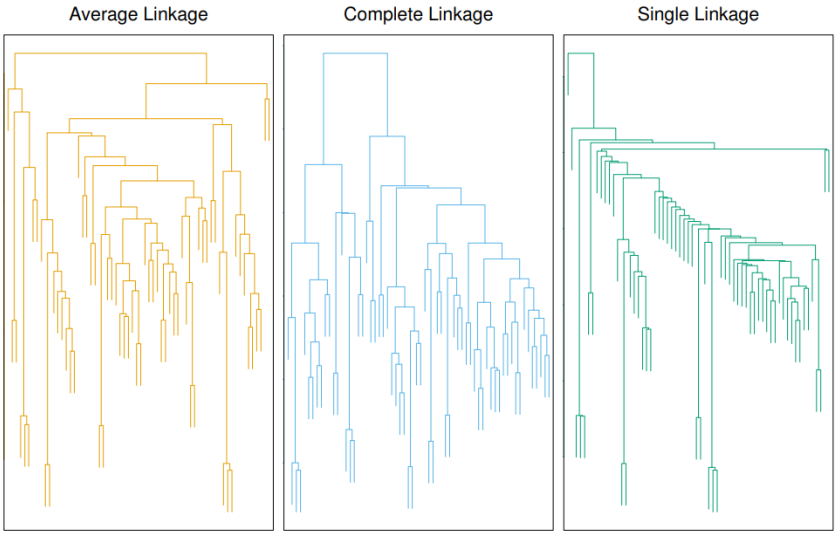


FIGURE 12.14. *Average, complete, and single linkage applied to an example data set. Average and complete linkage tend to yield more balanced clusters.*

Choice of dissimilarity measure

Apart from *Euclidean* distance as the measure, we might prefer other measurer.

Correlation-based distance considers two observations to be similar if their features are highly correlated, even though the observed values may be far apart in terms of Euclidean distance. *correlation-based distance focuses on the shapes of observation profiles rather than their magnitudes.*

The choice of the dissimilarity measure has a strong effect on the resulting dendrogram. In general, *careful attention should be paid to the type of data being clustered and the scientific question at hand.* The considerations should determine what type of measure is suitable.

Another example - choosing dissimilarity measure and scaling

An online retailer interested in clustering shoppers based on their histories. The goal is to identify subgroups of similar shoppers, so that shoppers within each subgroup can be shown items and advertisements that are particular likely to interest them.

Assume the data takes the form of a matrix where the rows are the shoppers and the columns are the items available for purchase; the elements of the data matrix indicate the number of times a given shopper has purchased a given item (i.e. 0 if they have never purchased it, 1 if the shopper purchased it once, etc.)

Q1: what type of dissimilarity measure should be used to cluster the shoppers?

- If Euclidean distance is used, then the shoppers who have bought very few items overall (i.e. infrequent users) will be clustered together. This may not be desirable.
- If correlation-based distance is used, then shoppers with similar preferences will be clustered together, even is some shoppers with these preferences are higher-volume shoppers than others.
- Therefore, for this application, correlation-based distance may be a better choice.

Q2: whether or not the variables should be scaled to have s.d. of one before the dissimilarity between the observations is computed?

- Some items may be purchased more frequently than others (e.g. sock to a computer).
- High-frequently purchases like socks therefore tend to have a much larger effect on the inter-shopper dissimilarities, and hence on the clustering ultimately obtained, than a rare purchase like computers. This may not be desirable.
- If the variables are scaled to have unit s.d. before the inter-observation dissimilarities are computed, then each variable will in effect be given equal importance in the hierarchical clustering performed.
- We might also want to scale the variables to have one s.d. if they are measured on different scales (e.g. centimeters vs kilometers).
- *The answer to the question depends on the application at hand.*

9.2.6 Hierarchical clustering in R

```
hc.complete=hclust(dist(x), method="complete")
hc.average=hclust(dist(x), method="average")
hc.single=hclust(dist(x), method="single")
par(mfrow=c(1,3))
plot(hc.complete,main="Complete Linkage", xlab="", sub="", cex=.9)
plot(hc.average, main="Average Linkage", xlab="", sub="", cex=.9)
plot(hc.single, main="Single Linkage", xlab="", sub="", cex=.9)
cutree(hc.complete, 2)
cutree(hc.average, 2)
cutree(hc.single, 2)
cutree(hc.single, 4)

xsc=scale(x) #Scaling the variables beforehand

plot(hclust(dist(xsc), method="complete"),
     main="Hierarchical Clustering with Scaled Features")

x=matrix(rnorm(30*3), ncol=3)
dd=as.dist(1-cor(t(x))) # use 1-correlation as the distance function
plot(hclust(dd, method="complete"),
```

```
     main="Complete Linkage with Correlation-Based Distance", xlab="", sub="")

x[,1]=x[,1]*2
x[,3]=x[,3]+5
plot(hclust(dd, method="complete"),
     main="Complete Linkage with Correlation-Based Distance", xlab="", sub="")
# So scales have no impact if we use correlation type of distance function.

plot(hclust(dist(x), method="complete"),
     main="Complete Linkage with U Distance", xlab="", sub="")
```

9.2.7 Practical issues in clustering

1. *Small Decisions with Big Consequences*: Each of the following decisions can have a strong impact on the results obtained. In practice, we try several choices and choose the one with the most useful solution.
   - Should the observations or features first be standardised in some way?
   - In the case of hierarchical clustering,
     - What dissimilarity measure should be used?
     - What type of linkage should be used?
     - Where should we cut the dendrogam in order to obtain clusters?
   - In the case of K-means clustering, how many clusters should we look for in the data?
2. *Validating the Clusters Obtained*: Any time clustering is performed we will find clusters. But we really want to know whether the clusters that have been found represent true subgroups in the data, or whether they are simply a result of *clustering the noise*.
   - There exist a number of techniques for assignment a p-value to a cluster in order to assess whether there is more evidence for the cluster than one would expect due to change. However, there is no consensus on a single best approach.
3. *Other Considerations in Clustering*: both K-means and hierarchical clustering will assign each observation to a cluster. However, sometimes this might not be appropriate due to the existence of outliers. Mixture models are an attractive approach for accommodating the presence of such outliers (models not need to be known for this subject).

9.2.8 Unsupervised learning methods example (NCI60 data) in R

```
library(ISLR)
nci.labs=NCI60$labs
nci.data=NCI60$data
dim(nci.data)
#View(nci.data)
nci.labs[1:4]
table(nci.labs)

# PCA on the NCI60 Data

pr.out=prcomp(nci.data, scale=TRUE)

#The function will be used to assign a color to each of the 64 cell lines,
# based on the cancer type to which it corresponds.

Cols=function(vec){
  cols=rainbow(length(unique(vec)))
  return(cols[as.numeric(as.factor(vec))])
}
```

```
par(mfrow=c(1,2))
plot(pr.out$x[,1:2], col=Cols(nci.labs), pch=19,xlab="Z1",ylab="Z2")
plot(pr.out$x[,c(1,3)], col=Cols(nci.labs), pch=19,xlab="Z1",ylab="Z3")
summary(pr.out)
plot(pr.out)

# plot the PVE and cumulative PVE

pve=100*pr.out$sdev^2/sum(pr.out$sdev^2)
par(mfrow=c(1,2))
plot(pve,  type="o", ylab="PVE", xlab="Principal Component", col="blue")
plot(cumsum(pve), type="o", ylab="Cumulative PVE", xlab="Principal Component", col="brown3")

# Clustering the Observations of the NCI60 Data

sd.data=scale(nci.data)
par(mfrow=c(3,1))
data.dist=dist(sd.data)
plot(hclust(data.dist), labels=nci.labs, main="Complete Linkage",
     xlab="", sub="",ylab="")
plot(hclust(data.dist, method="average"), labels=nci.labs,
     main="Average Linkage", xlab="", sub="",ylab="")
plot(hclust(data.dist, method="single"), labels=nci.labs,
     main="Single Linkage", xlab="", sub="",ylab="")

par(mfrow=c(1,1))
plot(hclust(data.dist, method="single"), labels=nci.labs,
     main="Single Linkage", xlab="", sub="",ylab="")

hc.out=hclust(dist(sd.data)) #default is Complete linkage
hc.clusters=cutree(hc.out,4)
table(hc.clusters,nci.labs)
par(mfrow=c(1,1))
plot(hc.out, labels=nci.labs)
abline(h=139, col="red")
hc.out

#Try the K-means clustering with K=4

set.seed(2)
km.out=kmeans(sd.data, 4, nstart=20)
km.clusters=km.out$cluster
table(km.clusters,hc.clusters)

#Hclusering on PCs

hc.out=hclust(dist(pr.out$x[,1:5]))
plot(hc.out, labels=nci.labs, main="Hier. Clust. on First Five Score Vectors")
table(cutree(hc.out,4), nci.labs)
```

10 Textbook Questions Solutions

10.1 Linear Regression - Chpt 3

10.1.1 Question 8 - simple linear regression model analysis

Part a

Use the lm() function to perform a simple linear regression with mpg as the response and horsepower as the predictor. Use the summary() function to print the results. Comment on the output. For example:

```
library(ISLR)
Auto <- na.omit(Auto)
attach(Auto)
lm.fit <- lm(mpg ~ horsepower, data=Auto)
summary(lm.fit)
```

Part i

Is there a relationship between the predictor and the response?
Yes, the F-statistic p-value is close to zero so we can reject the null hypothesis and state there is a statistically significant relationship between *horsepower* and *mpg*.

Part ii

How strong is the relationship between the predictor and the response?
Since the p-value is extremely small, the relationship is very strong.

Part iii

Is the relationship between the predictor and the response positive or negative?
$\hat{\beta}_1 = -0.15$, so the relationship is negative.

Part iv

What is the predicted mpg associated with a horsepower of 98? What are the associated 95% confidence and prediction intervals?

```
predict(lm.fit, data.frame(horsepower=98), interval="confidence")
predict(lm.fit, data.frame(horsepower=98), interval="prediction)
```

Part b

Plot the response and the predictor. Use the abline() function to display the least squares regression line.

```
plot(horsepower, mpg)
abline(lm.fit)
```

Part c

Use the plot() function to produce diagnostic plots of the least squares regression fit. Comment on any problems you see with the fit.

```
par(mfrow=c(2,2))
plot(lm.fit)
```

Based on the residuals plots, there is some evidence of non-linearity.

10.1.2 Question 9 - multiple linear regression on dataset

Part a

Produce a scatter plot matrix which includes all of the variables in the data set.

```
pairs(Auto)
```

Part b

Compute the matrix of correlations between the variables using the function cor(). You will need to exclude the name variable, which is qualitative.

```
cor(subset(Auto, select=-name))
```

Part c

Use the lm() function to perform a multiple linear regression with mpg as the response and all other variables except name as the predictors. Use the summary() function to print the results. Comment on the output. For instance:

```
lm.fit1 <- lm(mpg ~ . - name, data=Auto)
summary(lm.fit1)
```

Part i

Is there a relationship between the predictors and the response?
Yes, F-statistic p-value is basically 0, indicated evidence against the null-hypothesis. So there is a relationship.

Part ii

Which predictors appear to have a statistically significant relationship to the response?
Looking at the p-values of each predictors t-statistic. Displacement, weight, year, and origin have statistically significant relationship.

Part iii

What does the coefficient for the year variable suggest?
Coefficient is 0.75, suggesting that cars with later model years have higher *mpg*.

Part d

Use the plot() function to produce diagnostic plots of the linear regression fit. Comment on any problems you see with the fit. Do the residual plots suggest any unusually large outliers? Does the leverage plot identify any observations with unusually high leverage?

```
par(mfrow=c(2,2))
plot(lm.fit1)
```

Based on the diagnostic plots one can see that the data is clearly non-linear. There is also a data point (the 14th) that has high leverage.

Part e

Use the $*$ and : symbols to fit linear regression models with interaction effects. Do any interactions appear to be statistically significant?

```
lm.fit2 <- lm(mpg~displacement+weight+year+origin, data=Auto)
lm.fit3 <- lm(mpg~displacement+weight+year+origin+year:origin, data=Auto)
lm.fit4 <- lm(mpg~displacement+origin+year*weight, data=Auto)
lm.fit5 <- lm(mpg~year+origin+displacement*weight, data=Auto)
summary(lm.fit2) # could get summary of rest of the models as well
```

All interaction terms seem to be statistically significant

Part f

Try a few different transformations of the variables, such as $\log(X), \sqrt{X}, X^2$. *Comment on your findings.*

```
lm.fit6 <- lm(mpg~poly(displacement,3)+weight+year+origin, data=Auto)
lm.fit7 <- lm(mpg~displacement+I(log(weight))+year+origin, data=Auto)
lm.fit8 <- lm(mpg~displacement+I(weight^2)+year+origin, data=Auto)
summary(lm.fit6) # could get summary of the rest of the models as well
```

$\log(weight)$, $\sqrt{horsepower}$, and $acceleration^2$ all have statistical significance of some sort. The residuals plot has less of a discernible pattern than the plot of all linear regression terms. The studentized residuals displays potential outliers (>3). The leverage plot indicates more than three points with high leverage.

However, 2 problems are observed from the above plots: 1) the residuals vs fitted plot indicates heteroskedasticity (unconstant variance over mean) in the model. 2) The Q-Q plot indicates somewhat unnormality of the residuals.

So, a better transformation need to be applied to our model. From the correlation matrix in 9a., displacement, horsepower and weight show a similar nonlinear pattern against our response mpg. This nonlinear pattern is very close to a log form. So in the next attempt, we use log(mpg) as our response variable.

The outputs show that log transform of mpg yield better model fitting (better R^2, normality of residuals).

10.2 Classification Methods - Chpt 4

10.2.1 Question 5 – Comparing LDA and QDA

We now examine the differences between LDA and QDA.

Part (a)

If the Bayes decision boundary is linear, do we expect LDA or QDA to perform better on the training set? On the test set?
Answer: On the training set, QDA will typically have the lower error since its greater flexibility can fit even a linear boundary at least as well (and often overfit slightly). On the test set, LDA will usually perform better because QDA's extra parameters introduce higher variance without reducing bias when the true boundary is linear.

Part (b)

If the Bayes decision boundary is non-linear, do we expect LDA or QDA to perform better on the training set? On the test set?
Answer: In both cases—training and test—QDA will generally outperform LDA, because its quadratic boundary can more accurately approximate the true non-linear Bayes boundary (assuming sufficient data).

Part (c)

In general, as the sample size n increases, do we expect the test prediction accuracy of QDA relative to LDA to improve, decline, or be unchanged? Why?
Answer: As n grows, QDA's variance decreases (more data stabilizes its many covariance estimates), so its test accuracy improves relative to LDA. In the limit of large n, QDA can never do worse than LDA and will often do better when the true boundary is somewhat non-linear.

Part (d)

True or False: Even if the Bayes decision boundary for a given problem is linear, we will probably achieve a superior test error rate using QDA rather than LDA because QDA is flexible enough to model a linear decision boundary. Justify your answer.
Answer: False. When the true boundary is linear, LDA's lower variance (fewer parameters) generally yields better test performance than QDA's unnecessary flexibility, which only increases variance without reducing bias.

10.2.2 Question 13 – Weekly Data Set Classification

This question should be answered using the `Weekly` *data set from the* `ISLR2` *package. It contains 1,089 weekly returns for 21 years (1990–2010).*

Part (a)

Produce some numerical and graphical summaries of the `Weekly` *data. Do there appear to be any patterns?*

```
library(ISLR2)
data(Weekly)

# Numerical summary
summary(Weekly)

# Pairwise scatterplots
pairs(Weekly)
```

Part (b)

Use the full data set to perform a logistic regression with `Direction` *as the response and the five lag variables plus* `Volume` *as predictors. Which predictors are statistically significant?*

```
fit.logit <- glm(Direction ~ Lag1 + Lag2 + Lag3 +
                         Lag4 + Lag5 + Volume,
                 data = Weekly, family = binomial)
summary(fit.logit)
```

In the output, only `Lag2` has a p–value below 0.05, indicating it is the sole significant predictor.

Part (c)

Compute the confusion matrix and overall fraction of correct predictions on the full data. What types of mistakes does the logistic model make?

```
# Predicted probabilities and classes
probs <- predict(fit.logit, type = "response")
preds <- ifelse(probs > 0.5, "Up", "Down")

# Confusion matrix and accuracy
tab.logit <- table(Predicted = preds, Actual = Weekly$Direction)
tab.logit
mean(preds == Weekly$Direction)
```

The model has high sensitivity (correctly identifies `Up` weeks) but a low specificity (many `Down` weeks are misclassified as `Up`).

Part (d)

Fit a logistic regression using only `Lag2` *as predictor, training on years 1990–2008 and testing on 2009–2010. Report the confusion matrix and test accuracy.*

```
# Split into training (19902008) and test (20092010)
train_idx <- Weekly$Year <= 2008
train <- Weekly[train_idx, ]
test  <- Weekly[!train_idx, ]

# Fit and predict
fit2 <- glm(Direction ~ Lag2, data = train, family = binomial)
probs2 <- predict(fit2, newdata = test, type = "response")
pred2  <- ifelse(probs2 > 0.5, "Up", "Down")

# Confusion matrix and accuracy
tab2 <- table(Predicted = pred2, Actual = test$Direction)
tab2
mean(pred2 == test$Direction)
```

Part (e)

Repeat part (d) using LDA.

```
library(MASS)
fit.lda <- lda(Direction ~ Lag2, data = train)
pred.lda <- predict(fit.lda, newdata = test)$class
table(Predicted = pred.lda, Actual = test$Direction)
mean(pred.lda == test$Direction)
```

Part (f)

Repeat part (d) using QDA.

```
fit.qda <- qda(Direction ~ Lag2, data = train)
pred.qda <- predict(fit.qda, newdata = test)$class
table(Predicted = pred.qda, Actual = test$Direction)
mean(pred.qda == test$Direction)
```

Part (g)

Repeat part (d) using KNN with $K = 1$.

```
library(class)
train.X <- as.matrix(train$Lag2)
test.X  <- as.matrix(test$Lag2)
knn1    <- knn(train.X, test.X, train$Direction, k = 1)
table(Predicted = knn1, Actual = test$Direction)
mean(knn1 == test$Direction)
```

Part (h)

Repeat part (d) using naive Bayes.

```
library(e1071)
fit.nb  <- naiveBayes(Direction ~ Lag2, data = train)
pred.nb <- predict(fit.nb, newdata = test)
table(Predicted = pred.nb, Actual = test$Direction)
mean(pred.nb == test$Direction)
```

Part (i)

Which method provides the best test-set performance? Comparing test accuracies, logistic regression (with `Lag2`) and LDA achieve the highest accuracy, outperforming QDA, KNN and naive Bayes.

Part (j)

Experiment with different combinations of predictors (including transformations and interactions) and values of K for KNN. Which combination yields the best test performance?

```
# Example: KNN with k = 20
knn20 <- knn(train.X, test.X, train$Direction, k = 20)
mean(knn20 == test$Direction)

# Example: logistic with a Lag1^2 term
fit2b <- glm(Direction ~ Lag2 + I(Lag1^2), data = train, family = binomial)
pred2b <- predict(fit2b, newdata = test, type = "response")
pred2b <- ifelse(pred2b > 0.5, "Up", "Down")
mean(pred2b == test$Direction)
```

In these experiments, KNN with $K \approx 20$ and the logistic model including $I(\text{Lag1}^2)$ both slightly improved accuracy over the basic models.

10.2.3 Question 16 – Classifying High vs. Low Crime Tracts in `Boston`

Using the `Boston` *data set, we wish to predict whether a census tract has a crime rate* `crim` *above (1) or below (0) the median. We will compare logistic regression, LDA, QDA, naive Bayes, and KNN, using two different subsets of predictors.*

Data preparation (common to all parts)

```
library(MASS)           # for Boston, lda, qda
library(class)          # for knn
library(e1071)          # for naiveBayes

# Create binary response crim01 and split 70/30
```

```r
data(Boston)
crim01 <- ifelse(Boston$crim > median(Boston$crim), 1, 0)
df     <- data.frame(Boston, crim01)
set.seed(1)
n      <- nrow(df)
train  <- df[sample(n, 0.7*n), ]
test   <- df[-as.numeric(rownames(train)), ]

# Predictor subsets
X1.train <- as.matrix(train[, c("age","dis","lstat","medv")])
X1.test  <- as.matrix(test[,  c("age","dis","lstat","medv")])
X2.train <- as.matrix(train[, c("tax","rad")])
X2.test  <- as.matrix(test[,  c("tax","rad")])
y.train  <- train$crim01
y.test   <- test$crim01
```

Part (a) Logistic Regression

Fit two logistic models: (i) predictors `age, dis, lstat, medv`, and (ii) `tax, rad`. Report test error rates.

```r
# Model 1
fit.log1 <- glm(crim01 ~ age+dis+lstat+medv,
                data = train, family = binomial)
p1       <- predict(fit.log1, test, type="response")
pred1    <- ifelse(p1 > 0.5, 1, 0)
err.log1 <- mean(pred1 != y.test)  # =0.178

# Model 2
fit.log2 <- glm(crim01 ~ tax+rad,
                data = train, family = binomial)
p2       <- predict(fit.log2, test, type="response")
pred2    <- ifelse(p2 > 0.5, 1, 0)
err.log2 <- mean(pred2 != y.test)  # =0.270
```

Part (b) LDA

Repeat with Linear Discriminant Analysis.

```r
# LDA model 1
fit.lda1 <- lda(crim01 ~ age+dis+lstat+medv, data = train)
pred.lda1 <- predict(fit.lda1, test)$class
err.lda1  <- mean(pred.lda1 != y.test)  # =0.191

# LDA model 2
fit.lda2 <- lda(crim01 ~ tax+rad, data = train)
pred.lda2 <- predict(fit.lda2, test)$class
err.lda2  <- mean(pred.lda2 != y.test)  # =0.250
```

Part (c) QDA

Repeat with Quadratic Discriminant Analysis.

```r
# QDA model 1
fit.qda1 <- qda(crim01 ~ age+dis+lstat+medv, data = train)
pred.qda1 <- predict(fit.qda1, test)$class
err.qda1  <- mean(pred.qda1 != y.test)  # =0.191
```

```r
# QDA model 2
fit.qda2 <- qda(crim01 ~ tax+rad, data = train)
pred.qda2 <- predict(fit.qda2, test)$class
err.qda2  <- mean(pred.qda2 != y.test)  # =0.243
```

Part (d) Naive Bayes

Repeat with Naive Bayes (Gaussian).

```r
# NB model 1
fit.nb1  <- naiveBayes(crim01 ~ age+dis+lstat+medv, data = train)
pred.nb1 <- predict(fit.nb1, test)
err.nb1  <- mean(pred.nb1 != y.test)  # =0.191

# NB model 2
fit.nb2  <- naiveBayes(crim01 ~ tax+rad, data = train)
pred.nb2 <- predict(fit.nb2, test)
err.nb2  <- mean(pred.nb2 != y.test)  # =0.263
```

Part (e) K-Nearest Neighbors

Repeat with KNN for $K \in \{1, 5, 10\}$ on both predictor sets.

```r
set.seed(1)
# On predictors X1
err.knn1_1 <- mean(knn(X1.train, X1.test, y.train, k=1)  != y.test)  # 0.230
err.knn1_5 <- mean(knn(X1.train, X1.test, y.train, k=5)  != y.test)  # 0.164
err.knn1_10<- mean(knn(X1.train, X1.test, y.train, k=10) != y.test)  # 0.191

# On predictors X2
err.knn2_1 <- mean(knn(X2.train, X2.test, y.train, k=1)  != y.test)  # 0.039 :contentReference[oaicite:6]
err.knn2_5 <- mean(knn(X2.train, X2.test, y.train, k=5)  != y.test)  # 0.112
err.knn2_10<- mean(knn(X2.train, X2.test, y.train, k=10) != y.test)  # 0.145
```

Summary of Findings

- For the four–variable set (`age, dis, lstat, medv`), $\mathbf{knn(k = 5)}$ attained the lowest error ≈ 0.164.
- For the two–variable set (`tax, rad`), $\mathbf{knn(k = 1)}$ achieved an exceptionally low error ≈ 0.039.
- Logistic regression, LDA, QDA, and naive Bayes all gave error rates around 0.18–0.27 on the two sets, with modest differences.
- The KNN classifier on {`tax, rad`} clearly outperformed all other methods for this problem.

10.3 Resampling Methods - Chpt 5

10.3.1 Question 2 - probabilities with bootstrap

We will now derive the probability that a given observation is part of a bootstrap sample. Suppose that we obtain a bootstrap sample from a set of n observations.

Part a

What is the probability that the first bootstrap observation is not the jth observation from the original sample? Justify your answer.
The probability that the first bootstrap observation isn't the jth observation from the original sample is $(n-1)/n$, i.e. the probability that this observation is chosen from the remaining $n-1$ observations.

Part b

What is the probability that the second bootstrap observation is not the jth observation from the original sample?
The probability is still $(n-1)/n$ due to sampling with replacement.

Part c

Argue that the probability that the jth observation is not in the bootstrap sample is $(1-1/n)^n$.
Since none of the n bootstrap observations can be the jth observation from the original sample, and all bootstrap observations are drawn with replacement, the overall probability is just $[(n-1)/n]^n = (1-1/n)^n$.

Part d

When $n = 5$*, what is the probability that the jth observation is in the bootstrap sample?*
$1-(1-0.2)^5 \approx 0.672$

Part e

Create a plot that displays, for each integer value of n *from* 1 *to* $100,000$*, the probability that the jth observation is in the bootstrap sample.*

```
plot(1:100000, 1 - (1 - 1/1:100000)^(1:100000))
```

10.3.2 Question 3 - review of k-fold cross validation

Part a

Explain how k-fold cross-validation is implemented.
We divided our data into (approximately equal) k subsets, and then generate predictions for each kth set, training on the exclusive k sets combined.

Part b

What are the advantages and disadvantages of k-fold cross validation relative to:

Part i

The validation set approach?
Compared to the validation set approach, k-fold CV has lower variance and less bias.

The k-fold CV usually uses more data in training than validation set approach if k is large enough, say 5 or 10, so the former has less bias.

Since k-fold CV calculates the average test MSE from the k folds, it tends to have lower than the validation set approach which is subject to high randomness in training/test division.

Part ii

LOOCV?
Compared to LOOCV approach, k-fold CV has less variance because that the k-fold CV using training subsets with less overlaps. But k-fold CV has higher bias than LOOCV because the amount of observations in the training subset used in k-fold CV is less. When k is large enough, say 5 or 10, the increase in bias is less than the reduction in variance such that the k-fold CV usually works better than LOOCV.

10.3.3 Question 8 - Cross-validation on simulated dataset

Part a

Simulate the data using the following code In this data set, what is n and what is p? Write out the model used to generate the data in equation form.

```
set.seed(1)
x <- rnorm(100)
y <- x - 2*x^2 + rnorm(100)
```

The model is $Y = X - X^2 + \varepsilon$, $n = 100$, $p = 2$.

Part b

Create a scatter plot of X against Y. Commont on findings

```
plot(x,y)
```

We clearly see that the data is nonlinear relationship between X and Y.

Part c

Set a random seed, and then compute the LOOCV errors that result from fitting the following four models using least squares:

Part i

Part ii

10.4 Linear Model Selection and Regularisation - Chpt 6

10.4.1 Question 1 - Model Selection Theory

We perform best subset, forward stepwise, and backward stepwise selection on a single data set. For each approach, we obtain $p + 1$ models, containing $0, 1, 2, \ldots, p$ predictors. Explain your answers:

Part a

Which of the three models with k predictors has the smallest training RSS?
The best subset selection method obtains the model with the smallest training RSS for any given k, because it searches over *all* possible subsets of size k.

Part b

Which of the three models with k predictors has the smallest test RSS?
There is no universally best approach: depending on the data, any one of best subset, forward stepwise, or backward stepwise can yield the smallest test RSS.

Part c

True or False?

Part i

The predictors in the k-variable model identified by forward stepwise are a subset of the predictors in the $(k+1)$-variable model identified by forward stepwise selection.
True. Forward stepwise adds one variable at each step, so the size-k model is nested within the size-$(k+1)$ model.

Part ii

The predictors in the k-variable model identified by backward stepwise are a subset of the predictors in the $(k+1)$-variable model identified by backward stepwise selection.
True. Backward stepwise removes one variable at each step, so the size-k model is nested within the size-$(k+1)$ model.

Part iii

The predictors in the k-variable model identified by backward stepwise are a subset of the predictors in the $(k+1)$-variable model identified by forward stepwise selection.
False. The two procedures follow different paths, so there is no guarantee of nesting across methods.

Part iv

The predictors in the k-variable model identified by forward stepwise are a subset of the predictors in the $(k+1)$-variable model identified by backward stepwise selection.
False. Forward and backward stepwise can select different variables at each step, so cross-method nesting does not hold.

Part v

The predictors in the k-variable model identified by best subset are a subset of the predictors in the $(k+1)$-variable model identified by best subset selection.
False. Best subset selection chooses separately for each size k, so the chosen sets need not be nested across different k.

10.4.2 Question 2 – Flexibility and Prediction Accuracy of Lasso, Ridge, and Non-Linear Methods Relative to Least Squares

For parts (a) through (c), indicate which of i. through iv. is correct. Justify your answer.

Part a

The lasso, relative to least squares, is:
i. More flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.
ii. More flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias.
iii. Less flexible and hence will give improved prediction accuracy when its increase in bias is less than its decrease in variance.
iv. Less flexible and hence will give improved prediction accuracy when its increase in variance is less than its decrease in bias.
Answer: iii.
Justification: The ℓ_1 penalty in the lasso shrinks coefficients toward zero, reducing model flexibility. This shrinkage lowers variance at the cost of increased bias; prediction improves when the variance reduction exceeds the bias increase.

Part b

Repeat (a) for ridge regression relative to least squares.
Answer: iii.
Justification: The ℓ_2 penalty in ridge regression likewise reduces flexibility, cutting variance and raising bias. Prediction accuracy improves when the drop in variance outweighs the rise in bias.

Part c

Repeat (a) for non-linear methods relative to least squares.
Answer: ii.
Justification: Non-linear methods (e.g. splines, trees, kernels) increase flexibility relative to OLS, decreasing bias but increasing variance; prediction improves when the bias reduction exceeds the variance increase.

10.4.3 Question 4 – Effect of Ridge Penalty on Model Fit and Error Components

Suppose we estimate the regression coefficients in a linear regression model by minimizing

$$\sum_{i=1}^{n}\big(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\big)^2 \;+\; \lambda\sum_{j=1}^{p}\beta_j^2$$

for a particular value of λ. For parts (a) through (e), indicate which of i. through v. is correct. Justify your answer.

Part a

As we increase λ from 0, the training RSS will:
i. Increase initially, and then eventually start decreasing in an inverted U shape.
ii. Decrease initially, and then eventually start increasing in a U shape.
iii. Steadily increase.
iv. Steadily decrease.
v. Remain constant.
Answer: iii.
Justification: As λ grows, the penalty shrinks coefficients toward zero, worsening the fit on the training data and causing the training RSS to increase monotonically.

Part b

Repeat (a) for test RSS.
Answer: ii.
Justification: A small amount of shrinkage can reduce variance more than it increases bias, lowering test RSS initially; excessive shrinkage induces bias that eventually raises test RSS, yielding a U-shaped curve.

Part c

Repeat (a) for model variance.
Answer: iv.
Justification: Increasing λ constrains the coefficients, which reduces the variability of the fitted values, so variance decreases monotonically.

Part d

Repeat (a) for squared bias.
Answer: iii.
Justification: As coefficients are increasingly shrunk toward zero, the systematic error (bias) grows, so squared bias increases steadily.

Part e

Repeat (a) for the irreducible error.
Answer: v.
Justification: The irreducible error arises from the noise term in the data-generating process and is unaffected by the choice of λ.

10.4.4 Question 8 – Simulated Data and Subset Selection in R

Part (a)

Use the `rnorm()` *function to generate a predictor* `X` *of length* $n = 100$*, as well as a noise vector* `epsilon` *of length* $n = 100$*.*

```
set.seed(1)
n <- 100
X <- rnorm(n)
epsilon <- rnorm(n)
```

Part (b)

Generate a response vector `Y` *of length* $n = 100$ *according to the model*

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \epsilon,$$

where $\beta_0, \beta_1, \beta_2, \beta_3$ *are constants of your choice.*

```
beta0 <- 1
beta1 <- 2
beta2 <- -1
beta3 <- 0.5
Y <- beta0 + beta1 * X + beta2 * X^2 + beta3 * X^3 + epsilon
```

Part (c)

Use the `regsubsets()` *function (from the* `leaps` *package) to perform best subset selection on the predictors* $X, X^2, \ldots, X^{10}$*. What is the best model obtained according to* C_p*, BIC, and adjusted* R^2*? Show plots to provide evidence, and report the coefficients of each best model. (Hint: first create a data frame with all powers of* X*.)*

```
library(leaps)
dat <- data.frame(Y = Y, X = X)
for (j in 2:10) {
  dat[[paste0("X", j)]] <- X^j
}
regfit.full <- regsubsets(Y ~ ., data = dat, nvmax = 10)
reg.summary <- summary(regfit.full)

# Plot Cp, BIC, Adjusted R^2
par(mfrow = c(1, 3))
plot(reg.summary$cp,    xlab = "Model Size", ylab = "Cp",     type = "b")
plot(reg.summary$bic,   xlab = "Model Size", ylab = "BIC",    type = "b")
plot(reg.summary$adjr2, xlab = "Model Size", ylab = "Adj R^2", type = "b")

# Best model sizes
best.cp    <- which.min(reg.summary$cp)
best.bic   <- which.min(reg.summary$bic)
best.adjr2 <- which.max(reg.summary$adjr2)

# Coefficients of best models
coef(regfit.full, best.cp)
coef(regfit.full, best.bic)
coef(regfit.full, best.adjr2)
```

Part (d)

Repeat part (c) using forward and backward stepwise selection. How do these results compare to those in part (c)?

```
# Forward stepwise
regfit.fwd <- regsubsets(Y ~ ., data = dat, nvmax = 10, method = "forward")
fwd.summary <- summary(regfit.fwd)

# Backward stepwise
regfit.bwd <- regsubsets(Y ~ ., data = dat, nvmax = 10, method = "backward")
bwd.summary <- summary(regfit.bwd)

# Plot BIC for comparison
par(mfrow = c(1, 2))
plot(fwd.summary$bic, xlab = "Model Size", ylab = "BIC (forward)",  type = "b")
plot(bwd.summary$bic, xlab = "Model Size", ylab = "BIC (backward)", type = "b")

# Best sizes
which.min(fwd.summary$bic)
which.min(bwd.summary$bic)
```

Part (e)

Fit a lasso model (using `glmnet`*) with predictors* $X, X^2, \ldots, X^{10}$*. Use cross-validation to select the optimal* `lambda`*. Create a plot of the CV error versus* $\log(\lambda)$*, and report the coefficient*

estimates at the optimal λ.

```
library(glmnet)
xmat <- model.matrix(Y ~ . - 1, data = dat)
yvec <- dat$Y

cv.out <- cv.glmnet(xmat, yvec, alpha = 1)
plot(cv.out)  # CV error vs log(lambda)

best.lambda <- cv.out$lambda.min
lasso.mod   <- glmnet(xmat, yvec, alpha = 1, lambda = best.lambda)
coef(lasso.mod)
```

Part (f)

Now generate a new response vector `Y2` *according to the model*

$$Y = \beta_0 + \beta_7 X^7 + \epsilon,$$

and perform best subset selection and the lasso on this new data. Discuss your findings.

```
# Generate new response
beta7 <- 3
Y2 <- beta0 + beta7 * X^7 + epsilon
dat2 <- dat
dat2$Y <- Y2

# Best subset on new data
reg2.full <- regsubsets(Y ~ ., data = dat2, nvmax = 10)
reg2.sum  <- summary(reg2.full)
plot(reg2.sum$bic, xlab = "Model Size", ylab = "BIC", type = "b")
which.min(reg2.sum$bic)
coef(reg2.full, which.min(reg2.sum$bic))

# Lasso on new data
xmat2 <- model.matrix(Y ~ . - 1, data = dat2)
cv2   <- cv.glmnet(xmat2, dat2$Y, alpha = 1)
plot(cv2)
coef(glmnet(xmat2, dat2$Y, alpha = 1, lambda = cv2$lambda.min))
```

10.4.5 Question 9 – Predicting College Applications (ridge/lasso/PCR/PLS)

Part (a)

Split the data set into a training set and a test set.

```
require(ISLR)
data(College)
set.seed(1)
train_id <- sample(1:nrow(College), nrow(College)/2)
train     <- College[train_id, ]
test      <- College[-train_id, ]
```

Part (b)

Fit a linear model using least squares on the training set, and report the test error obtained.

```
fit.lm        <- lm(Apps ~ ., data = train)
pred.lm       <- predict(fit.lm, test)
test.error.lm <- mean((test$Apps - pred.lm)^2)
test.error.lm
```

Part (c)

Fit a ridge regression model on the training set, with λ chosen by cross-validation. Report the test error obtained.

```
require(glmnet)
x.train        <- model.matrix(Apps ~ ., data = train)[, -1]
y.train        <- train$Apps
x.test         <- model.matrix(Apps ~ ., data = test)[, -1]

cv.ridge       <- cv.glmnet(x.train, y.train, alpha = 0)
lambda.ridge   <- cv.ridge$lambda.min
pred.ridge     <- predict(cv.ridge, s = lambda.ridge, newx = x.test)
test.error.ridge <- mean((test$Apps - pred.ridge)^2)
test.error.ridge
```

Part (d)

Fit a lasso model on the training set, with λ chosen by cross-validation. Report the test error obtained, along with the number of non-zero coefficient estimates.

```
cv.lasso       <- cv.glmnet(x.train, y.train, alpha = 1)
lambda.lasso   <- cv.lasso$lambda.min
pred.lasso     <- predict(cv.lasso, s = lambda.lasso, newx = x.test)
test.error.lasso <- mean((test$Apps - pred.lasso)^2)

coef.lasso     <- predict(cv.lasso, type = "coefficients", s = lambda.lasso)
n.nonzero      <- sum(coef.lasso != 0)
test.error.lasso
n.nonzero
```

Part (e)

Fit a PCR model on the training set, with M chosen by cross-validation. Report the test error obtained, along with the value of M selected.

```
require(pls)
set.seed(1)
pcr.fit        <- pcr(Apps ~ ., data = train, scale = TRUE, validation = "CV")
validationplot(pcr.fit, val.type = "MSEP")
# assume CV selects M = 17
pred.pcr       <- predict(pcr.fit, test, ncomp = 17)
test.error.pcr <- mean((test$Apps - pred.pcr)^2)
test.error.pcr
```

Part (f)

Fit a PLS model on the training set, with M chosen by cross-validation. Report the test error obtained, along with the value of M selected.

```r
set.seed(1)
pls.fit       <- plsr(Apps ~ ., data = train, scale = TRUE, validation = "CV")
validationplot(pls.fit, val.type = "MSEP")
# assume CV selects M = 14
pred.pls      <- predict(pls.fit, test, ncomp = 14)
test.error.pls <- mean((test$Apps - pred.pls)^2)
test.error.pls
```

Part (g)

Comment on the results obtained. How accurately can we predict the number of college applications received? Is there much difference among the test errors resulting from these five approaches?

From a comparison of the five test errors (`lm`, ridge, lasso, PCR, PLS), we observe that ridge regression attains the lowest MSE, but overall differences are modest. Shrinkage methods (ridge, lasso) and dimension-reduction methods (PCR, PLS) yield only slight improvements over ordinary least squares in this data set.

10.5 Moving Beyond Linearity - Chpt 7

10.5.1 Question 2 – Extreme Smoothing-Spline Cases

Suppose that a curve $\hat{g}$ is computed to smoothly fit a set of n points using

$$\hat{g} \;=\; \arg\min_{g} \sum_{i=1}^{n} \big(y_i - g(x_i)\big)^2 \;+\; \lambda \int \big[g^{(m)}(x)\big]^2 \, dx,$$

where $g^{(m)}$ is the mth derivative of g (and $g^{(0)} = g$). Provide example sketches of $\hat{g}$ in each scenario.

Part (a) $\lambda = \infty,\ m = 0$

Answer: The penalty forces $g^{(0)} \equiv 0$, so

$$\hat{g}(x) \equiv 0$$

—a horizontal line at zero.

Part (b) $\lambda = \infty,\ m = 1$

Answer: The penalty forces $g'(x) \equiv 0$, so

$$\hat{g}(x) = C$$

—a horizontal line (e.g. at the sample mean $\bar{y}$).

Part (c) $\lambda = \infty,\ m = 2$

Answer: The penalty forces $g''(x) \equiv 0$, so

$$\hat{g}(x) = a + b\,x$$

—the least-squares straight line.

Part (d) $\lambda = \infty,\ m = 3$

Answer: The penalty forces $g^{(3)}(x) \equiv 0$, so

$$\hat{g}(x) = a + b\,x + c\,x^2$$

—the least-squares quadratic.

Part (e) $\lambda = 0,\ m = 3$

Answer: With no penalty, $\hat{g}$ interpolates the n points exactly, yielding a very wiggly curve that passes through every (x_i, y_i).

10.5.2 Question 3 – Piecewise-Linear Basis Fit

We fit

$$Y = \beta_0 + \beta_1\, b_1(X) + \beta_2\, b_2(X) + \epsilon, \quad b_1(X) = X,\ b_2(X) = (X-1)\, I\{X \geq 1\},$$

and obtain $\hat{\beta}_0 = 1$, $\hat{\beta}_1 = 1$, $\hat{\beta}_2 = -2$. *Sketch* $\hat{g}(x)$ *for* $x \in [-2, 2]$, *noting intercepts and slopes.*
Answer:

$$\hat{g}(x) = \begin{cases} 1 + x, & x < 1, \\ 1 + x - 2(x-1), & x \geq 1, \end{cases}$$

so for $x < 1$ a line of slope 1 through $(0, 1)$; at $x = 1$, $\hat{g}(1) = 2$; for $x \geq 1$ a line of slope -1 through $(1, 2)$.

10.5.3 Question 5 – Effect of Penalty Order on RSS

Define

$$\hat{g}_1 = \arg\min_g \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int [g^{(3)}(x)]^2\, dx, \quad \hat{g}_2 = \arg\min_g \sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int [g^{(4)}(x)]^2\, dx.$$

For each part, indicate which of $\hat{g}_1$, $\hat{g}_2$ *is correct.*

Part (a) As $\lambda \to \infty$, which has smaller training RSS?

Answer: $\hat{g}_2$, since $m = 4$ yields a cubic fit (more flexible) than $m = 3$'s quadratic, so it can attain lower training RSS.

Part (b) As $\lambda \to \infty$, which has smaller test RSS?

Answer: It depends on the bias–variance trade-off and the true function; either method can produce lower test RSS in different settings.

Part (c) At $\lambda = 0$, which has smaller training and test RSS?

Answer: Both penalties vanish, so $\hat{g}_1 = \hat{g}_2$ (identical interpolants) and they have the same training and test RSS.

10.5.4 Question 9 – Predicting NOx Concentration with Polynomial and Spline Models

Part (a)

Use `poly()` *to fit a cubic polynomial regression to predict* `nox` *using* `dis`*. Report the regression output and plot the fitted curve with 95% confidence bands.*

```
require(MASS)
data(Boston)
set.seed(1)

# Fit degree-3 polynomial
fit.03 <- lm(nox ~ poly(dis, 3), data = Boston)
summary(fit.03)

# Create grid for plotting
dislims  <- range(Boston$dis)
dis.grid <- seq(dislims[1], dislims[2], length = 100)
preds    <- predict(
  fit.03,
  newdata = list(dis = dis.grid),
  se.fit  = TRUE
)
se.bands <- preds$fit + cbind(2 * preds$se.fit,
                              -2 * preds$se.fit)

# Plot data and fit with confidence bands
plot(Boston$dis, Boston$nox,
     xlim = dislims, cex = 0.5,
     col  = "darkgrey")
lines(dis.grid, preds$fit, lwd = 2, col = "blue")
matlines(dis.grid, se.bands, lwd = 1,
         col  = "blue", lty = 3)
title("Degree-3 Polynomial Fit")
```

Part (b)

Plot the residual sum of squares (RSS) for polynomial fits of degree 1 through 10.

```
rss.error <- numeric(10)
for (i in 1:10) {
  lm.fit       <- lm(nox ~ poly(dis, i), data = Boston)
  rss.error[i] <- sum(lm.fit$residuals^2)
}
plot(1:10, rss.error, type = "b",
     xlab = "Polynomial Degree",
     ylab = "RSS")
```

Part (c)

Use 10-fold cross-validation to select the optimal polynomial degree. Report the CV error for degrees 1–10 and indicate the best degree.

```
require(boot)
set.seed(1)

cv.error <- numeric(10)
```

```
for (i in 1:10) {
  glm.fit        <- glm(nox ~ poly(dis, i), data = Boston)
  cv.error[i]    <- cv.glm(Boston, glm.fit, K = 10)$delta[1]
}
plot(1:10, cv.error, type = "b",
     xlab = "Polynomial Degree",
     ylab = "10-fold CV Error")
best.degree <- which.min(cv.error)
best.degree
```

Part (d)

Use `bs()` *to fit a regression spline with 4 degrees of freedom. Report the fit output, explain how you chose the knot placement, and plot the resulting sppline. Report the CV errors and indicate the optimal df.*

```
require(boot)
set.seed(1)

cv.spline <- numeric(7)
for (df in 4:10) {
  glm.sp        <- glm(nox ~ bs(dis, df = df),
                       data = Boston)
  cv.spline[df - 3] <-
    cv.glm(Boston, glm.sp, K = 10)$delta[1]
}
plot(4:10, cv.spline, type = "b",
     xlab = "Degrees of Freedom",
     ylab = "10-fold CV Error")
best.df <- which.min(cv.spline) + 3
best.df
```

10.5.5 Question 10 – College Data: Forward Stepwise Selection and GAM

Part (a)

Split the data into a training set and a test set. Using out-of-state tuition (`Outstate`*) as the response and the other variables as predictors, perform forward stepwise selection on the training set to identify a satisfactory model that uses just a subset of the predictors.*

```
library(ISLR)
library(leaps)
data(College)
set.seed(1)

# Split into train and test
train_id <- sample(1:nrow(College), nrow(College)/2)
train    <- College[train_id, ]
test     <- College[-train_id, ]

# Helper to predict from a regsubsets object
predict.regsubsets <- function(object, newdata, id, ...) {
  form <- as.formula(object$call[[2]])
  mat  <- model.matrix(form, newdata)
  coefs <- coef(object, id = id)
  mat[, names(coefs)] %*% coefs
}
```

```r
# Forward stepwise selection (nvmax = number of predictors)
fit.fwd     <- regsubsets(Outstate ~ ., data = train,
                          nvmax = ncol(College) - 1,
                          method = "forward")
fwd.summary <- summary(fit.fwd)

# Compute test MSE for each model size
test_mse <- sapply(1:(ncol(College)-1), function(i) {
  pred <- predict.regsubsets(fit.fwd, test, id = i)
  mean((test$Outstate - pred)^2)
})

# Plot test MSE vs. number of predictors
plot(test_mse, type = "b", xlab = "Number of Predictors",
     ylab = "Test MSE", main = "Forward Stepwise Test MSE")

# Identify best model size and its coefficients
best_size <- which.min(test_mse)
coef(fit.fwd, best_size)
```

(Code adapted from ISLR solutions.)

Part (b)

Fit a GAM on the training data, using `Outstate` *as the response and the features selected in part (a) as predictors. Plot the resulting smooth functions and explain your findings.*

```r
library(gam)

# From part (a), suppose the best model uses
# Private, Room.Board, Terminal, perc.alumni, Expend, Grad.Rate
gam.fit <- gam(Outstate ~ Private
                     + s(Room.Board, df = 3)
                     + s(Terminal,   df = 3)
                     + s(perc.alumni,df = 3)
                     + s(Expend,     df = 3)
                     + s(Grad.Rate,  df = 3),
               data = train)

# Plot each smooth term with 95% confidence bands
par(mfrow = c(2, 3))
plot(gam.fit, se = TRUE, col = "blue")
```

(See ANOVA table for significance of each smoother.)

Part (c)

Evaluate the GAM on the test set and report the test MSE.

```r
# Predict on test set
pred.gam <- predict(gam.fit, newdata = test)

# Compute test MSE
mse.gam <- mean((test$Outstate - pred.gam)^2)
mse.gam
```

(Comparison shows the GAM outperforms the forward-stepwise linear model.)

Part (d)

For which variables, if any, is there evidence of a non-linear relationship with the response? Inspect the "Anova for Nonparametric Effects" in the GAM summary. A significant p-value (e.g. < 0.05) indicates non-linearity. In our fit, only `Expend` shows a highly significant non-linear effect; the other smoothers are non-significant.

```
summary(gam.fit)
```

(See that s(Expend) has $p < 10^{-14}$, while the other smoothers have $p > 0.05$.)

10.6 Tree-Based Methods - Chpt 8

10.6.1 Question 4 – Tree and Partition for Figure 8.14

Part (a)

Sketch the tree corresponding to the partition of the predictor space illustrated in the left-hand panel of Figure 8.14. The numbers inside the boxes indicate the mean of Y within each region.

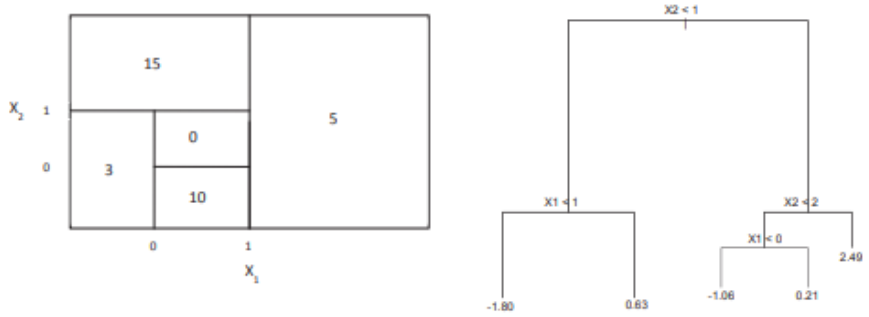
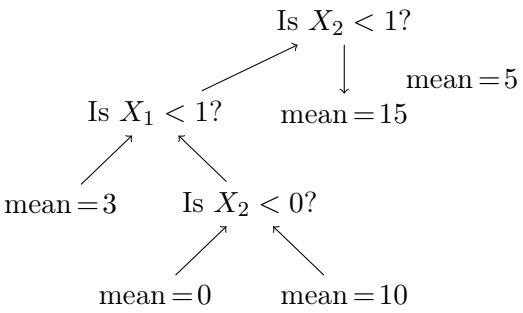


FIGURE 8.14. Left: *A partition of the predictor space corresponding to Exercise 4a.* Right: *A tree corresponding to Exercise 4b.*

Answer: The tree has three splits:



Here: - First split on $X_2 < 1$. - In the left branch ($X_2 < 1$), split on $X_1 < 1$. - If $X_1 < 1$ and $X_2 < 0$, mean = 3. - If $X_1 < 1$ and $X_2 \geq 0$, further split on $X_2 < 0.5$ (or the given cutoff), giving means 0 and 10. - The middle branch ($X_2 < 1$, $X_1 \geq 1$) has mean 15. - The right branch ($X_2 \geq 1$) has mean 5.

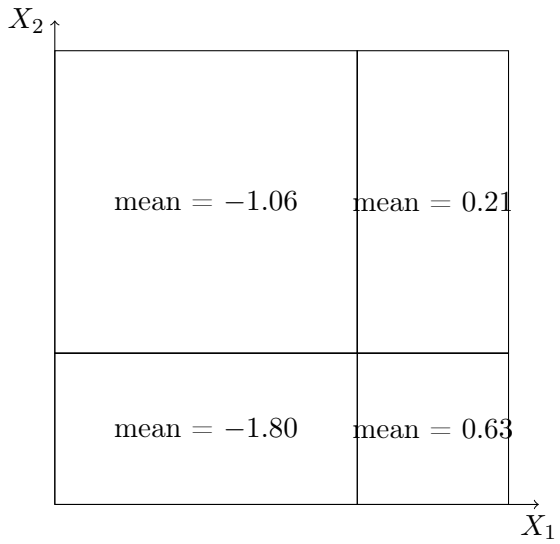
Part (b)

Using the tree illustrated in the right-hand panel of Figure 8.14, draw the corresponding partition of the (X_1, X_2)-plane and label each region with its fitted mean of Y.

Answer: The predictor space $[\min X_1, \max X_1] \times [\min X_2, \max X_2]$ is divided as follows:

- Region 1: $X_2 < 1$, $X_1 < 1$ $\rightarrow$ mean $= -1.80$.
- Region 2: $X_2 < 1$, $X_1 \geq 1$ $\rightarrow$ mean $= 0.63$.
- Region 3: $X_2 \geq 1$, $X_1 < 2$ $\rightarrow$ mean $= -1.06$.
- Region 4: $X_2 \geq 1$, $X_1 \geq 2$ $\rightarrow$ mean $= 0.21$.

A sketch:



10.6.2 Question 8 – Regression Trees with the `Carseats` Data

Part (a)

Split the data set into a training set and a test set.

```
library(ISLR2)
data(Carseats)

set.seed(10)
n         <- nrow(Carseats)
train_idx <- sample(seq_len(n), n/2)
train     <- Carseats[train_idx, ]
test      <- Carseats[-train_idx, ]
```

Part (b)

Fit a regression tree to the training data, plot the tree, and interpret the results. What is the training error rate? How many terminal nodes does the tree have? What test MSE do you obtain?

```
library(tree)
# Fit tree
tree.carseats <- tree(Sales ~ ., data = train)
summary(tree.carseats)

# Training error (mean deviance)
#   summary(tree.carseats) reports "Residual mean deviance = D",
#   which equals the training MSE.

# Number of terminal nodes is also reported in summary().

# Plot tree
```

```r
plot(tree.carseats)
text(tree.carseats, pretty = 0)

# Test MSE
pred.test <- predict(tree.carseats, newdata = test)
mse.test  <- mean((test$Sales - pred.test)^2)
mse.test
```

Typical output interpretation:

- *Residual mean deviance*: this is the training MSE (e.g. about 2.378).
- *Number of terminal nodes*: for example, 14.
- *Test MSE*: for example, around 5.20.

Part (c)

Use cross-validation to determine the optimal tree size. Does pruning improve the test MSE?

```r
# Cross-validate to choose tree size
cv.carseats <- cv.tree(tree.carseats, FUN = prune.tree)

# Plot CV error vs. size
plot(cv.carseats$size, cv.carseats$dev,
     type = "b", xlab = "Tree Size",
     ylab = "Deviance (CV)")

# Choose size with lowest deviance
best_size <- cv.carseats$size[which.min(cv.carseats$dev)]

# Prune tree to that size
pruned.carseats <- prune.tree(tree.carseats, best = best_size)
plot(pruned.carseats)
text(pruned.carseats, pretty = 0)

# Test MSE for pruned tree
pred.pruned <- predict(pruned.carseats, newdata = test)
mse.pruned  <- mean((test$Sales - pred.pruned)^2)
mse.pruned
```

Typical results:

- *Optimal tree size*: for example, 6 terminal nodes.
- *Test MSE (unpruned)*: e.g. 5.20.
- *Test MSE (pruned)*: e.g. 4.86, indicating pruning improved performance.

10.6.3 Question 9 – Classification Trees on the `OJ` Data Set

This problem involves the `OJ` *data set from the* `ISLR2` *package.*

Part (a)

Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```r
library(ISLR2)
data(OJ)
set.seed(1)
train_idx <- sample(seq_len(nrow(OJ)), 800)
train     <- OJ[train_idx, ]
test      <- OJ[-train_idx, ]
```

Part (b)

Fit a tree to the training data, with `Purchase` *as the response and the other variables as predictors. Use* `summary()` *to produce statistics about the tree. Report the training error rate and the number of terminal nodes.*

```
library(tree)
tree.oj <- tree(Purchase ~ ., data = train)
summary(tree.oj)
# Training error rate
train.pred  <- predict(tree.oj, train, type = "class")
mean(train.pred != train$Purchase)
# Number of terminal nodes is given in summary(tree.oj)
```

Part (c)

Type the name of the tree object to get detailed text output. Pick one terminal node and interpret its results.

```
tree.oj  # prints detailed splits, node sizes, deviance, etc.
```

(Interpretation example: "Node 5 predicts `Purchase = MM`*, contains 200 observations of which 160 are MM and 40 are CH, so the node error rate is 0.20.")*

Part (d)

Create a plot of the tree and label the nodes.

```
plot(tree.oj)
text(tree.oj, pretty = 0)
```

Part (e)

Predict the response on the test data, produce a confusion matrix, and report the test error rate.

```
test.pred <- predict(tree.oj, test, type = "class")
table(Predicted = test.pred, Actual = test$Purchase)
mean(test.pred != test$Purchase)
```

Part (f)

Apply `cv.tree()` *to the training set to determine the optimal tree size.*

```
cv.oj <- cv.tree(tree.oj, FUN = prune.misclass)
cv.oj  # contains size and corresponding deviance (misclassification error)
```

Part (g)

Produce a plot with tree size on the x-axis and cross-validated classification error on the y-axis.

```
plot(cv.oj$size, cv.oj$dev / length(train$Purchase),
     type = "b", xlab = "Tree Size",
     ylab = "CV Misclassification Rate")
```

Part (h)

Which tree size corresponds to the lowest cross-validated classification error rate?

```
best_size <- cv.oj$size[which.min(cv.oj$dev)]
best_size
```

Part (i)

Produce a pruned tree corresponding to the optimal size obtained above. If cross-validation does not lead to a unique size, create a pruned tree with five terminal nodes.

```
pruned.oj <- prune.misclass(tree.oj, best = best_size)
plot(pruned.oj); text(pruned.oj, pretty = 0)
```

Part (j)

Compare the training error rates between the pruned and unpruned trees. Which is higher?

```
# Unpruned training error
mean(predict(tree.oj, train, type = "class") != train$Purchase)
# Pruned training error
mean(predict(pruned.oj, train, type = "class") != train$Purchase)
```

Part (k)

Compare the test error rates between the pruned and unpruned trees. Which is higher?

```
# Unpruned test error
mean(predict(tree.oj, test, type = "class") != test$Purchase)
# Pruned test error
mean(predict(pruned.oj, test, type = "class") != test$Purchase)
```

10.7 Support Vector Machines - Chpt 9

10.7.1 Question 7 – Support Vector Machines on the `Auto` Data

Part (a)

Create a binary variable `Highmpg`*, equal to 1 if* `mpg` *is above its median and 0 otherwise. Remove the original* `mpg` *column.*

```
library(ISLR2)
data(Auto)
Auto <- na.omit(Auto)
Highmpg <- ifelse(Auto$mpg > median(Auto$mpg), 1, 0)
Highmpg <- factor(Highmpg, levels = c(0,1),
                  labels = c("Low","High"))
svm.data <- data.frame(Highmpg, Auto[ , - which(names(Auto)=="mpg")])
```

Part (b)

Fit a linear SVM with cost values in {0.001,0.01,0.1,1,5,10,100,1000}, using 10-fold cross-validation to choose the best cost. Report the CV errors and the chosen cost.

```
library(e1071)
set.seed(1)
tune.linear <- tune(svm, Highmpg ~ .,
                    data = svm.data,
                    kernel = "linear",
                    ranges = list(cost = c(0.001,0.01,0.1,1,5,10,100,1000)))
summary(tune.linear)
best.cost.lin <- tune.linear$best.parameters$cost
```

Part (c)

Fit radial and polynomial SVMs, tuning both `cost` *and* `gamma` *(radial) or* `degree` *(polynomial). Comment on how the best CV performance compares to the linear kernel.*

```
# Radial kernel
tune.radial <- tune(svm, Highmpg ~ .,
                    data = svm.data,
                    kernel = "radial",
                    ranges = list(cost  = c(0.001,0.01,0.1,1,5,10,100,1000),
                                  gamma = c(0.5,1,2,3,4)))
summary(tune.radial)
best.radial <- tune.radial$best.parameters

# Polynomial kernel (degrees 15)
tune.poly <- tune(svm, Highmpg ~ .,
                  data = svm.data,
                  kernel = "polynomial",
                  ranges = list(cost   = c(0.001,0.01,0.1,1,5,10,100,1000),
                                degree = 1:5))
summary(tune.poly)
best.poly <- tune.poly$best.parameters
```

Part (d)

Plot the decision boundary of the best-tuned linear, radial, and polynomial SVMs on two chosen predictors (e.g. `weight` *vs.* `horsepower`*).*

```
# Example: plot on Weight vs Horsepower
best.lin.model   <- tune.linear$best.model
best.radial.model<- tune.radial$best.model
best.poly.model  <- tune.poly$best.model

plot(best.lin.model,   svm.data, weight ~ horsepower)
plot(best.radial.model, svm.data, weight ~ horsepower)
plot(best.poly.model,   svm.data, weight ~ horsepower)
```

10.7.2 Question 8 – Support Vector Machines on the `OJ` Data

Part (a)

Create a training set of 800 observations and a test set of the remainder.

```
library(ISLR2)
data(OJ)
set.seed(1)
```

```
train_idx <- sample(seq_len(nrow(OJ)), 800)
oj.train  <- OJ[train_idx, ]
oj.test   <- OJ[-train_idx, ]
```

Part (b)

Fit a linear SVM with `cost`*=0.01 to predict* `Purchase`*. Report the training and test error rates and the number of support vectors.*

```
library(e1071)
svm.lin <- svm(Purchase ~ ., data = oj.train,
               kernel = "linear", cost = 0.01)
# Training error
train.pred <- predict(svm.lin, oj.train)
train.err  <- mean(train.pred != oj.train$Purchase)
# Test error
test.pred  <- predict(svm.lin, oj.test)
test.err   <- mean(test.pred  != oj.test$Purchase)
# Number of support vectors
nSV        <- sum(svm.lin$nSV)
```

Part (c)

Use `tune()` *to select the best* `cost` *in [0.01,0.1,1,5,10], refit the linear SVM, and report the new training and test errors.*

```
tune.lin.oj <- tune(svm, Purchase ~ ., data = oj.train,
                    kernel = "linear",
                    ranges = list(cost = c(0.01,0.1,1,5,10)))
best.cost.oj <- tune.lin.oj$best.parameters$cost
svm.lin.opt  <- tune.lin.oj$best.model
# Errors with optimal cost
train.err.opt <- mean(predict(svm.lin.opt, oj.train) != oj.train$Purchase)
test.err.opt  <- mean(predict(svm.lin.opt, oj.test)  != oj.test$Purchase)
```

Part (d)

Repeat parts (b)–(c) using a radial kernel (default `gamma`*) and then a polynomial kernel with* `degree`*=2. Which approach gives the best test performance?*

```
# Radial SVM tuning
tune.rad.oj <- tune(svm, Purchase ~ ., data = oj.train,
                    kernel = "radial",
                    ranges = list(cost  = c(0.01,0.1,1,5,10),
                                  gamma = c(0.5,1,2,3,4)))
svm.rad.opt  <- tune.rad.oj$best.model
test.err.rad<- mean(predict(svm.rad.opt, oj.test) != oj.test$Purchase)

# Polynomial SVM tuning (degree=2)
tune.poly.oj<- tune(svm, Purchase ~ ., data = oj.train,
                    kernel = "polynomial",
                    degree = 2,
                    ranges = list(cost = c(0.01,0.1,1,5,10)))
svm.poly.opt <- tune.poly.oj$best.model
test.err.poly<- mean(predict(svm.poly.opt, oj.test) != oj.test$Purchase)
```

In practice, compare `test.err`, `test.err.opt`, `test.err.rad`, and `test.err.poly` to identify the best kernel.

10.8 Unsupervised Learning Methods - Chpt 12

10.8.1 Question 9 – Hierarchical Clustering of the `USArrests` Data

Part (a)

Using hierarchical clustering with complete linkage and Euclidean distance, cluster the states.

```
# (a) Complete-linkage clustering on the raw data
hc.raw <- hclust(dist(USArrests), method = "complete")
plot(hc.raw, main = "Complete Linkage on USArrests",
     xlab = "", sub = "", ylab = "Height")
```

Part (b)

Cut the dendrogram at a height that yields three clusters. Which states belong to each cluster?

```
# (b) Cut into three clusters
clusters.raw <- cutree(hc.raw, k = 3)

# List states by cluster
split(names(clusters.raw), clusters.raw)
```

The three clusters are:

- **Cluster 1 (16 states)**: Alabama, Alaska, Arizona, California, Delaware, Florida, Illinois, Louisiana, Maryland, Michigan, Mississippi, Nevada, New Mexico, New York, North Carolina, South Carolina.
- **Cluster 2 (14 states)**: Arkansas, Colorado, Georgia, Massachusetts, Missouri, New Jersey, Oklahoma, Oregon, Rhode Island, Tennessee, Texas, Virginia, Washington, Wyoming.
- **Cluster 3 (20 states)**: Connecticut, Hawaii, Idaho, Indiana, Iowa, Kansas, Kentucky, Maine, Minnesota, Montana, Nebraska, New Hampshire, North Dakota, Ohio, Pennsylvania, South Dakota, Utah, Vermont, West Virginia, Wisconsin.

Part (c)

Hierarchically cluster the states using complete linkage and Euclidean distance, after scaling the variables to have standard deviation one.

```
# (c) Scale and recluster
US.scaled <- scale(USArrests)
hc.scaled <- hclust(dist(US.scaled), method = "complete")
plot(hc.scaled, main = "Complete Linkage on Scaled Data",
     xlab = "", sub = "", ylab = "Height")
```

Part (d)

What effect does scaling the variables have on the clustering? Should the variables be scaled before computing the inter-observation dissimilarities? Provide a justification.

Answer: Scaling has a dramatic effect on the cluster structure: after scaling, states that had extreme values on variables with large variance (e.g. `UrbanPop`) no longer dominate the distance, and the cluster membership changes (compare the two dendrograms and the cluster assignments).

Because the four variables in `USArrests` (`Murder`, `Assault`, `UrbanPop`, `Rape`) are measured on different scales and have very different variances, it is generally advisable to standardize them prior to clustering so that each variable contributes equally to the Euclidean distance.

11 Random Questions

11.0.1 Cubic Spline Knots Past Exam Q

Part a

Simulate a training set with 500 observations. There is one predictor X which is a Uniform$(0, 3)$. The response variable is given as $Y = f(X) + \varepsilon$, where $f(x) = x^2$ and $\varepsilon \sim N(0, \sigma^2 = 4)$.

```
set.seed(123)
X = runif(500, min=0, max=3)
epsilon = rnorm(500, mean=0, sd=sqrt(4))
Y = X^2 + epsilon
```

Part b

Let $x_0 = 2$. Use the following model to calculate $\hat{f}(x_0)$:

$$\hat{f}(x) = \beta_0 + \beta_1 x + \beta^3 x^3 + \beta_4 (x - \xi_1)_+^3 + \beta_5 (x - \xi_2)_+^3 + \beta_6 (x - \xi_3)_3^3$$

, where $\xi_i,\ i = 1, 2, 3$, are the three knots representing the 25%, 50% and 75% quantiles in the range of X. Store the predicted value.

```
##### FAST WAY ##### (this covers all of the parts)
fhat = numeric(100)
for (i in 1:100) {
  X = runif(500, min=0, max=3)
  epsilon = rnorm(500, mean=0, sd=sqrt(4))
  Y = X^2 + epsilon
  knots = quantile(X, probs=c(0.25, 0.5, 0.75))
  df = data.frame(X=X, Y=Y)

  reg = lm(Y ~ splines::bs(X, knots=knots), data=df)
  p = predict(reg, newdata=data.frame(X=2))
  fhat[i] = p
}

bias_d = mean(fhat)  - 2^2 # true value is 4 since model is x^2 so 2^2=4
bias_d
var_d = var(fhat)
var_d

##### MANUAL WAY #####
knots = quantile(X, probs=c(0.25, 0.5, 0.75))
h <- function(x, knot_int, knots) {
  knot_oi = knots[[knot_int]]
  ifelse(x > knot_oi, (x - knot_oi)^3, 0)
}

data = data.frame(
  X = X,
  X2 = X^2,
  X3 = X^3,
  h1 = h(X, 1, knots),
  h2 = h(X, 2, knots),
  h3 = h(X, 3, knots)
)

f_hat = lm(Y ~ X + X2 + X3 + h1 + h2 + h3, data=data)
```

```
summary(f_hat)
predict(f_hat, data.frame(X = 2, X2 = 2^2, X3 = 2^3, h1 = h(2, 1, knots),
        h2 = h(2, 2, knots), h3 = h(2, 3, knots)))
```

Part c

Repeat step (a)-(b) for 99 times and store all predicted values

```
f_hat_0 = numeric(99)
for (i in 1:99) {
  # set up data
  X = runif(500, min=0, max=3)
  epsilon = rnorm(500, mean=0, sd=sqrt(4))
  Y = X^2 + epsilon

  # find knots
  knots = quantile(X, probs=c(0.25, 0.5, 0.75))

  # make data frame of data and prediction
  data = data.frame(
    X = X,
    X2 = X^2,
    X3 = X^3,
    h1 = h(X, 1, knots),
    h2 = h(X, 2, knots),
    h3 = h(X, 3, knots)
  )
  data_prediction = data.frame(
    X = 2,
    X2 = 2^2,
    X3 = 2^3,
    h1 = h(2, 1, knots),
    h2 = h(2, 2, knots),
    h3 = h(2, 3, knots)
  )

  # make model and store result
  f_hat = lm(Y ~ X + X2 + X3 + h1 + h2 + h3, data=data)
  f_hat_0[i] = predict(f_hat, data_prediction)
}
f_hat_0
```

Part d

Estimate bias of $\hat{f}(x_0)$ under the given model by the formula: bias $\approx Ave(\hat{f}(x_0)) - f(x_0)$ using the predicted values. Estimate the variance of $\hat{f}(x_0)$ by the formula: variance $= Var(\hat{f}(x_0))$ using the predicted values.

```
bias_f = mean(f_hat_0)  - 2^2 # true value is 4 since model is x^2 so 2^2=4
bias_f

variance_f = var(f_hat_0)
variance_f
```

Part e

If you let $\hat{f}(x) = \beta_0 + \beta_1 x$, then what do you expect regarding the resulted bias and variance of $\hat{f}(x_0)$ relative to the results obtained above? Answer without doing calculations.

If we now make the model to be a linear one it cannot capture the shape (or curvature) of x^2 as we force the shape of the model to be linear, because of this it makes sense that our bias will increase. Also, our variance will now decrease since a 2-parameter model is far more stable (with less parameters being estimated) than the previous 7-parameter spline (which makes sense since bias and variance are inversely related). Overall, the simple linear model trades higher bias for lower variance, compared to the original more complex model.

11.0.2 K Means Clustering

You are given a data set with two features X_1 and X_2. There are six observations in the data that are given below:

$$(3.5, -4.5),\ (4, -6),\ (3.5, -3),\ (0.5, 0),\ (1.5, 0),\ (1, 1)$$

Part a

Let $K = 2$. Randomly assign three observations from the data into cluster 1 and the rest observations into cluster 2 (using R). Find the centroid of each cluster.

```r
set.seed(2025)
x <- matrix(c( 3.5, -4.5,
               4.0, -6.0,
               3.5, -3.0,
               0.5, 0.0,
               1.5, 0.0,
               1.0, 1.0),
            ncol=2, byrow=TRUE)

# random assignment of 3 observations to cluster 1, 3 to cluster 2:
clust0 = sample(rep(1:2, each=3))
C1 = which(clust0==1)
C2 = which(clust0==2)

# compute centroids
centroid = function(idx) colMeans(x[idx, ])
c1 = centroid(C1)
c2 = centroid(C2)

# gives observations in each cluster
C1; C2;

# gives new centroids of clusters
c1; c2 # c1 = (1.000, 3.333), c2 = (3.667, -4.500)
```

$C_1 = \{4, 5, 6\}, C_2 = \{1, 2, 3\}$. Thus the centroids are $\bar{x}_1^{(1)} = \frac{1}{3}\sum_{i \in C_1} x_i = \left(\frac{3.5+4+3.5}{3},\ \frac{-4.5-6-3}{3}\right) = (3.667, -4.500)$ and $\bar{x}_2^{(1)} = \frac{1}{3}\sum_{i \in C_2} x_i = \left(\frac{0.5+1.5+1}{3},\ \frac{0+0+1}{3}\right) = (1.000, 0.333)$.

Part b

Using the Euclidean distance, perform the K-means clustering algorithm manually. Note that you need to type your answers step by step and state your final results (the two clusters) clearly.

Compute

$$d_{ik} = ||x_i - \bar{x}_k^{(1)}||_2 = \sqrt{(x_{i1} - \bar{x}_{k1})^2 + (x_{i2} - \bar{x}_{k2})^2}$$

The table below shows d_{i1} and d_{i2} for each i:

i	x_i	$d_{i1} = d(x_i, \bar{x}_1)$	$d_{i2} = d(x_i, \bar{x}_2)$	$\arg\min_k d_{ik}$
1	$(3.5, -4.5)$	5.4410	0.1667	2
2	$(4, -6)$	7.0080	1.5360	2
3	$(3.5, -3)$	4.1670	1.5093	2
4	$(0.5, 0)$	0.6009	5.5040	1
5	$(1.5, 0)$	0.6009	4.9944	1
6	$(1, 1)$	0.6667	6.1133	1

Because each cluster membership did not change, the algorithm terminates after a single iteration.

11.0.3 Automate K-Mean Clustering By Hand in R

```r
#' Perform and fully annotated, by-hand style k-means
#'
#' @param x      numeric matrix or data frame (rows = observations, cols = features)
#' @param k      number of clusters
#' @param init   optional vector of length k giving initial center row-indices
#' @param max.iter  maximum iterations
#' @return invisibly, kmeans object; prints every formula & result
explain_kmeans_detailed <- function(x, k, init = NULL, max.iter = 10) {
  # ensure matrix
  if (is.data.frame(x)) x <- as.matrix(x)
  n <- nrow(x); p <- ncol(x)
  labs <- if (!is.null(rownames(x))) rownames(x) else as.character(1:n)

  # 1) initialize centers
  if (is.null(init)) {
    set.seed(1)
    init <- sample(n, k)
  }
  centers <- x[init, , drop = FALSE]
  cat("=== Initial centers selected ===\n")
  for (j in seq_len(k)) {
    cat(sprintf("C%d^(0) = point %s = (%s)\n",
                j,
                labs[init[j]],
                paste(round(centers[j,],2), collapse=", ")))
  }

  # prepare
  clusters <- integer(n)

  for (it in seq_len(max.iter)) {
    cat("\n=== Iteration", it, "===\n")

    # 2) assign points
    new.cl <- integer(n)
    for (i in seq_len(n)) {
      cat(sprintf("\nPoint %s = (%s)\n",
                  labs[i],
```

```
                  paste(x[i,], collapse=", ")))
      # compute each distance
      dists <- numeric(k)
      for (j in seq_len(k)) {
        # build formula string
        terms <- character(p)
        for (m in seq_len(p)) {
          terms[m] <- sprintf("(%s - %s)^2",
                              x[i,m], round(centers[j,m],2))
        }
        formula <- paste(terms, collapse=" + ")
        dval <- sqrt(sum((x[i,] - centers[j,])^2))
        dists[j] <- dval
        cat(sprintf("  d(%s, C%d) = sqrt(%s) = %.2f\n",
                    labs[i], j, formula, round(dval,2)))
      }
      # decide
      assign <- which.min(dists)
      new.cl[i] <- assign
      cat(sprintf("-> assign to cluster %d (smallest d = %.2f)\n",
                  assign, round(dists[assign],2)))
    }

    # print summary
    cat("\nCluster assignments:\n")
    print(data.frame(Point=labs, Cluster=new.cl))

    # convergence?
    if (all(new.cl == clusters)) {
      cat("\nNo change in assignments; converged.\n")
      break
    }
    clusters <- new.cl

    # 3) update centers
    cat("\nUpdating centers:\n")
    for (j in seq_len(k)) {
      mem <- which(clusters == j)
      pts <- paste(sprintf("(%s)",
                           apply(x[mem,,drop=FALSE], 1,
                                 function(row) paste(row, collapse=","))),
                   collapse=" + ")
      nmem <- length(mem)
      newc <- colMeans(x[mem, , drop=FALSE])
      cat(sprintf("C%d^(%d) = (1/%d) * [%s] = (%s)\n",
                  j, it,
                  nmem,
                  pts,
                  paste(round(newc,2), collapse=", ")))
      centers[j,] <- newc
    }
  }

  cat("\nFinished k-means with k =", k, "\n")
  invisible(kmeans(x, centers, iter.max = it))
}

######## EXAMPLE APPLICATION ########
```

```r
# Application of example 11.0.2 above

dat = matrix(
  c( 3.5, -4.5,
     4, -6,
     3.5, -3,
     0.5, 0,
     1.5, 0,
     1, 1),
  byrow=TRUE, ncol=2
)
rownames(dat) = paste0("P", 1:6)
colnames(dat) = c("X1", "X2")


km = explain_kmeans_detailed(dat, k=2, init=c(1,3))
```

11.0.4 Automate Hierarchical Clustering By Hand in R (given coordinates)

```r
#' Fully-annotated, by-hand style hierarchical clustering, with optional plot
#'
#' @param x       numeric matrix or data frame (rows = obs, cols = features)
#' @param labels  optional vector of names for the observations
#' @param method  one of "single", "complete", "average", or "ward"
#' @param plot    logical; if TRUE, draw the dendrogram at the end
#' @return invisibly, the hclust object; prints every formula & merge
explain_hclust_detailed <- function(
  x,
  labels = NULL,
  method = c("complete","single","average","ward"),
  plot = FALSE
) {
  if (is.data.frame(x)) x <- as.matrix(x)
  method <- match.arg(method)
  n <- nrow(x); p <- ncol(x)
  labs <- if (!is.null(labels)) labels else as.character(1:n)
  rownames(x) <- labs

  # 1) Initial pairwise distances
  cat("=== Initial pairwise distances ===\n")
  for (i in 1:(n-1)) {
    for (j in (i+1):n) {
      terms <- sprintf("(%s - %s)^2", x[i,], x[j,])
      dij   <- sqrt(sum((x[i,] - x[j,])^2))
      cat(sprintf(
        "d(%s,%s) = sqrt(%s) = %.2f\n",
        labs[i], labs[j],
        paste(terms, collapse=" + "),
        dij
      ))
    }
  }

  # 2) Initialize clusters
  clusters <- setNames(
    lapply(seq_len(n), function(i) i),
```

```
  labs
)
step <- 1

# 3) Merge until one cluster remains
while (length(clusters) > 1) {
  cat(sprintf("\n=== Step %d: %s linkage distances ===\n",
              step, method))

  cnames <- names(clusters)
  m <- length(clusters)
  D <- matrix(Inf, m, m, dimnames = list(cnames, cnames))

  # compute inter-cluster distances
  for (a in seq_len(m-1)) {
    for (b in (a+1):m) {
      A <- cnames[a]; B <- cnames[b]
      ptsA <- clusters[[A]]; ptsB <- clusters[[B]]
      pd   <- outer(ptsA, ptsB, Vectorize(function(i,j) {
        sqrt(sum((x[i,] - x[j,])^2))
      }))

      cat(sprintf("\nDistances between {%s} and {%s}:\n", A, B))
      for (i in ptsA) for (j in ptsB) {
        terms <- sprintf("(%s - %s)^2", x[i,], x[j,])
        dij   <- sqrt(sum((x[i,] - x[j,])^2))
        cat(sprintf(
          "  d(%s,%s) = sqrt(%s) = %.2f\n",
          labs[i], labs[j],
          paste(terms, collapse=" + "),
          dij
        ))
      }

      dc <- switch(method,
        single   = min(pd),
        complete = max(pd),
        average  = mean(pd),
        ward     = {
          mA   <- colMeans(x[ptsA,,drop=FALSE])
          mB   <- colMeans(x[ptsB,,drop=FALSE])
          coef <- length(ptsA)*length(ptsB)/(length(ptsA)+length(ptsB))
          coef * sum((mA - mB)^2)
        }
      )

      if (method=="single") {
        cat(sprintf("  single linkage -> min = %.2f\n", dc))
      } else if (method=="complete") {
        cat(sprintf("  complete linkage -> max = %.2f\n", dc))
      } else if (method=="average") {
        cat(sprintf("  average linkage -> mean = %.2f\n", dc))
      } else {
        cat(sprintf("  Ward's criterion = %.2f\n", dc))
      }

      D[a,b] <- D[b,a] <- dc
    }
```

```
    }

    # pick clusters to merge
    idx <- which(D == min(D), arr.ind = TRUE)[1,]
    A <- rownames(D)[idx[1]]; B <- colnames(D)[idx[2]]
    dmin <- D[idx[1], idx[2]]
    cat(sprintf("\n-> Merging {%s} and {%s} at distance = %.2f\n", A, B, dmin))

    # update cluster list
    newName  <- paste0("{", A, ",", B, "}")
    members  <- c(clusters[[A]], clusters[[B]])
    clusters <- clusters[! names(clusters) %in% c(A,B)]
    clusters[[newName]] <- members

    cat("Clusters now:", paste(names(clusters), collapse=" | "), "\n")
    step <- step + 1
  }

  cat(sprintf("\nFinished hierarchical clustering (method = %s).\n", method))

  # 4) build and optionally plot dendrogram
  hc <- hclust(dist(x), method = method)
  if (plot) {
    plot(
      hc,
      main = paste0("Dendrogram (", method, " linkage)"),
      xlab = "",
      sub  = "",
      labels = labs
    )
  }

  invisible(hc)
}

######## EXAMPLE APPLICATION ########

dat <- matrix(c(
  1,  2,   # P1
  2,  1,   # P2
 -1, -2,   # P3
 -2, -1    # P4
), byrow=TRUE, ncol=2)
rownames(dat) <- paste0("P",1:4)
colnames(dat) <- c("X1","X2")

# Run complete-linkage with printed steps and plot the dendrogram:
hc <- explain_hclust_detailed(
  dat,
  labels = rownames(dat),
  method = "complete",
  plot = TRUE
)
```

11.0.5 Automate Hierarchical Clustering By Hand in R (given dissimilarity matrix)

```r
#' Fully-annotated, by-hand style hierarchical clustering *from* a
#' dissimilarity matrix, with optional plot
#'
#' @param diss   a dist object or symmetric matrix of pairwise D_ij
#' @param labels optional vector of length n naming the rows/cols of diss
#' @param method one of "single","complete","average","ward"
#' @param plot   logical; if TRUE, draws the dendrogram at the end
#' @return invisibly, the hclust object; prints every formula & merge
explain_hclust_from_diss <- function(diss,
                                     labels   = NULL,
                                     method   = c("average","single","complete","ward"),
                                     plot     = FALSE) {
  # 1) turn it into a matrix we can index
  Dmat   <- if (inherits(diss, "dist")) as.matrix(diss) else diss
  method <- match.arg(method)
  n      <- nrow(Dmat)
  labs   <- if (is.null(labels)) rownames(Dmat) else labels
  rownames(Dmat) <- colnames(Dmat) <- labs

  # 2) print initial pairwise dissimilarities
  cat("=== Initial pairwise dissimilarities ===\n")
  for (i in 1:(n-1)) {
    for (j in (i+1):n) {
      cat(sprintf("d(%s,%s) = %.2f\n", labs[i], labs[j], Dmat[i,j]))
    }
  }

  # 3) initialize clusters: each label in its own cluster
  clusters <- setNames(lapply(seq_len(n), function(i) i), labs)
  step <- 1

  # 4) repeat until one cluster remains
  while (length(clusters) > 1) {
    cat(sprintf("\n=== Step %d: %s-linkage ===\n", step, method))

    cnames <- names(clusters)
    m      <- length(clusters)
    # will hold the average-linkage distances
    Cmat   <- matrix(Inf, m, m, dimnames = list(cnames, cnames))

    # compute every inter-cluster distance
    for (a in seq_len(m-1)) {
      for (b in (a+1):m) {
        A    <- cnames[a]
        B    <- cnames[b]
        ptsA <- clusters[[A]]
        ptsB <- clusters[[B]]
        # extract the D_ij between every i in A and j in B
        subD <- Dmat[ptsA, ptsB, drop=FALSE]
        # print all of those
        cat(sprintf("\nDistances between {%s} and {%s}:\n", A, B))
        for (i in ptsA) for (j in ptsB) {
          cat(sprintf("  d(%s,%s) = %.2f\n", labs[i], labs[j], Dmat[i,j]))
        }
        # now aggregate by the chosen method
        dc <- switch(method,
          single   = min(subD),
```

```
          complete = max(subD),
          average  = mean(subD),
          ward     = {
            # Ward's criterion works on sums of squares; for true "dissimilarities" this
            # is often pre-computed, but here we'll do the approximate average
            mA   <- mean(as.numeric(subD))
            mA    # fallback if you really need ward
          }
        )
        cat(sprintf("  %s-linkage -> %.2f\n",
                    method, dc))
        Cmat[a,b] <- Cmat[b,a] <- dc
      }
    }

    # pick the pair to merge (smallest aggregate)
    idx <- which(Cmat == min(Cmat), arr.ind=TRUE)[1,]
    A   <- rownames(Cmat)[idx[1]]
    B   <- colnames(Cmat)[idx[2]]
    dAB <- Cmat[idx[1], idx[2]]
    cat(sprintf("\n-> merging {%s} & {%s} at height = %.2f\n", A, B, dAB))

    # update clusters: remove A & B, add union
    newName <- paste0("{", A, ",", B, "}")
    members <- c(clusters[[A]], clusters[[B]])
    clusters <- clusters[!names(clusters) %in% c(A,B)]
    clusters[[newName]] <- members

    cat("Clusters now:", paste(names(clusters), collapse=" | "), "\n")
    step <- step + 1
  }

  # 5) finally build & optionally plot the dendrogram
  hc <- hclust(as.dist(Dmat), method=method)
  if (plot) {
    plot(hc,
         main   = paste0(method,"-linkage dendrogram")5,
         xlab   = "",
         sub    = "",
         labels = labs)
  }

  invisible(hc)
}

######## EXAMPLE APPLICATION #######
Dmat <- matrix(
  c(
     0, 0.2, 0.4, 0.6,
     0.2, 0, 0.1, 0.5,
     0.4, 0.1, 0, 0.3,
     0.6, 0.5, 0.3, 0
  ),
  nrow = 4, byrow = TRUE
)
rownames(Dmat) <- colnames(Dmat) <- c("A","B","C","D")
diss <- as.dist(Dmat)
```

```
# now run the explainer in "average" mode *and* draw the dendrogram:
hc <- explain_hclust_from_diss(
        diss,
        labels = c("A","B","C","D"),
        method = "average",
        plot   = TRUE
      )
```

A Appendix: Summary of Statistical Learning Models

Refer to next page for summary.

Has Advantages, Disadvantages/Drawbacks, and Additional Notes for each model learnt.

Summary of Statistical Learning Methods

Topic	Advantages	Disadvantages/drawbacks	Additional Notes
Simple Linear Regression	• Simple to apply and interpret • Has many explicit results • Doesn't need a lot of data to fit • LOOCV result is easy to calculate	• One of the most restrictive models • Usually has low predicting power	• High bias, low variance
Multiple Linear Regression	• Easy to explain and to fit • Easy for inference tasks • LOOCV result is easy to calculate	• Variable selection can be a challenging task • Low predicting power	• Categorical variables • Interaction effect • High bias, low variance • Outliers, high leverage points • Collinearity
Best Subset Selection	• Can find the best subset of predictors based on the given criterion	• Can be computationally expensive when p is large	• It is better than stepwise selections if p is small
Stepwise Selection	• More computationally efficient	• Not working properly for $n \leq p$ • Not guarantee the best selection of variables	•
Ridge Regression	• Can reduce the variance in fitted model for a carefully chosen tuning parameter λ • Computationally efficient for selected λ values	• Can't be used as a feature selection approach • Finding the optimal tuning parameter can be a challenging task	• Cross validation is useful in selecting the tuning parameter λ • λ increases, higher bias, lower variance • works well when most of the predictors are related to the response variable
Lasso Approach	• Can reduce the variance in fitted model for a carefully chosen tuning parameter λ • Can achieve a feature selection goal	• Finding the optimal tuning parameter can be a challenging task	• Cross validation is useful in selecting the tuning parameter λ • λ increases, higher bias, lower variance

			• works well when very few of the predictors are related to the response variable
Principal Component Analysis (PCA)	• Can help to reduce the dimensions in modelling.	• The number of PCs to include needs to be determined. • It is not an approach to select features.	• The standardisation of predictors is usually needed. • CV is useful when determining the number of PCs.
Partial Least Squares	• Can help to reduce the dimensions in modelling. • The response variable is involved in transforming the predictors.	• The number of transformed variables to include in the model needs to be determined. • It is not an approach to select features.	• CV is useful when determining the number of PCs. •
Validation Set Approach	• Can estimate the test MSE or test error rate when test data are not available • It is a general approach	• The results can vary significantly due to randomness in data split	• Has the highest bias in the CV approaches
LOOCV Approach	• It is a widely applicable approach • Can estimate the test MSE/test error rate when test data are not available • The results are stable as no randomness in data split • Have shortcuts in certain cases, eg MLR, polynomial regression, smoothing splines	• Can be computationally expensive • The estimated test MSE or test error rate can be inaccurate	• Has the lowest bias in the CV approaches
***K*-fold CV Approach**	• It is a widely applicable approach • Can estimate the test MSE/test error rate when test data are not available • The results are more stable than the validation set approach	• The estimated test MSE or test error rate can be inaccurate	• Has lower variance than LOOCV method • Usually $K = 5$ or 10

	• More efficient than the LOOCV approach		
Bootstrap Approach	• It is a widely applicable approach • Can estimate the test MSE/test error rate when test data are not available	• The estimated test MSE or test error rate can be inaccurate	• The original data quality and number of bootstrap samples to drawn do matter.
Polynomial Regression	• Easy to fit using the least squares method and the result is smooth • Can produce extremely non-linear fit • LOOCV result is easy to calculate	• The model can become overly flexible for if *d* is large	• Usually *d* is 3 or 4.
Regression Splines	• Combining low-degree polynomials at the chosen knots with constraints gives a smooth and non-linear fit • Variations of results in certain ranges can be lowered down through additional constraints (natural spline) • Can produce more stable results than high-degree polynomials • The flexibility of results can be adjusted through number of knots	• The number and location of knots can be hard to optimise.	
Smoothing Splines	• The results are just natural cubic splines • The tuning parameter λ controls the flexibility of results • LOOCV result is easy to calculate	• The number of knots involved in modelling can be very large (the number of unique predictor values) • The tuning parameter λ needs to be optimised	• The effective degrees of freedom play an important role in model performance.
Local Regression	• Adapts well to bias problems at boundaries and in regions of high curvature.	• Need to find optimal span/bandwidth values. • May need to optimise the parametric form for the local models.	• Quadratic or cubic functions are common choices for the local models.

	• Easy to understand and interpret. • Methods have been developed that provide fast computation for one or more independent variables. • Because of its simplicity, can be tailored to work for many different distributional assumptions. • Having a local model enables derivation of response adaptive methods for span value and polynomial order selection in a straightforward manner.	• Need to define a suitable weight function.	• The tricube weight function is a commonly used one.
GAMs	• Allows us to fit a non-linear model to each predictor. • Retains a nice additive structure within the model. • Can deal with different distributional assumptions. • The individual effect of each predictor on the response variable can be assessed. • The smoothness of each building block can be summarized via degrees of freedom.	• The method is restricted to be additive. • Interaction effects are hard to incorporate (not impossible). • The fitted models often don't have explicit expressions.	• It can combine parametric and non-parametric methods.
Logistic regression	• Provides a proper model to study the conditional probability P(Y=1|X). • It retains the linear framework for predictors.	• Logistic regression works better with bivariate response variables. • When the classes are well separated, the logistic regression results can be unstable.	• It results in a linear decision boundary.

LDA	• It works well when *n* is small and the predictors are approximately normal in each class. • It can handle multi-class response variable well.	• The normal assumption is key to this method. • The predictors are assumed to have same variances among different classes.	• It results in a linear decision boundary. • When the true decision boundary is linear or close to linear, LR and LDA tend to work better, in particular for small data set. If normality is absent, then LR outperforms LDA.
QDA	• It works well when the predictors are approximately normal in each class when the covariance matrix vary among classes.	• It works well when the true decision boundary is close to quadratic. • It wouldn't work well if *n* is too small.	• It results in a quadratic decision boundary. • Though not as flexible as KNN, QDA can perform better in the presence of a limited number of training observations, as it does make some assumptions about the form of the decision boundary. • When the true decision boundary is quadratic or moderate non-linear, QDA would outperform LR and LDA.
KNN	• It is a non-parametric method that doesn't need any distributional assumptions. • The algorithm is simple and easy to implement. • The algorithm is versatile. It can be used for classification and regression.	• Choosing the right K value is critical. It can be found using CV. • The algorithm gets significantly slower as *n* and/or *p* increase. • No inference results for KNN.	• The KNN-CV is the most robust method. It performs relatively well in most cases. It outperforms the rest when the true decision boundary is very non-linear and the normality assumption is not met.

Decision trees	• Trees are very easy to explain. • Some people believe that decision trees more closely mirror human decision-making than do the other regression and classification approaches. • Trees can be displayed graphically, and are easily interpreted even by a non-expert. • Trees can easily handle qualitative predictors without the need to create dummy variables.	• Trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches. • Trees can be very non-robust. A small change in the data can cause a large change in the final estimated tree.	• Tree pruning is useful in improving the prediction accuracy.
Maximal margin classifier	• The maximal margin hyperplane is a natural choice when separating hyperplanes exist in the given data. • It is effective in high-dimensional spaces. • It is usually memory efficient as only based on support vectors.	• The classes in the given data must be separable, i.e. separating hyperplanes must exist. • The maximal margin hyperplane is extremely sensitive to a change in a single observation. • It often overfits the training data when p is large. • It doesn't directly provide probability estimates.	• The maximal margin hyperplane depends directly on only a small subset of the observations, i.e. the support vectors. • The training error rate is always 0.
Support vector classifier	• It adopts soft margins that leads to a greater robustness to individual observations and better classification of most training observations. • The level of flexibility of the classifier can be adjusted through a tuning parameter C. • It is effective in high-dimensional spaces.	• It often has poor performance for data with non-linear decision boundaries. • The tuning parameter C needs to be optimised to get the best prediction performance. • It doesn't work well when the data set contains more noise. • It doesn't directly provide probability estimates.	• The tuning parameter C controls the bias-variance trade-off of the support vector classifier. Small C values lead to low bias but high variance, and vice versa.

	• It is usually memory efficient as only based on support vectors.		
Support vector machine (SVM)	• The SVM enlarges the feature space used by the support vector classifier in an efficient way. • It can classify data with non-linear decision boundaries. • It is effective in high-dimensional spaces. • It is usually memory efficient as only based on support vectors.	• Selecting the right type of kernel functions with appropriate parameters needs extra work. • It is computationally expensive when we have a large data set. • It doesn't work well when the data set contains more noise. • It doesn't directly provide probability estimates.	
PCA	• It can decorrelate the original features in the data. • Can help with data visualization.	• The number of PC's to keep is hard to determine. • Finding intuitive interpretations for the PC's can be difficult.	• Standardization is something to consider.
K-means clustering	• The algorithm is simple and easy to implement. • It generates the number of clusters we need.	• The results are not stable for single run. Multiple attempts are required. • It can only generate the number of clusters we specify.	•
Hierarchical clustering	• It has an upside-down tree presentation, i.e. a dendrogram. • A single dendrogram can produce any number of clusters between 1 and *n*. • The clusters generated by this method have a nesting relationship. • We don't need to guess the number of clusters before building the dendrogram.	• If the true clusters in the data don't have a nesting relationship, then Hierarchical clustering won't work properly. • The dissimilarity measure and linkage need to be selected carefully.	• Whether to scale the data or not is something to mind. • Small decisions could have big consequences.

B Appendix: Bias-Variance Trade-off Discussions

1. **Simple/Multiple Linear Regression**
   Complexity control: Number of predictors (p) and interaction/polynomial terms.
   Bias–Variance:
   - *Few predictors* $\rightarrow$ high bias, low variance.
   - *Many predictors* $\rightarrow$ low bias, high variance.

   Discussion: Adding a predictor that truly explains variation reduces bias by capturing more signal, but if it's noisy or collinear it inflates variance by making coefficient estimates unstable. Dropping predictors smooths the model (lower variance) at the cost of omitting relationships (higher bias).
2. **Polynomial Regression**
   Complexity control: Polynomial degree (d).
   Bias–Variance:
   - *Small* d (e.g. 1–2) $\rightarrow$ high bias, low variance.
   - *Large* $d \rightarrow$ low bias, high variance.

   Discussion: Higher degree allows the curve to bend and follow data more closely (lower bias) but can oscillate wildly at the edges (higher variance). Keeping d small forces a smooth, stable fit.
3. **Best Subset & Stepwise Selection**
   Complexity control: Number of selected variables (k).
   Bias–Variance:
   - *Small* $k \rightarrow$ high bias, low variance.
   - *Large* $k \rightarrow$ low bias, high variance.

   Discussion: Allowing more variables improves training fit (lower bias) but risks including noise variables, making the model wobble on new data (higher variance). Greedy stepwise search can amplify variance if small data changes alter the selection path.
4. **Ridge Regression**
   Complexity control: Penalty $\lambda \geq 0$.
   Bias–Variance:
   - *Large* $\lambda \rightarrow$ higher bias, lower variance.
   - *Small* $\lambda \rightarrow$ lower bias, higher variance.

   Discussion: Larger λ uniformly shrinks coefficients toward zero—simplifying and stabilizing estimates (low variance) but potentially underrepresenting true effects (higher bias). As $\lambda \rightarrow 0$, the fit approaches OLS (low bias, high variance).
5. **Lasso**
   Complexity control: Penalty λ.
   Bias–Variance:
   - *Large* $\lambda \rightarrow$ higher bias, lower variance.
   - *Small* $\lambda \rightarrow$ lower bias, higher variance.

   Discussion: Like ridge, but lasso can set coefficients exactly to zero—aggressively reducing variance. Too much shrinkage drops true predictors (high bias); too little leaves many small coefficients (high variance).
6. **Principal Component Analysis (PCA)**
   Complexity control: Number of components retained (m).
   Bias–Variance:
   - *Small* $m \rightarrow$ high bias, low variance.
   - *Large* $m \rightarrow$ low bias, high variance.

   Discussion: Keeping only top PCs filters out low-variance (often noisy) directions—reducing variance but discarding signal (higher bias). Retaining more PCs captures finer structure (lower bias) at the cost of reintroducing noise (higher variance).

7. **Partial Least Squares (PLS)**
   Complexity control: Number of latent factors.
   Bias–Variance:
   - *Few factors* → high bias, low variance.
   - *Many factors* → low bias, high variance.

   Discussion: Each additional factor models more covariance between X and Y, lowering bias but eventually fitting noise (higher variance). Stopping early yields a stable but coarse regression.
8. **Validation Set Approach**
   Complexity control: n/a.
   Bias–Variance:
   - *Estimate* → high bias, high variance.

   Discussion: Holding out part of data inflates test-error estimates (overestimation bias) and results vary greatly with different random splits (high variance).
9. **LOOCV**
   Complexity control: n/a.
   Bias–Variance:
   - *Estimate* → low bias, high variance.

   Discussion: Training on $n-1$ cases yields nearly unbiased test-error estimates, but errors from each fold are highly correlated, making the estimate noisy.
10. **K-Fold CV**
    Complexity control: Number of folds (K).
    Bias–Variance:
    - *Small K* (e.g. 5) → higher bias, lower variance.
    - *Large K* (e.g. 10) → lower bias, higher variance.

    Discussion: Fewer folds mean larger validation sets (more pessimistic but stable), while more folds yield more accurate yet noisier error estimates.
11. **Bootstrap**
    Complexity control: Number of resamples (B).
    Bias–Variance:
    - *Small B* → high variance in the estimate.
    - *Large B* → lower variance (bias depends on estimator).

    Discussion: More bootstrap samples smooth out Monte Carlo noise (lower variance), but bootstrap-based error metrics (e.g. .632 estimator) carry intrinsic bias depending on model complexity.
12. **Regression Splines**
    Complexity control: Number of knots & spline degree.
    Bias–Variance:
    - *Few knots* → high bias, low variance.
    - *Many knots* → low bias, high variance.

    Discussion: More knots allow the fit to bend at more locations—capturing local trends (lower bias) but creating jagged segments (higher variance). Sparse knots force smoother global curves.
13. **Smoothing Splines**
    Complexity control: Penalty λ.
    Bias–Variance:
    - *Large λ* → high bias, low variance.
    - *Small λ* → low bias, high variance.

    Discussion: Strong penalty forces near–linear fit (stable but underfitting), while weak penalty yields highly flexible curves that can chase noise.
14. **Local Regression (LOESS)**

Complexity control: Span (α).
Bias–Variance:
- *Large* $\alpha \to$ high bias, low variance.
- *Small* $\alpha \to$ low bias, high variance.

Discussion: A wide neighborhood smooths over noise (stable but coarse), whereas a narrow neighborhood adapts to fine structure (low bias) but is sensitive to local fluctuations (high variance).

15. **Generalized Additive Models (GAMs)**
Complexity control: Degrees of freedom or smoothing penalty.
Bias–Variance:
- *Heavy penalty* $\to$ high bias, low variance.
- *Light penalty* $\to$ low bias, high variance.

Discussion: Tighter smoothing makes each additive term change slowly (stable but underfitting), while looser smoothing captures subtler curves at the risk$_o f over fitting$.

16. **Logistic Regression**
Complexity control: Predictors/interactions included; penalties.
Bias–Variance:
- *Fewer terms/stronger penalty* $\to$ high bias, low variance.
- *More terms/weaker penalty* $\to$ low bias, high variance.

Discussion: As in linear regression, adding features improves fit but may amplify coefficient uncertainty; penalties shrink estimates to stabilize the model.

17. **Linear Discriminant Analysis (LDA)**
Complexity control: n/a (shared covariance assumption).
Bias–Variance:
- inherently high bias, low variance.

Discussion: The common–covariance assumption oversimplifies class boundaries (bias) but yields very stable discriminant rules—ideal with small class samples.

18. **Quadratic Discriminant Analysis (QDA)**
Complexity control: n/a (separate covariances).
Bias–Variance:
- lower bias than LDA, higher variance.

Discussion: Separate covariance estimates allow curved boundaries (lower bias) but require estimating many parameters, which increases variance in small samples.

19. **K-Nearest Neighbors (KNN)**
Complexity control: Number of neighbors (k).
Bias–Variance:
- *Large* $k \to$ high bias, low variance.
- *Small* $k \to$ low bias, high variance.

Discussion: Averaging over many neighbors smooths the boundary (stable but coarse), while few neighbors yield a jagged boundary—sensitive to noise.

20. **Decision Trees**
Complexity control: Tree depth, minimum bucket size, pruning.
Bias–Variance:
- *Shallow/pruned* $\to$ high bias, low variance.
- *Deep/unpruned* $\to$ low bias, high variance.

Discussion: Strong pruning forces broad splits (stable but underfitting), while deep trees capture intricate patterns (low bias) but are unstable: small data changes reorder branches.

21. **Maximal Margin Classifier**
Complexity control: n/a (hard margin).
Bias–Variance:
- low bias, high variance.

Discussion: Demanding zero training error fits a boundary that perfectly separates classes (minimal bias) but is extremely sensitive to noise (high variance).

22. **Support Vector Classifier (Soft-Margin SVM)**
    Complexity control: Cost parameter (C).
    Bias–Variance:
    - *Small C* $\rightarrow$ high bias, low variance.
    - *Large C* $\rightarrow$ low bias, high variance.

    Discussion: Lowering C widens the margin and tolerates misclassifications (more bias, stable), while raising C tightens the margin to fit data closely (less bias, sensitive to outliers).
23. **SVM with Kernel**
    Complexity control: Kernel parameters (e.g. γ, degree) plus C.
    Bias–Variance:
    - *Rich kernel + large C* $\rightarrow$ low bias, high variance.
    - *Smooth kernel + small C* $\rightarrow$ high bias, low variance.

    Discussion: A flexible kernel carves intricate boundaries (low bias) but can overfit training peculiarities (high variance). Smoothing the kernel or penalizing errors more heavily reins in variance at the cost of underfitting.
24. **K-Means Clustering**
    Complexity control: Number of clusters (K).
    Bias–Variance:
    - *Small K* $\rightarrow$ high bias, low variance.
    - *Large K* $\rightarrow$ low bias, high variance.

    Discussion: Fewer clusters impose broad groupings (may miss substructure but stable), whereas more clusters capture finer distinctions (lower bias) but depend heavily on initialization (higher variance).
25. **Hierarchical Clustering**
    Complexity control: Cut height / number of clusters.
    Bias–Variance:
    - *Few clusters* $\rightarrow$ high bias, low variance.
    - *Many clusters* $\rightarrow$ low bias, high variance.

    Discussion: Cutting the dendrogram high yields coarse clusters that generalize stably (low variance) but may lump dissimilar points (high bias). Cutting low produces many small clusters that fit local patterns (low bias) but can be unstable.

C Appendix: Example R Applications

Refer to next page for summary.

C.1 AI Prac Exam 1

The following is some example questions completed on `ISLR` datasets in R.

AI Prac Exam #1

Omar

2025-06-24

1 - Subset Selection & Regression

Dataset is Auto.

Part a - Best subset

```
x1 = ISLR2::Auto
x1 = na.omit(x1)
x1 = subset(x1, select=-name)
library(leaps)

regfit = regsubsets(mpg ~., data=x1)
preds.summ = summary(regfit)
preds.summ
```

```
## Subset selection object
## Call: regsubsets.formula(mpg ~ ., data = x1)
## 7 Variables  (and intercept)
##              Forced in Forced out
## cylinders        FALSE      FALSE
## displacement     FALSE      FALSE
## horsepower       FALSE      FALSE
## weight           FALSE      FALSE
## acceleration     FALSE      FALSE
## year             FALSE      FALSE
## origin           FALSE      FALSE
## 1 subsets of each size up to 7
## Selection Algorithm: exhaustive
##          cylinders displacement horsepower weight acceleration year origin
## 1  ( 1 ) " "       " "          " "        "*"    " "          " "  " "
## 2  ( 1 ) " "       " "          " "        "*"    " "          "*"  " "
## 3  ( 1 ) " "       " "          " "        "*"    " "          "*"  "*"
## 4  ( 1 ) " "       "*"          " "        "*"    " "          "*"  "*"
## 5  ( 1 ) " "       "*"          "*"        "*"    " "          "*"  "*"
## 6  ( 1 ) "*"       "*"          "*"        "*"    " "          "*"  "*"
## 7  ( 1 ) "*"       "*"          "*"        "*"    "*"          "*"  "*"
```

```r
# extracting lowest bic
min.bic = which.min(preds.summ$bic)
preds.summ$bic[min.bic]
```

```
## [1] -642.8063
```

```r
# best model according to lowest bic
preds.summ$which[min.bic, ]
```

```
##  (Intercept)    cylinders displacement   horsepower       weight acceleration
##         TRUE        FALSE        FALSE        FALSE         TRUE        FALSE
##         year       origin
##         TRUE         TRUE
```

Part b - fit chosen model

```r
# Best model includes weight, year and origin
best.model = lm(mpg ~ weight + year + origin, data=x1)
summary(best.model)
```

```
##
## Call:
## lm(formula = mpg ~ weight + year + origin, data = x1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.9440 -2.0948 -0.0389  1.7255 13.2722
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.805e+01  4.001e+00  -4.510 8.60e-06 ***
## weight      -5.994e-03  2.541e-04 -23.588  < 2e-16 ***
## year         7.571e-01  4.832e-02  15.668  < 2e-16 ***
## origin       1.150e+00  2.591e-01   4.439 1.18e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.348 on 388 degrees of freedom
## Multiple R-squared:  0.8175, Adjusted R-squared:  0.816
## F-statistic: 579.2 on 3 and 388 DF,  p-value: < 2.2e-16
```

All predictors are significant of this best model. The adjusted R^2 of 0.816 shows the proportion of variance explained, penalising extra predictors. The residual standard error gives the typical size of prediction errors on the training set.

Part c - LOOCV

```r
library(boot)

glm.fit.full = glm(mpg ~ ., data=x1)
cv.err.full = cv.glm(x1, glm.fit.full)
cv.err.full$delta[1]
```

```
## [1] 11.37113
```

```r
glm.fit.subset = glm(mpg ~ weight + year + origin, data=x1)
cv.err.subset = cv.glm(x1, glm.fit.subset)
cv.err.subset$delta[1]
```

```
## [1] 11.32646
```

Since the subset model MSE is less than the full model MSE, under LOOCV, we prefer the subset model as opposed to the full model. We can also relate this to the bias-variance trade off since because the subset model as fewer predictors there is lower variance without too much bias being introduced.

2 - Polynomial & Regression Splines

Dataset is Carseats.

Part a - Polynomial regression

```r
x2 = ISLR2::Carseats

cv.error = numeric(5)
for (i in 1:5) {
  glm.fit = glm(Sales ~ poly(Price, i), data=x2)
  cv.error[i] = cv.glm(x2, glm.fit, K=10)$delta[1]
}
cv.error
```

```
## [1] 6.417070 6.468640 6.610566 6.355842 6.458030
```

```r
which.min(cv.error) # best degree of polynomial
```

```
## [1] 4
```

The best degree of the polynomial according to 10-fold CV is degree 4. As we increase the degree our polynomial fit becomes more flexible so the bias gets lower, however as a result of this flexibility the variance increases. As we decrease the degree of our polynomial fit it cannot flex to the fit of the data as much and it must abide by its polynomial shape more, meaning there is more bias and now less variance.

Part b - Splines

```r
fit.poly = glm(Sales ~ poly(Price, which.min(cv.error)), data=x2)

library(splines)
# b spline fit
fit.b = glm(Sales ~ bs(Price, df=4), data=x2)
summary(fit.b)
```

```
##
## Call:
## glm(formula = Sales ~ bs(Price, df = 4), data = x2)
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)           11.018      2.239   4.921 1.27e-06 ***
## bs(Price, df = 4)1     2.067      3.092   0.669 0.504095
## bs(Price, df = 4)2    -6.076      2.117  -2.870 0.004328 **
## bs(Price, df = 4)3    -3.944      2.674  -1.475 0.140964
## bs(Price, df = 4)4   -10.555      2.733  -3.861 0.000132 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 6.368468)
##
##     Null deviance: 3182.3  on 399  degrees of freedom
## Residual deviance: 2515.5  on 395  degrees of freedom
## AIC: 1882.7
##
## Number of Fisher Scoring iterations: 2
```

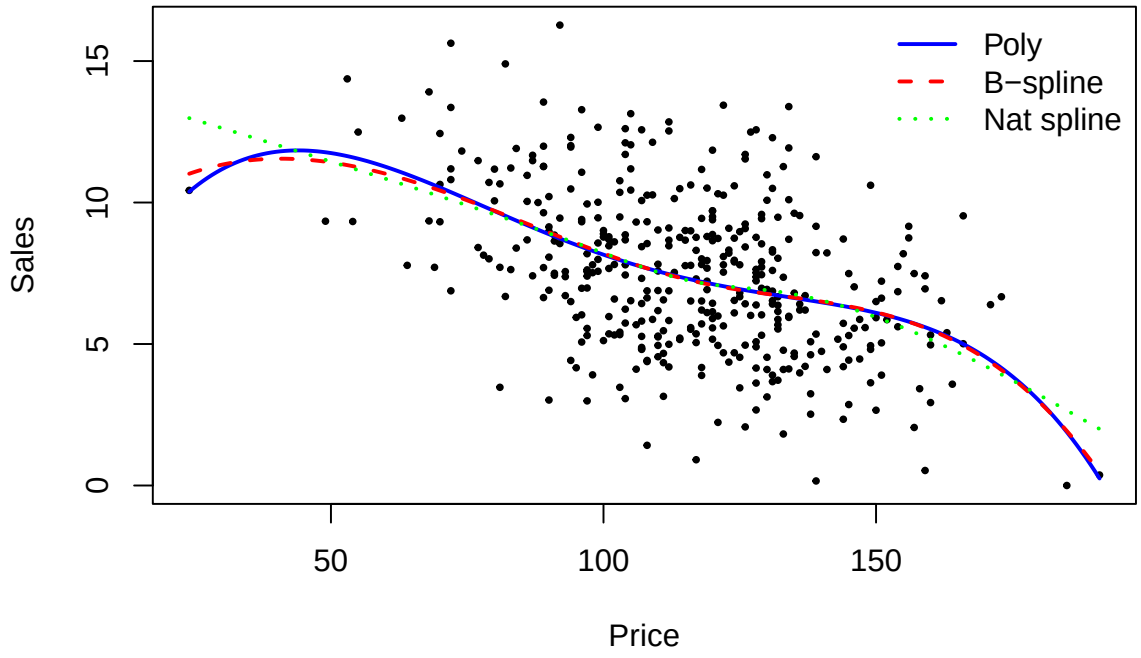
```r
# natural spline fit
fit.ns = glm(Sales ~ ns(Price, df=4), data=x2)
summary(fit.ns)
```

```
##
## Call:
## glm(formula = Sales ~ ns(Price, df = 4), data = x2)
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)           12.983      1.714   7.575 2.57e-13 ***
## ns(Price, df = 4)1    -5.932      1.640  -3.617 0.000337 ***
## ns(Price, df = 4)2    -4.775      1.136  -4.204 3.25e-05 ***
## ns(Price, df = 4)3   -11.744      3.691  -3.182 0.001577 **
## ns(Price, df = 4)4    -9.357      1.525  -6.135 2.07e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 6.408142)
##
##     Null deviance: 3182.3  on 399  degrees of freedom
## Residual deviance: 2531.2  on 395  degrees of freedom
## AIC: 1885.1
```

```
##
## Number of Fisher Scoring iterations: 2
```

Part c - Plot 3 fits

```
plot(x2$Price, x2$Sales, xlab="Price", ylab="Sales", pch=20, cex=0.6)

# grid of price values
grid = seq(min(x2$Price), max(x2$Price), length.out=200)
lines(grid, predict(fit.poly, newdata=list(Price=grid)), lwd=2, col="blue")
lines(grid, predict(fit.b, newdata=list(Price = grid)), lwd=2, lty=2, col="red")
lines(grid, predict(fit.ns, newdata=list(Price = grid)), lwd=2, lty=3, col="green")
legend("topright", legend=c("Poly", "B-spline", "Nat spline"),
       lty = c(1,2,3), lwd=2, col=c("blue", "red", "green"), bty="n")
```



Part d - Use 10-fold CV to compare models

```
cv.poly = cv.glm(x2, fit.poly, K=10)$delta[1]
cv.bs = cv.glm(x2, fit.b, K=10)$delta[1]
```

```
## Warning in bs(Price, degree = 3L, knots = 117, Boundary.knots = c(49, 191: some
```

```
## 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(Price, degree = 3L, knots = 117, Boundary.knots = c(49, 191: some
## 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
## Warning in bs(Price, degree = 3L, knots = 117, Boundary.knots = c(24, 185: some
## 'x' values beyond boundary knots may cause ill-conditioned bases
## Warning in bs(Price, degree = 3L, knots = 117, Boundary.knots = c(24, 185: some
## 'x' values beyond boundary knots may cause ill-conditioned bases
```

```
cv.ns = cv.glm(x2, fit.ns, K=10)$delta[1]
c(cv.poly, cv.bs, cv.ns)
```

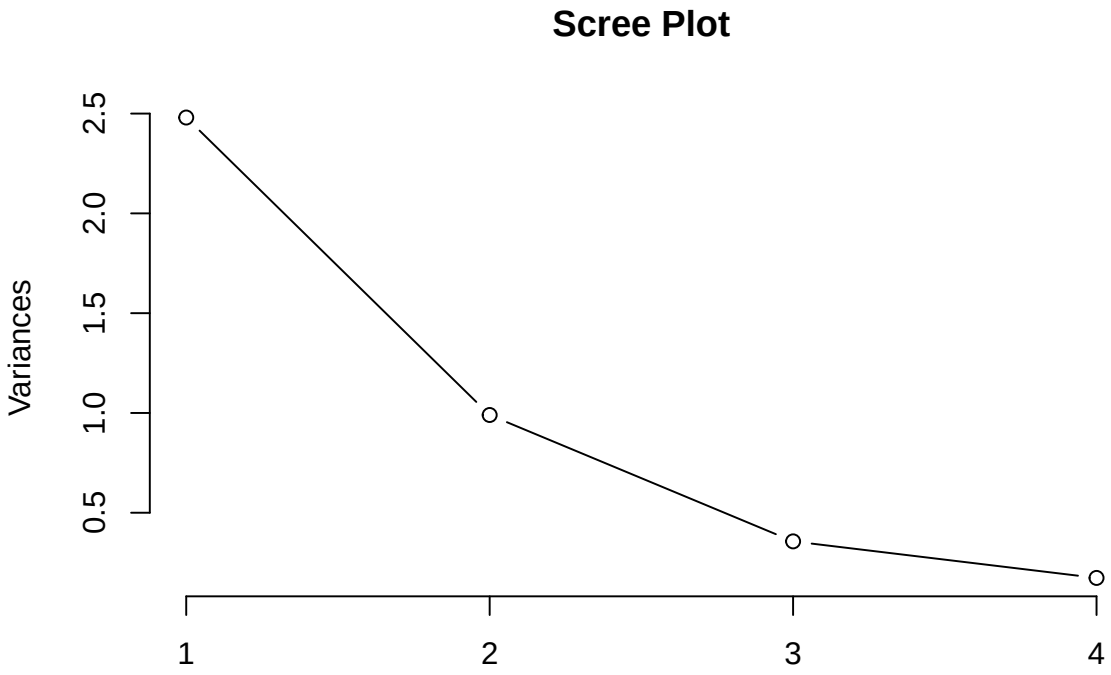
```
## [1] 6.381439 6.447354 6.546430
```

The polynomial fit model has the lowest 10 fold cross validation error, therefore it strikes the best balance between bias and variance.

Question 3 - Unsupervised Learning

Part a - PCA (in R)

```
x3 = USArrests

# PCA on data
pca = prcomp(x3, scale=TRUE)

# scree plot
plot(pca, type="l", main="Scree Plot")
```




```
# variance explained
var.prop = pca$sdev^2 / sum(pca$sdev^2)
var.prop[1:2]
```

```
## [1] 0.6200604 0.2474413
```

Why eigen values = variance? The sample covariance matrix's eigenvalues are the variances of the data projected onto each principal direction.

4 - Ridge & Lasso Regression

Using Boston dataset.

Part a - Ridge regression fit and MSE

```
x4 = ISLR2::Boston
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-8
```

```
X = model.matrix(medv ~ ., x4)[, -1]
Y = x4$medv

grid = 10^seq(10,-2,length=100)
ridge.mod = cv.glmnet(X, Y, alpha=0, lambda=grid)
lambda.ridge = ridge.mod$lambda.min
lambda.ridge
```

```
## [1] 0.09326033
```

```
mse.ridge = min(ridge.mod$cvm)
mse.ridge
```

```
## [1] 23.81286
```

Part b - Lasso regression fit and MSE

```
lasso.mod = cv.glmnet(X, Y, alpha=1, lambda=grid)
lambda.lasso = lasso.mod$lambda.min
lambda.lasso
```

```
## [1] 0.0231013
```

```
mse.lasso = min(ridge.mod$cvm)
mse.lasso
```

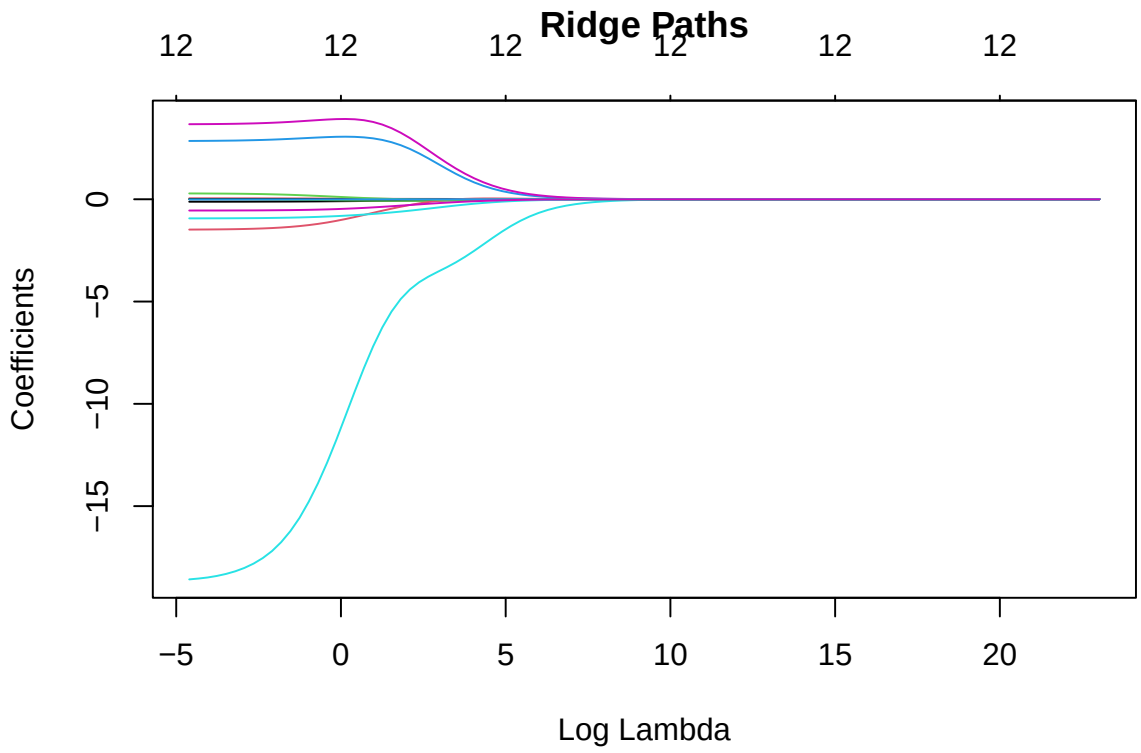
```
## [1] 23.81286
```

```
# coefficients of model with lambda that gives lowest MSE
coef(lasso.mod, s = "lambda.min")
```

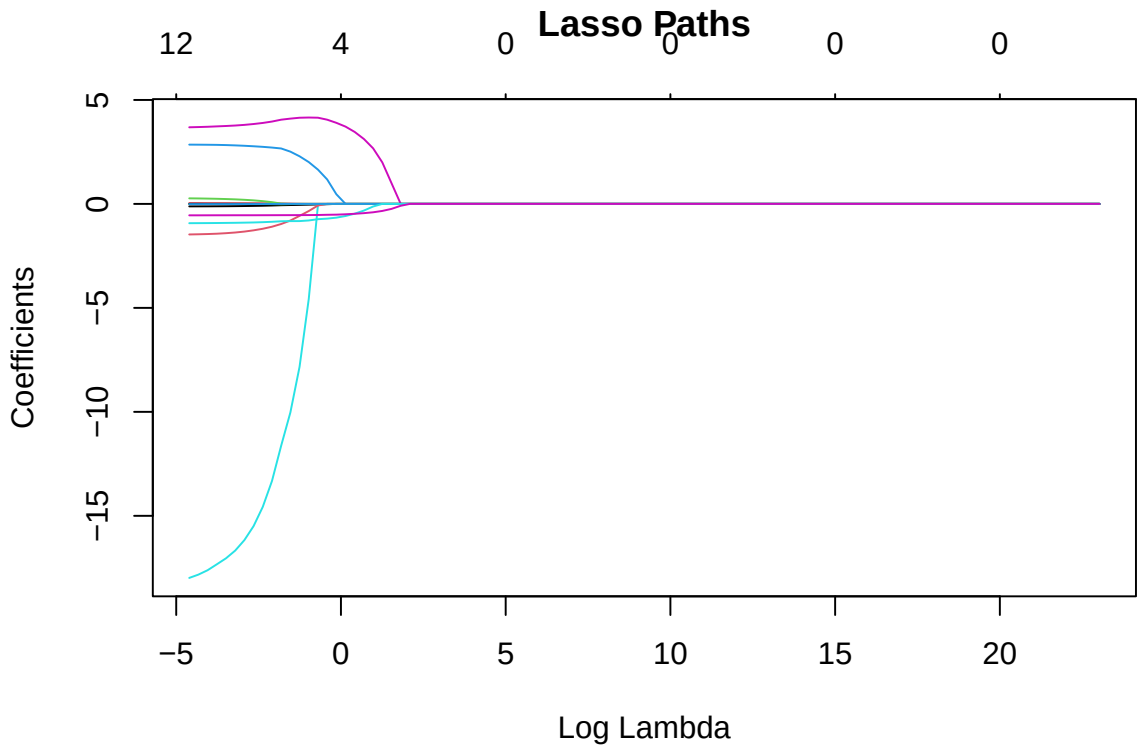
```
## 13 x 1 sparse Matrix of class "dgCMatrix"
##                         s1
## (Intercept)  3.974454e+01
## crim        -1.132081e-01
## zn           4.242999e-02
## indus        .
## chas         2.839954e+00
## nox         -1.733255e+01
## rm           3.729120e+00
## age          1.427585e-05
## dis         -1.433203e+00
## rad          2.460619e-01
## tax         -1.068205e-02
## ptratio     -9.181554e-01
## lstat       -5.460282e-01
```

Part c - Plot coefficient paths

```r
# ridge regression coefficent paths
plot(ridge.mod$glmnet.fit, xvar="lambda", main="Ridge Paths")
```




```r
# lasso regression coefficient paths
plot(lasso.mod$glmnet.fit, xvar="lambda", main="Lasso Paths")
```



An increasing λ places more emphasis on shrinking the coefficients so we expect there to be higher bias but lower variance. Ridge regression shrinks all continuously but not all the way to 0 (so it doesn't perform parameter selection), however lasso regression can set some parameters exactly to 0 (so it does perform parameter selection).

5 - Classification Methods

Using Default dataset.

Part a - Logistic regression

```
x5 = ISLR2::Default
set.seed(1463797)
n = nrow(x5)
train = sample(1:n, size=floor(0.7*n))
test = -train


glm.fits = glm(default ~ ., family="binomial", data=x5[train, ])
summary(glm.fits)
```

```
##
## Call:
```

```
## glm(formula = default ~ ., family = "binomial", data = x5[train,
##     ])
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.090e+01  5.896e-01 -18.494   <2e-16 ***
## studentYes  -4.962e-01  2.829e-01  -1.754   0.0794 .
## balance      5.579e-03  2.705e-04  20.625   <2e-16 ***
## income       1.012e-05  9.796e-06   1.033   0.3015
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 2030.3  on 6999  degrees of freedom
## Residual deviance: 1125.2  on 6996  degrees of freedom
## AIC: 1133.2
##
## Number of Fisher Scoring iterations: 8
```

```
glm.probs = predict(glm.fits, x5[test, ], type="response")
glm.pred = ifelse(glm.probs > 0.5, "Yes", "No")
acc.log = mean(glm.pred == x5[test, ]$default)

# odds ratio for balance
exp(coef(glm.fits)["balance"])
```

```
##  balance
## 1.005595
```

Part b - LDA and QDA

```
library(MASS)

# LDA
lda.fit = lda(default ~ ., data=x5[train, ])
lda.pred = predict(lda.fit, x5[test, ])$class
acc.lda = mean(lda.pred == x5[test, ]$default)
acc.lda
```

```
## [1] 0.9726667
```

```
# QDA
qda.fit = qda(default ~ ., data=x5[train, ])
qda.pred = predict(qda.fit, x5[test, ])$class
acc.qda = mean(qda.pred == x5[test, ]$default)
acc.qda
```

```
## [1] 0.972
```

Part c - KNN classification

```
library(class)
# only numeric predictors balance and income for simplicity
train.X = x5[train, ][, c("balance","income")]
test.X = x5[test, ][, c("balance","income")]
train.Y = x5[train, ]$default

k.values = c(1,3,5,7)
knn.rp = numeric(length(k.values))

for (i in 1:length(k.values)) {
  set.seed(1463797)
  knn.pred = knn(train.X, test.X, train.Y, k=k.values[i])
  knn.rp[i] = mean(knn.pred == x5[test, ]$default)
}

best.k = k.values[which.max(knn.rp)]
best.k
```

```
## [1] 3
```

```
best.knn.acc = max(knn.rp)
best.knn.acc
```

```
## [1] 0.9696667
```

Part d - Fit linear SVM

```
library(e1071)
set.seed(1463797)

tune.out = tune(svm, default ~ ., data=x5[train, ], kernel="linear",
                ranges = list(cost = c(0.01, 0.1, 1, 10)))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.033
##
## - Detailed performance results:
##    cost error  dispersion
```

```
## 1  0.01 0.033 0.005926611
## 2  0.10 0.033 0.005926611
## 3  1.00 0.033 0.005926611
## 4 10.00 0.033 0.005926611
```

```
best.model = tune.out$best.model
summary(best.model)
```

```
##
## Call:
## best.tune(METHOD = svm, train.x = default ~ ., data = x5[train, ],
##     ranges = list(cost = c(0.01, 0.1, 1, 10)), kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  0.01
##
## Number of Support Vectors:  469
##
##  ( 238 231 )
##
##
## Number of Classes:  2
##
## Levels:
##  No Yes
```

```
svm.pred = predict(best.model, x5[test, ])

acc.svm = mean(svm.pred == x5[test, ]$default)
acc.svm
```

```
## [1] 0.966
```

Part e - Rank models

```
results = data.frame(
  Method = c("Logistic", "LDA", "QDA", paste0("KNN(k=",best.k,")"),"SVM"),
  Accuracy = c(acc.log, acc.lda, acc.qda, best.knn.acc, acc.svm)
)
results[order(-results$Accuracy), ]
```

```
##     Method  Accuracy
## 1 Logistic 0.9726667
## 2      LDA 0.9726667
## 3      QDA 0.9720000
## 4 KNN(k=3) 0.9696667
## 5      SVM 0.9660000
```

Interpretations of Bias-Variance:

- **LDA vs QDA**: QDA is more flexible (lower bias) but can overfit (higher variance); in this case LDA is slightly better than QDA (in general QDA may win when covariances differ strongly).
- **KNN**: A small k gives lower bias and high variance and a large k gives high bias and lower variance.
- **SVM Cost**: A higher cost means margin violations are penalised more, giving lower bias and higher variance.

C.2 AI Prac Exam 2

The following is some example questions completed on `ISLR` datasets in R.

AI Prac Exam #2

Omar

2025-06-26

```
library(boot)
library(MASS)
library(splines)
library(class)
library(e1071)
library(ISLR2)
library(leaps)
library(glmnet)
```

Q1 - Linear Regression & Subset Selection

Dataset is `Carseats` and predicting `Sales`.

Part 1 - What type of problem is this and summarise the data

The problem is a regression problem since the response variable which is wage is a continuous numerical variable. The predictors we have access to are a mix of numeric variables, some are numerical and some are factor variables (for simplicity we will drop `ShelveLoc`)

Part 2 - Validation

Choose either a 70/30 test/train split or 10-fold CV. Briefly justify in bias-variance terms.

We will choose a 10-fold CV since it uses all of the data available for training and validating, yielding a more stable test error-estimate (with lower variance). It avoids potential higher variance of a single train/test split of the data.

Part 3 - Best-subset selection

```
x1 = Carseats

regfit = regsubsets(Sales ~ . - ShelveLoc, data=x1, method="exhaustive", nvmax=10)
ss.reg = summary(regfit)
ss.reg
```

```
## Subset selection object
## Call: regsubsets.formula(Sales ~ . - ShelveLoc, data = x1, method = "exhaustive",
##     nvmax = 10)
## 9 Variables  (and intercept)
##             Forced in Forced out
## CompPrice       FALSE      FALSE
## Income          FALSE      FALSE
## Advertising     FALSE      FALSE
## Population      FALSE      FALSE
## Price           FALSE      FALSE
## Age             FALSE      FALSE
## Education       FALSE      FALSE
## UrbanYes        FALSE      FALSE
## USYes           FALSE      FALSE
## 1 subsets of each size up to 9
## Selection Algorithm: exhaustive
##          CompPrice Income Advertising Population Price Age Education UrbanYes
## 1  ( 1 ) " "       " "    " "         " "        "*"   " " " "       " "
## 2  ( 1 ) "*"       " "    " "         " "        "*"   " " " "       " "
## 3  ( 1 ) "*"       " "    "*"         " "        "*"   " " " "       " "
## 4  ( 1 ) "*"       " "    "*"         " "        "*"   "*" " "       " "
## 5  ( 1 ) "*"       "*"    "*"         " "        "*"   "*" " "       " "
## 6  ( 1 ) "*"       "*"    "*"         " "        "*"   "*" "*"       " "
## 7  ( 1 ) "*"       "*"    "*"         " "        "*"   "*" "*"       "*"
## 8  ( 1 ) "*"       "*"    "*"         " "        "*"   "*" "*"       "*"
## 9  ( 1 ) "*"       "*"    "*"         "*"        "*"   "*" "*"       "*"
##          USYes
## 1  ( 1 ) " "
## 2  ( 1 ) " "
## 3  ( 1 ) " "
## 4  ( 1 ) " "
## 5  ( 1 ) " "
## 6  ( 1 ) " "
## 7  ( 1 ) " "
## 8  ( 1 ) "*"
## 9  ( 1 ) "*"
```

```
min.bic = which.min(ss.reg$bic)
# lowest bic value
ss.reg$bic[min.bic]
```

```
## [1] -274.9223
```

```
# best model according to lowest bic
ss.reg$which[min.bic, ]
```

```
## (Intercept)   CompPrice      Income Advertising  Population       Price
##        TRUE        TRUE        TRUE        TRUE       FALSE        TRUE
##         Age   Education    UrbanYes       USYes
##        TRUE       FALSE       FALSE       FALSE
```

```r
best_preds = names(coef(regfit, min.bic))[-1]

# fit best model in lm
best.model = lm(paste("Sales ~ ", paste(best_preds, collapse="+")), data=x1)
summ.b.model = summary(best.model)
summ.b.model
```

```
##
## Call:
## lm(formula = paste("Sales ~ ", paste(best_preds, collapse = "+")),
##     data = x1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.9071 -1.3081 -0.1892  1.1495  4.6980
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.109190   0.943940   7.531 3.46e-13 ***
## CompPrice    0.093904   0.007792  12.051  < 2e-16 ***
## Income       0.013092   0.003465   3.779 0.000182 ***
## Advertising  0.130611   0.014572   8.963  < 2e-16 ***
## Price       -0.092543   0.005044 -18.347  < 2e-16 ***
## Age         -0.044971   0.005994  -7.503 4.20e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.927 on 394 degrees of freedom
## Multiple R-squared:  0.5403, Adjusted R-squared:  0.5345
## F-statistic: 92.62 on 5 and 394 DF,  p-value: < 2.2e-16
```

```r
adjR2 = summ.b.model$adj.r.squared
RSE = summ.b.model$sigma

best_preds # selected predictors
```

```
## [1] "CompPrice"   "Income"      "Advertising" "Price"       "Age"
```

```r
adjR2 # adjusted R^2
```

```
## [1] 0.5344646
```

```r
RSE # residual SE
```

```
## [1] 1.926898
```

Adjusted $R^2 = 0.5345$ and is the variance explained (after penalty)–resonable in-sample fit (low bias) but leaves noise (variance). $RSE = 1.927$, so an average error of of ± 1.927 units of Sales; a low RSE relative to a range suggests a good fit but risks overfitting if too many predictors.

Q2 - Non-linear Regression & Smoothing

Dataset is `Hitters` and predicting `Salary`, using predictors `Hits`.

Part 1 - Polynomial Regression and CV

```
x2 = Hitters
x2 = na.omit(Hitters)

set.seed(1463797)

cv.err = numeric(5)
for (i in 1:5) {
  poly.fit = glm(Salary ~ poly(Hits, i), data=x2)
  cv.err[i] = cv.glm(x2, poly.fit, K=10)$delta[1]
}

# Cross validation error for degrees 1,2,3,4,5
cv.err
```

```
## [1] 166313.3 165866.7 164474.8 164327.4 189825.3
```

```
# degree which gives minimum cross validation error
min.degree = which.min(cv.err)
min.degree
```

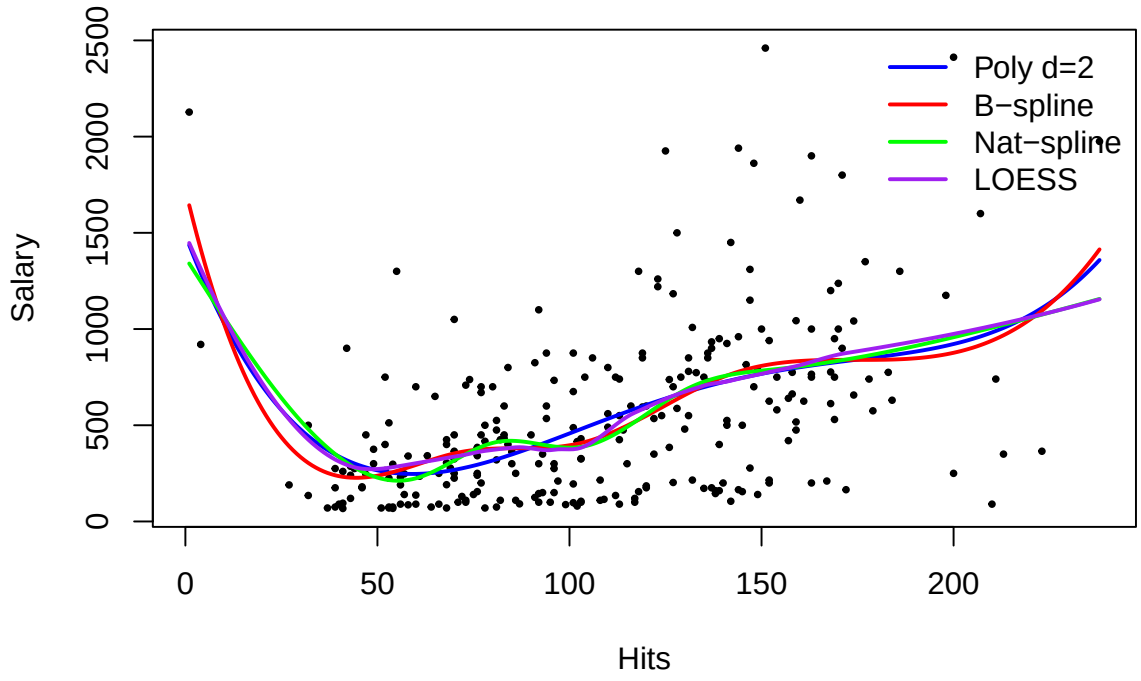
```
## [1] 4
```

The degree of the polynomial in the model trades bias and variance in the model. The reason for this is because a high degree polynomial gives us a more flexible polynomial which can fit to the data better, this may cause overfitting, and due to its flexibility the bias is low and the variance is high. Conversely, if the degree of the polynomial is low then it cannot flex to the data as much as it must abide by its shape (for example a degree 1 polynomial is forced to have a linear fit), therefore it has high bias and low variance. The cross validation error is minimum with a degree of 4 polynomial, therefore the CV error prefers an intermediate degree (and not extremely high or low one) because it choose a "sweet-spot" for this bias-variance trade off.

Part 2 - Spline and LOESS fits

```
# create models
bs_mod = lm(Salary ~ bs(Hits, df=6), data=x2)
ns_mod = lm(Salary ~ ns(Hits, df=6), data=x2)
loess_mod = loess(Salary ~ Hits, span=0.5, data=x2)
best_poly_mod = lm(Salary ~ poly(Hits, min.degree), data=x2)

# plot with all models overlaid on data
plot(x2$Hits, x2$Salary, xlab="Hits", ylab="Salary", pch=20, cex=0.6)
grid = seq(min(x2$Hits), max(x2$Hits))
lines(grid, predict(best_poly_mod, newdata=list(Hits=grid)), lwd=2, col="blue")
```

```r
lines(grid, predict(bs_mod, newdata=list(Hits=grid)), lwd=2, col="red")
lines(grid, predict(ns_mod, newdata=list(Hits=grid)), lwd=2, col="green")
lines(grid, predict(loess_mod, newdata=data.frame(Hits=grid)), lwd=2,
      col="purple")
legend("topright", legend=c("Poly d=2","B-spline","Nat-spline","LOESS"),
       lwd=2, col=c("blue", "red", "green", "purple"), bty="n")
```



Part 3 - GAM Theory

Describe how a GAM combines smooth functions for multiple predictors and explain how smoothing penalties control bias vs. variance in GAM.

A GAM model is a generalised additive model where each predictor is a function that can take any suitable form (e.g. f_j for predictor j). This allows f_j to be a smooth function, with smoothing parameters λ_j which penalise flexibility "wiggliness". So to control the bias and variance tradeoff in a GAM, a high λ_j allows for smoother f_j, so a higher bias and lower variance, and a lower λ_j allows for a less smooth and "wigglier" f_j, so lower bias and higher variance.

Q3 - Penalised Regression and PCR/PLS

Dataset is `College` and we are predicting `Apps`. All other predictors except `Private`.

Part 1 - Data preperation (model matrix)

```
# model matrix to design matrix X (no intercept) and response Y
x3 = College
X = model.matrix(Apps ~ . - 1 - Private, data=x3)
y = x3$Apps
```

Part 2 - Ridge and lasso regression with CV

```
cv_r = cv.glmnet(X, y, alpha=0, nfolds=10)
cv_l = cv.glmnet(X, y, alpha=1, nfolds=10)

lam_r = cv_r$lambda.min
lam_l = cv_l$lambda.min
mse_r = min(cv_r$cvm)
mse_l = min(cv_l$cvm)
coef_l = coef(cv_l, s="lambda.min")
coef_l_mat <- as.matrix(coef_l)
zero_coef <- rownames(coef_l_mat)[coef_l_mat[, 1] == 0]

list(
  ridge = list(lambda=lam_r, CV_MSE=mse_r),
  lasso= list(lambda=lam_l, CV_MSE=mse_l, zeros=zero_coef)
)
```

```
## $ridge
## $ridge$lambda
## [1] 364.8993
##
## $ridge$CV_MSE
## [1] 1644103
##
##
## $lasso
## $lasso$lambda
## [1] 2.137223
##
## $lasso$CV_MSE
## [1] 1288258
##
## $lasso$zeros
## [1] "Books"
```

Part 3 - PCR and PLS (manual)

```
# compute principal components using PCA then do PCR manually (making lm of PCs)
pca = prcomp(X, scale.=TRUE)
PCs = pca$x[,1:3]
```

```r
pcr = lm(y ~ PCs)
summary(pcr)
```

```
##
## Call:
## lm(formula = y ~ PCs)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5748.8 -1062.7  -136.9   754.8 28526.1
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3001.64      70.21  42.753   <2e-16 ***
## PCsPC1        556.25      30.68  18.130   <2e-16 ***
## PCsPC2       1563.20      35.62  43.880   <2e-16 ***
## PCsPC3       -179.69      64.96  -2.766   0.0058 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1957 on 773 degrees of freedom
## Multiple R-squared:  0.7453, Adjusted R-squared:  0.7443
## F-statistic: 753.9 on 3 and 773 DF,  p-value: < 2.2e-16
```

```r
summary(pcr)$r.squared
```

```
## [1] 0.7452868
```

The R^2 statistic is around 75% for three principal components.

- **PCR** finds components explaining the highest variance in $\mathbf{X}$, then regresses $\mathbf{y}$ on them–may miss directions most predictive of $\mathbf{y}$, leading to bias if few PCs were chosen
- **PLS** finds components that maximise covariance between $\mathbf{X}$ and $\mathbf{y}$, often reducing bias with fewer components, but potentially increasing variance if over fitted.
- **Ridge/Lasso** penalise coefficients directly: ridge shrinks all (reducing variance), lasso zeros some (variable selection with moderate bias).

Q4 - Classification and Confusion Matricies

Dataset is `OJ` and predicting `Purchase`, using predictors `PriceCH`, `PriceMM`, `DiscCH` and `DiscMM`.

Part 1 - Data split (70/30)

```r
x3 = OJ
train_id = sample(1:nrow(x3), size=floor(0.7*nrow(x3)))
test_id = -train_id

train = x3[train_id, ]
test = x3[test_id, ]
```

Part 2 - Logistic, LDA & QDA

```
# Fit models on training data
# Logistic
log.fit = glm(Purchase ~ PriceCH+PriceMM+DiscCH+DiscMM,
              family="binomial", data=train)
p.log = predict(log.fit, test, type="response")
pred.log = factor(ifelse(p.log > 0.5, "CH", "MM"), levels=c("CH", "MM"))
cm.log = table(Pred = pred.log, True=test$Purchase)

# LDA
lda.fit = lda(Purchase ~ PriceCH+PriceMM+DiscCH+DiscMM, data=train)
pred.lda = predict(lda.fit, test)$class
cm.lda = table(Pred = pred.lda, True=test$Purchase)

# QDA
qda.fit = qda(Purchase ~ PriceCH+PriceMM+DiscCH+DiscMM, data=train)
pred.qda = predict(qda.fit, test)$class
cm.qda = table(Pred = pred.qda, True=test$Purchase)

# report accuracy, sensitivity and specificity
metrics <- function(cm){
  acc  <- sum(diag(cm))/sum(cm)
  sens <- cm["MM","MM"]/sum(cm[,"MM"])
  spec <- cm["CH","CH"]/sum(cm[,"CH"])
  c(accuracy=acc, sensitivity=sens, specificity=spec)
}

# Confusion Matricies and Metrics
log_met <- metrics(cm.log)
cm.log
```

```
##     True
## Pred  CH  MM
##   CH  16  36
##   MM 171  98
```

```
log_met
```

```
##    accuracy sensitivity specificity
##   0.3551402   0.7313433   0.0855615
```

```
lda_met <- metrics(cm.lda)
cm.lda
```

```
##     True
## Pred  CH  MM
##   CH 171  98
##   MM  16  36
```

```
lda_met
```

```
##    accuracy sensitivity specificity
##   0.6448598   0.2686567   0.9144385
```

```
qda_met <- metrics(cm.qda)
cm.qda
```

```
##     True
## Pred  CH  MM
##   CH 154  73
##   MM  33  61
```

```
qda_met
```

```
##    accuracy sensitivity specificity
##   0.6697819   0.4552239   0.8235294
```

Part 3 - K-nearest neighbours

```
# need to tune K, note KNN can only use numerical predictors
k = 1:10
acc_knn <- sapply(k, function(k){
  pred <- knn(train[,c("PriceCH","PriceMM")],
              test[,c("PriceCH","PriceMM")],
              train$Purchase, k=k)
  metrics(table(Pred=pred,True=test$Purchase))["accuracy"]
})
best_k  <- k[which.max(acc_knn)]
pred_K  <- knn(train[,c("PriceCH","PriceMM")],
               test[,c("PriceCH","PriceMM")],
               train$Purchase, k=best_k)
cm_knn  <- table(Pred=pred_K,True=test$Purchase)
knn_met <- metrics(cm_knn)

list(KNN=list(k=best_k, cm=cm_knn, met=knn_met))
```

```
## $KNN
## $KNN$k
## [1] 1
##
## $KNN$cm
##     True
## Pred  CH  MM
##   CH 163  94
##   MM  24  40
##
## $KNN$met
##    accuracy sensitivity specificity
##   0.6323988   0.2985075   0.8716578
```

A small k means we are more flexible (low bias, and high variance), and a high k means we are more smooths (high bias, low variance)

Part 4 - Radial Kernel SVM

```
# need to tune for best model as well
t_out = tune(svm, Purchase~PriceCH+PriceMM+DiscCH+DiscMM,
             data=train, kernel="radial",
             ranges=list(cost=c(0.1, 1, 10),
                         gamma=c(0.1, 1, 10)))
svm_f = t_out$best.model
pred_S = predict(svm_f, test)
cm_svm = table(Pred = pred_S, True=test$Purchase)
svm_met = metrics(cm_svm)
best_c = t_out$best.parameter$cost
best_g = t_out$best.parameter$gamma

cm_svm
```

```
##     True
## Pred  CH  MM
##   CH 161  86
##   MM  26  48
```

```
list(SVM = list(cost=best_c, gamma=best_g, met=svm_met))
```

```
## $SVM
## $SVM$cost
## [1] 10
##
## $SVM$gamma
## [1] 10
##
## $SVM$met
##    accuracy sensitivity specificity
##   0.6510903   0.3582090   0.8609626
```

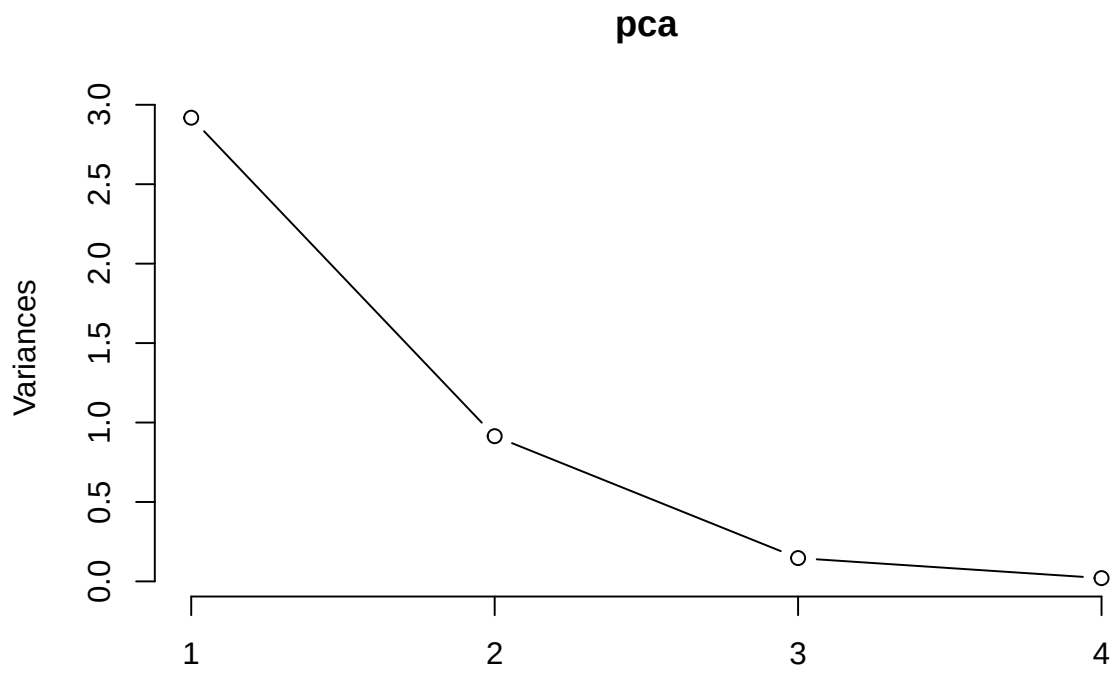
Using the R defintion of cost. When our cost is higher we seek narrow margins meaning our bias is lower and our variance is higher. Conversely when our cost is lower we seek wider margins meaning our bias is higher and our variance is lower.

Q5 - Unsupervised Learning and K-Means Clustering

Dataset is `iris` (base R).

Part 1 - PCA and Scree Plot

```r
X = scale(iris[,1:4])
pca = prcomp(X)

# scree plot
plot(pca, type="l")
```




```r
# cumulative variances of the principal components
var_exp = pca$sdev^2/sum(pca$sdev^2)
cumsum(var_exp)[1:3]
```

```
## [1] 0.7296245 0.9581321 0.9948213
```

Part 2 - Clustering in R

```r
# report k-means centroids (on PC1 and PC2), give contingency table vs. hierarchical clusters
PCs = pca$x[,1:2]
set.seed(1)
km3 = kmeans(PCs, centers=3)
hc3 = hclust(dist(PCs), method="complete")
hc_cl = cutree(hc3, k=3)

km3$centers
```

```
##          PC1        PC2
## 1  0.5707095  0.8045137
## 2  1.7152903 -0.6008742
## 3 -2.2173249 -0.2879627
```

```r
table(Kmeans=km3$cluster, HClust=hc_cl)
```

```
##       HClust
## Kmeans  1  2  3
##      1  0 53  0
##      2  0 14 33
##      3 49  1  0
```

Part 3 - Kmeans By Hand

Data points are $(0,0), (0,4), (4,0), (4,4)$, need to init centroids $C_1 = (0,0)$ and $C_2 = (4,4)$, compute distances and assign each point, and update new centroids.

=== Initial centers selected === C1^(0) = point P1 = (0, 0) C2^(0) = point P4 = (4, 4)

=== Iteration 1 ===

Point P1 = (0, 0) d(P1, C1) = sqrt((0 - 0)^2 + (0 - 0)^2) = 0.00 d(P1, C2) = sqrt((0 - 4)^2 + (0 - 4)^2) = 5.66 -> assign to cluster 1 (smallest d = 0.00)

Point P2 = (0, 4) d(P2, C1) = sqrt((0 - 0)^2 + (4 - 0)^2) = 4.00 d(P2, C2) = sqrt((0 - 4)^2 + (4 - 4)^2) = 4.00 -> assign to cluster 1 (smallest d = 4.00)

Point P3 = (4, 0) d(P3, C1) = sqrt((4 - 0)^2 + (0 - 0)^2) = 4.00 d(P3, C2) = sqrt((4 - 4)^2 + (0 - 4)^2) = 4.00 -> assign to cluster 1 (smallest d = 4.00)

Point P4 = (4, 4) d(P4, C1) = sqrt((4 - 0)^2 + (4 - 0)^2) = 5.66 d(P4, C2) = sqrt((4 - 4)^2 + (4 - 4)^2) = 0.00 -> assign to cluster 2 (smallest d = 0.00)

Cluster assignments:

Updating centers: C1^(1) = (1/3) * [(0,0) + (0,4) + (4,0)] = (1.33, 1.33) C2^(1) = (1/1) * [(4,4)] = (4, 4)

=== Iteration 2 ===

Point P1 = (0, 0) d(P1, C1) = sqrt((0 - 1.33)^2 + (0 - 1.33)^2) = 1.89 d(P1, C2) = sqrt((0 - 4)^2 + (0 - 4)^2) = 5.66 -> assign to cluster 1 (smallest d = 1.89)

Point P2 = (0, 4) d(P2, C1) = sqrt((0 - 1.33)^2 + (4 - 1.33)^2) = 2.98 d(P2, C2) = sqrt((0 - 4)^2 + (4 - 4)^2) = 4.00 -> assign to cluster 1 (smallest d = 2.98)

Point P3 = (4, 0) d(P3, C1) = sqrt((4 - 1.33)^2 + (0 - 1.33)^2) = 2.98 d(P3, C2) = sqrt((4 - 4)^2 + (0 - 4)^2) = 4.00 -> assign to cluster 1 (smallest d = 2.98)

Point P4 = (4, 4) d(P4, C1) = sqrt((4 - 1.33)^2 + (4 - 1.33)^2) = 3.77 d(P4, C2) = sqrt((4 - 4)^2 + (4 - 4)^2) = 0.00 -> assign to cluster 2 (smallest d = 0.00)

Cluster assignments:

No change in assignments; converged.

Finished k-means with k = 2

D Appendix: Practice Exam with Solutions (Lecturer Copy)

Refer to next page for practice exam with solutions (note it is shorter than the actual exam).

ACTL30008 ACTUARIAL ANALYTICS AND DATA I

Practice Assessment Sample Solutions

Total Marks: 30 marks

Number of pages:

Authorised materials: R; lecture material

Instructions to students:

This is a practice exam only. The types of questions are consistent with the questions in the final assessment. The length of this practice paper is shorter.

You need to submit a single pdf file named as `ACTL30008_xxxxxx.pdf` where xxxxxx is your student ID by the due time of this test.

You will need to use R to complete this exam when needed. Creating a final submission by R markdown is a preferrable way of editing your answers. You may use a R script file to produce your final submission, but you will need to include the required outputs and/or hand written answers manually.

Question One (2+4+4 = 10 marks)

You are given a data set which is named `ACTL30008_practice.csv`. You are asked to build statistical models that help to predict the response variable Y given new predictor observations.

(a)

Load the data set and build a multiple linear regression model using the data. Describe how well the obtained model fit the given data and show at least two numerical evidences.

(b)

You are suspecting that there might be some non-linear relationship between the response Y and the two predictors.

- Present a polynomial regression model using the data. The degrees of any polynomial terms in your model should be optimised and explain how these best degrees are selected.
- Is this non-linear model a better fit to the given data than the MLR obtained in (a)? Why?

(c)

- Using the LOOCV method to estimate the test MSE of the models you obtained in (a) and (b).
- Which model is likely to give a more accurate prediction for any new observations? why?
- Can the estimated test MSE accurately represent the true test MSE values? Why?

Question One Solutions:

(a)

```
data= read.csv("ACTL30008_practice.csv", header=T)
str(data)
```

```
## 'data.frame':    30 obs. of  3 variables:
##  $ X1: num  2.78 2.63 2.29 2.57 2.71 ...
##  $ X2: num  113 119 101 188 20 ...
##  $ Y : num  99.9 92.6 34.7 78.7 68 ...
```

```
attach(data)
lm.fit=lm(Y~., data)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = Y ~ ., data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.7900 -5.5297 -0.3682  4.1799 19.5911
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -226.8882    17.2725 -13.136 3.05e-13 ***
## X1           108.2979     6.9420  15.600 4.96e-15 ***
## X2             0.1736     0.0257   6.753 2.99e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.566 on 27 degrees of freedom
## Multiple R-squared:  0.927,  Adjusted R-squared:  0.9216
## F-statistic: 171.3 on 2 and 27 DF,  p-value: 4.547e-16
```

The MLR obtained above is a good fit to the given data. Firstly, the overall relationship between the response variable and two predictors is very significant with a p-value $= 4.55 \times 10^{-16}$. Secondly, the $R^2 = 92.7\%$ which also indicate that the MLR is a strong fit.

(b)

```
poly.fit=lm(Y~poly(X1,5)+poly(X2,5),data=data)
summary(poly.fit)
```

```
##
## Call:
## lm(formula = Y ~ poly(X1, 5) + poly(X2, 5), data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.4999 -1.7411  0.0192  2.6680  7.3079
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)    62.745      0.846  74.167  < 2e-16 ***
## poly(X1, 5)1  120.097      5.372  22.355 4.17e-15 ***
## poly(X1, 5)2   13.413      4.758   2.819 0.010959 *
## poly(X1, 5)3  -21.217      5.338  -3.975 0.000812 ***
## poly(X1, 5)4   -3.592      4.833  -0.743 0.466439
## poly(X1, 5)5    6.229      4.836   1.288 0.213251
## poly(X2, 5)1   43.778      5.508   7.948 1.85e-07 ***
## poly(X2, 5)2  -11.919      4.701  -2.535 0.020182 *
## poly(X2, 5)3   -9.746      4.877  -1.998 0.060212 .
## poly(X2, 5)4    8.163      4.964   1.644 0.116554
## poly(X2, 5)5   10.088      5.086   1.983 0.061961 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.634 on 19 degrees of freedom
## Multiple R-squared:  0.9807, Adjusted R-squared:  0.9706
## F-statistic: 96.66 on 10 and 19 DF,  p-value: 4.04e-14
```

```
poly.fit2=lm(Y~poly(X1,3)+poly(X2,2),data=data)
summary(poly.fit2)
```

```
##
## Call:
## lm(formula = Y ~ poly(X1, 3) + poly(X2, 2), data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.2453 -3.1879 -0.8365  2.6919 11.4377
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   62.7447     0.9751  64.347  < 2e-16 ***
## poly(X1, 3)1 121.0470     5.5155  21.947  < 2e-16 ***
## poly(X1, 3)2  15.3261     5.3580   2.860 0.008625 **
## poly(X1, 3)3 -25.2440     5.9896  -4.215 0.000306 ***
## poly(X2, 2)1  40.3076     6.1094   6.598 7.99e-07 ***
## poly(X2, 2)2 -12.3973     5.3995  -2.296 0.030714 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.341 on 24 degrees of freedom
## Multiple R-squared:  0.9676, Adjusted R-squared:  0.9609
## F-statistic: 143.6 on 5 and 24 DF,  p-value: < 2.2e-16
```

- From the first polynomial model that we try one can see that X1 probably has a degree 3 and X2 has degree 2. Of course, CV apporach can be used to find the best degrees as well. The best polynomial regression model is given in `poly.fit2`.
- Yes. The best polynomial regression model has an increased R^2 value which means can better fit the given data than the MLR in (a).

(c)

```
library(boot)
glm.fit1=glm(Y~.,data=data)
cv.err1=cv.glm(data,glm.fit1)
```

```
cv.err1$delta[1]
```

```
## [1] 64.58769
```

```
glm.fit2=glm(Y~poly(X1,3)+poly(X2,2),data=data)
cv.err2=cv.glm(data,glm.fit2)
cv.err2$delta[1]
```

```
## [1] 36.21384
```

- The estimated test MSE for the MLR in (a) is 64.59 and the one for the polynomial regression model in (b) is 36.21. So the polynomial regression model tends to give a more accurate prediction than the MLR.
- In general the LOOCV can estimate the test MSE properly. However, in this case, as the number of observations in the data is only 30, there is a small chance that the estimated test MSE will overestimate the true test MSE.

Question Two (3+1+4+2+3+2 = 15 marks)

This question aims to use a simulated data set to study a classification problem.

(a) Generate two predictors X_1 and X_2 with $n = 50$ as follows:

- the first 25 observations of X_1 are normal with mean 2 and s.d. 1;
- the remaining 25 observations of X_1 are normal with mean -1 and s.d. 2;
- the first 25 observations of X_2 are normal with mean 10 and s.d. 5;
- the remaining 25 observations of X_2 are normal with mean 8 and s.d. 2.

(b) Generate a response Y that takes value 1 for the first 25 observations and takes value 2 for the remaining 25 observations. Note here both 1 and 2 are labels.

(c) Use an appropriate discriminant method to construct a classifier for Y and state the reasons that you choose the method.

(d) Describe the shape of the decision boundary of the classifier you build in (c) with reasons.

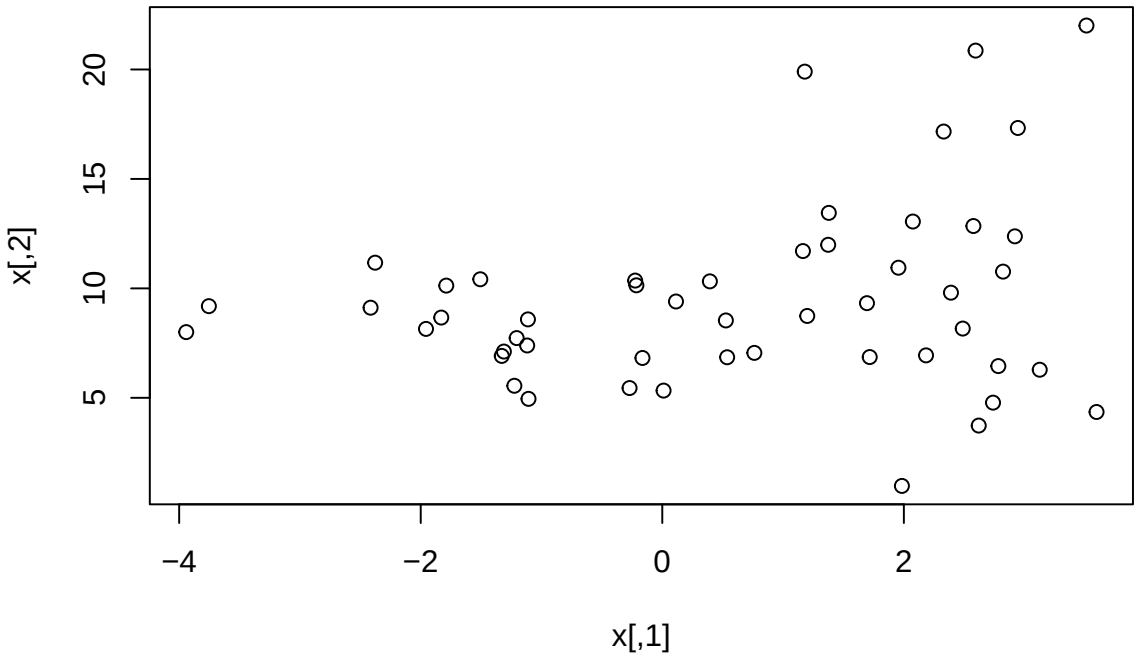
(e) Construct a confusion matrix using the classifier you obtain in (c) and calculate the correct classification rate of this classifier on this simulated data set.

(f) If you build another classifier that gives you a lower correct classification rate on this simulated data set, then can you conclude that the classifier that you obtain in (c) is a better one? Why?

Question Two Solutions:

(a)

```
set.seed(1)
x=matrix(rnorm(50*2), ncol=2)
x[1:25,1]=x[1:25,1]+2
x[1:25,2]=x[1:25,2]*5+10
x[26:50,1]=x[26:50,1]*2-1
x[26:50,2]=x[26:50,2]*2+8
plot(x)
```



(b)

```
y=rep(1, 50)
y[26:50]=2
y=as.factor(y)
data=data.frame(y, x)
str(data)
```

```
## 'data.frame':    50 obs. of  3 variables:
##  $ y : Factor w/ 2 levels "1","2": 1 1 1 1 1 1 1 1 1 1 ...
##  $ X1: num  1.37 2.18 1.16 3.6 2.33 ...
##  $ X2: num  11.99 6.94 11.71 4.35 17.17 ...
```

(c)

According to the information of the data set, we know that the two predictors follow a two-dimensional normal distribution under each class of response, and these two normal distribution have different covariance matricies. Therefore, QDA would be a suitable approach to build a classifier for the response.

```
library(MASS)
qda.fit=qda(y~.,data=data)
qda.fit
```

```
## Call:
## qda(y ~ ., data = data)
##
## Prior probabilities of groups:
##   1   2
```

```
## 0.5 0.5
##
## Group means:
##           X1        X2
## 1  2.1686652 10.827045
## 2 -0.9355373  8.138488
```

(d)

We will get a quadratic decision boundary using the QDA approach.

(e)

```
qda.class=predict(qda.fit,data)$class
table(qda.class,y)
```

```
##          y
## qda.class  1  2
##         1 23  1
##         2  2 24
```

```
mean(qda.class==y)
```

```
## [1] 0.94
```

The correct classification rate is 94%.

(f)

Having a higher correct classification rate on the training set, i.e. a lower training error rate, doesn't guarantee a higher correct classification rate in classifying a new data set. On the contrary, very low training error rate might indicate a potential over-fitting. So we can't conclude that the classfier obtained in (c) is definitely a better one.

Question Three (3 + 2 = 5 marks)

This is an unsupervised learning problem. There are four observations in a given data set, labelled as A, B, C and D, which have the following dissimilarity matrix:

$$\begin{bmatrix} & 0.2 & 0.4 & 0.6 \\ 0.2 & & 0.1 & 0.5 \\ 0.4 & 0.1 & & 0.3 \\ 0.6 & 0.5 & 0.3 & \end{bmatrix}$$

For example, the dissimilarity between A and C is 0.4.

(a) (3 marks)

On the basis of this dissimilarity matrix, sketch the average linkage dendrogram that results from hierarchically clustering these four observations. Indicate in your dendrogram the height at which each fusion occurs, as well as the observations corresponding to each leaf in the dendrogram.

(b) (2 marks)

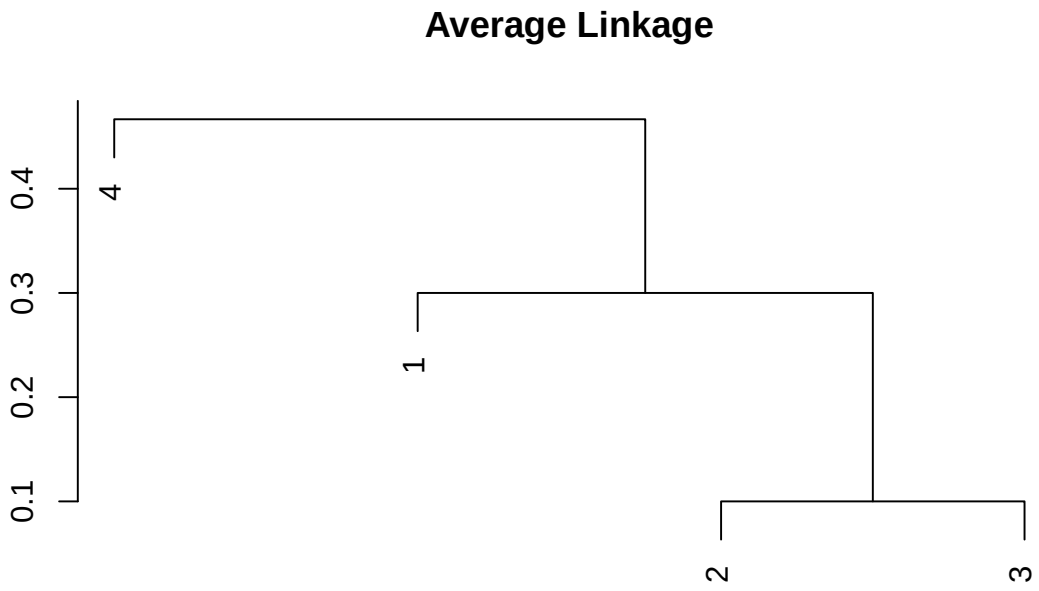
Suppose that we cut the dendrogram obtained in (a) such that two clusters result. Suggest a height of cutting and find which observations are in each cluster.

Question Three solutions:

(a)

You should draw this dendrogram by hand and present the scanned picture here.

```
d = as.dist(matrix(c(0, 0.2, 0.4, 0.6,
                     0.2, 0, 0.1, 0.5,
                     0.4, 0.1, 0.0, 0.3,
                     0.6, 0.5, 0.3, 0.0), nrow=4))
plot(hclust(d, method="average"),main="Average Linkage", xlab="", sub="",ylab="")
```



(b)

We can cut at 0.4 to get two dendrogram. The two clusters are (1,2,3) and (4).

END OF QUESTIONS

END OF EXAM

E Appendix: Practice Exam with Solutions (Personal Copy)

Refer to next page for practice exam with solutions (note it is shorter than the actual exam).

ACTL30008_1463797

Omar

2025-06-25

Set up libraries

```
library(boot)
library(MASS)
library(splines)
library(e1071)
library(ISLR)
library(ROCR)
library(leaps)
library(glmnet)
library(class)
```

Question 1

Part a - fit a MLR

```
x1 = read.csv("ACTL30008_practice.csv")

# MLR model
mlr.model = lm(Y ~ X1 + X2, data=x1)
summary(mlr.model)
```

```
##
## Call:
## lm(formula = Y ~ X1 + X2, data = x1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9.7900 -5.5297 -0.3682  4.1799 19.5911
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -226.8882    17.2725 -13.136 3.05e-13 ***
## X1           108.2979     6.9420  15.600 4.96e-15 ***
## X2             0.1736     0.0257   6.753 2.99e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 7.566 on 27 degrees of freedom
## Multiple R-squared:  0.927,  Adjusted R-squared:  0.9216
## F-statistic: 171.3 on 2 and 27 DF,  p-value: 4.547e-16
```

The model fit the data pretty well since the multiple R^2 statistic is 0.927 which is pretty high meaning about 92.7% of the variance in Y explained by the predictors X_1 and X_2. Additionally, the intercept and the X_1 and X_2 are significant since their p-values are smaller than 0.001 since there are 3 *'s present next to the p-values, the meaning of this is that all of the predictors are significant when it comes to fitting a model for to predict the response variable Y.

Part b - Non-linear model

```
# create set of degrees to test for best non-linear polynomial model
degrees = list()
k = 1

for (i in 1:5) {
  for (j in 1:5) {
    degrees[[k]] = c(i,j)
    k = k + 1
  }
}

# test each pairing of degrees and compute CV MSE
cv.error = numeric(length(degrees))
for (d in 1:length(degrees)) {
  poly.fit = glm(Y ~ poly(X1, degrees[[d]][1]) + poly(X2, degrees[[d]][2]),
                 data = x1)
  cv.error[d] = cv.glm(x1, poly.fit)$delta[1]
}
cv.error
```

```
##  [1] 64.58769 64.28005 66.84740 61.93174 57.30604 60.82800 56.62764 57.52793
##  [9] 55.75218 51.22781 43.36889 36.21384 34.89581 34.69888 31.05305 48.17719
## [17] 40.06875 38.55668 38.41475 34.16342 49.71149 40.50074 37.61144 37.14079
## [25] 33.51112
```

```
# fit model which gives lowest CV MSE
best_degrees = degrees[[which.min(cv.error)]]
best_degrees
```

```
## [1] 3 5
```

```
poly.fit = lm(Y ~ poly(X1, best_degrees[1]) + poly(X2, best_degrees[2]),
              data=x1)
summary(poly.fit)
```

```
##
## Call:
## lm(formula = Y ~ poly(X1, best_degrees[1]) + poly(X2, best_degrees[2]),
```

```
##     data = x1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.6674 -3.6568  0.6943  2.1935  9.3406
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  62.7447     0.8495  73.863  < 2e-16 ***
## poly(X1, best_degrees[1])1 121.0629     5.3530  22.616 3.17e-16 ***
## poly(X1, best_degrees[1])2  13.6421     4.7698   2.860 0.009373 **
## poly(X1, best_degrees[1])3 -22.1146     5.3194  -4.157 0.000446 ***
## poly(X2, best_degrees[2])1  41.7402     5.3522   7.799 1.24e-07 ***
## poly(X2, best_degrees[2])2 -12.2658     4.7131  -2.602 0.016625 *
## poly(X2, best_degrees[2])3  -8.6564     4.8363  -1.790 0.087908 .
## poly(X2, best_degrees[2])4   7.1616     4.8389   1.480 0.153717
## poly(X2, best_degrees[2])5  10.6914     5.0334   2.124 0.045704 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.653 on 21 degrees of freedom
## Multiple R-squared:  0.9785, Adjusted R-squared:  0.9703
## F-statistic: 119.6 on 8 and 21 DF,  p-value: 9.353e-16
```

I created a list and tested every single combination of degrees for this polynomial regression model using 10-fold cross validation method. The model which gaves the lowest CV MSE was when in the polynomial regression model X_1 was raised to degree 3 and X_2 was raised to degree 5. This non-linear model is better than the linear model obtained in part (a) because now the R^2 statistic is 97.85% so it has by quite a few percentage points, pointing towards it being a better model.

Part c - LOOCV on Model

Question 2

Part a - Generate Data

```
set.seed(1)
n = 50
X1.first = rnorm(25, mean=2, sd=1)
X1.last = rnorm(25, mean=-1, sd=2)
X1 = c(X1.first, X1.last)
X2.first = rnorm(25, mean=10, sd=5)
X2.last = rnorm(25, mean=8, sd=2)
X2 = c(X2.first, X2.last)
df = data.frame(X1, X2)
df
```

```
##            X1         X2
## 1   1.3735462 11.9905294
## 2   2.1836433  6.9398680
## 3   1.1643714 11.7055985
```

```
## 4   3.5952808  4.3531845
## 5   2.3295078 17.1651185
## 6   1.1795316 19.9019995
## 7   2.4874291  8.1638926
## 8   2.7383247  4.7793269
## 9   2.5757814 12.8485981
## 10  1.6946116  9.3247270
## 11  3.5117812 22.0080888
## 12  2.3898432  9.8038000
## 13  1.3787594 13.4486968
## 14 -0.2146999 10.1400108
## 15  3.1249309  6.2836340
## 16  1.9550664 10.9439615
## 17  1.9838097  0.9752069
## 18  2.9438362 17.3277743
## 19  2.8212212 10.7662667
## 20  2.5939013 20.8630584
## 21  2.9189774 12.3775476
## 22  2.7821363  6.4502678
## 23  2.0745650 13.0536318
## 24  0.0106483  5.3295118
## 25  2.6198257  3.7318330
## 26 -1.1122575  8.5828925
## 27 -1.3115910  7.1134163
## 28 -3.9415048  8.0022107
## 29 -1.9563001  8.1486826
## 30 -0.1641169  6.8209581
## 31  1.7173591  6.8626625
## 32 -1.2055755  7.7296428
## 33 -0.2246568 10.3561740
## 34 -1.1076101  4.9528664
## 35 -3.7541191  9.1878924
## 36 -1.8299891  8.6659007
## 37 -1.7885799 10.1261997
## 38 -1.1186268  7.3916322
## 39  1.2000507  8.7400376
## 40  0.5263515  8.5341976
## 41 -1.3290472  6.9149599
## 42 -1.5067234 10.4157356
## 43  0.3939268 10.3208052
## 44  0.1133264  9.4004273
## 45 -2.3775114 11.1736669
## 46 -2.4149903  9.1169729
## 47 -0.2708361  5.4468156
## 48  0.5370658  6.8534692
## 49 -1.2246924  5.5507748
## 50  0.7622155  7.0531987
```

Part b - generate response variable Y

```
Y = c(rep(1, 25), rep(2, 25))
Y = as.factor(Y)
```

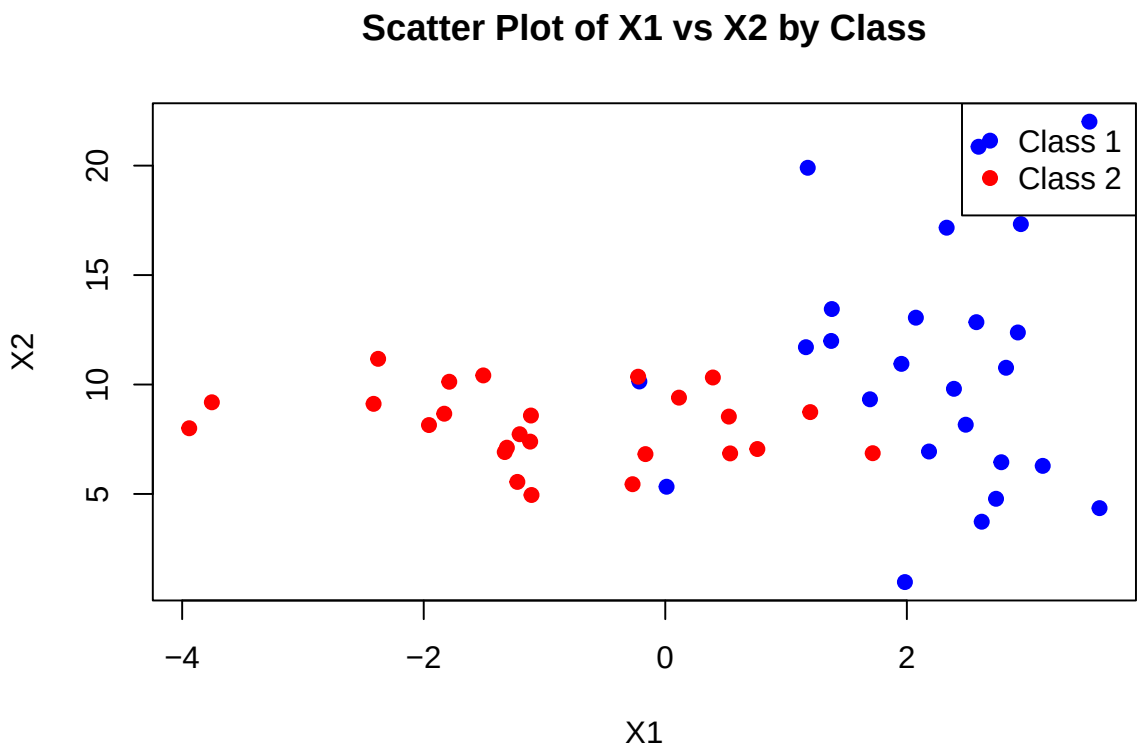
```r
df = data.frame(X1, X2, Y)
df
```

```
##            X1         X2 Y
## 1   1.3735462 11.9905294 1
## 2   2.1836433  6.9398680 1
## 3   1.1643714 11.7055985 1
## 4   3.5952808  4.3531845 1
## 5   2.3295078 17.1651185 1
## 6   1.1795316 19.9019995 1
## 7   2.4874291  8.1638926 1
## 8   2.7383247  4.7793269 1
## 9   2.5757814 12.8485981 1
## 10  1.6946116  9.3247270 1
## 11  3.5117812 22.0080888 1
## 12  2.3898432  9.8038000 1
## 13  1.3787594 13.4486968 1
## 14 -0.2146999 10.1400108 1
## 15  3.1249309  6.2836340 1
## 16  1.9550664 10.9439615 1
## 17  1.9838097  0.9752069 1
## 18  2.9438362 17.3277743 1
## 19  2.8212212 10.7662667 1
## 20  2.5939013 20.8630584 1
## 21  2.9189774 12.3775476 1
## 22  2.7821363  6.4502678 1
## 23  2.0745650 13.0536318 1
## 24  0.0106483  5.3295118 1
## 25  2.6198257  3.7318330 1
## 26 -1.1122575  8.5828925 2
## 27 -1.3115910  7.1134163 2
## 28 -3.9415048  8.0022107 2
## 29 -1.9563001  8.1486826 2
## 30 -0.1641169  6.8209581 2
## 31  1.7173591  6.8626625 2
## 32 -1.2055755  7.7296428 2
## 33 -0.2246568 10.3561740 2
## 34 -1.1076101  4.9528664 2
## 35 -3.7541191  9.1878924 2
## 36 -1.8299891  8.6659007 2
## 37 -1.7885799 10.1261997 2
## 38 -1.1186268  7.3916322 2
## 39  1.2000507  8.7400376 2
## 40  0.5263515  8.5341976 2
## 41 -1.3290472  6.9149599 2
## 42 -1.5067234 10.4157356 2
## 43  0.3939268 10.3208052 2
## 44  0.1133264  9.4004273 2
## 45 -2.3775114 11.1736669 2
## 46 -2.4149903  9.1169729 2
## 47 -0.2708361  5.4468156 2
## 48  0.5370658  6.8534692 2
## 49 -1.2246924  5.5507748 2
```

```
## 50  0.7622155  7.0531987 2
```

Part c - construct classifier for Y

```
# Create the scatter plot
plot(df$X1, df$X2,
     col = ifelse(df$Y == 1, "blue", "red"),
     pch = 19,
     xlab = "X1",
     ylab = "X2",
     main = "Scatter Plot of X1 vs X2 by Class")

# Add legend
legend("topright", legend = c("Class 1", "Class 2"),
       col = c("blue", "red"), pch = 19)
```




```
qda.fit = qda(Y ~ X1 + X2, data=df)
qda.fit
```

```
## Call:
## qda(Y ~ X1 + X2, data = df)
##
## Prior probabilities of groups:
```

```
##   1   2
## 0.5 0.5
##
## Group means:
##           X1        X2
## 1  2.1686652 10.827045
## 2 -0.9355373  8.138488
```

Since the data is generated from two normal distributions we can safely say that the assumption of common covariance matrices is not held. As a result linear discriminant analysis is not a good candidate and quadratic discriminant analysis is the best analysis because of the fact mentioned. So a QDA model was made to classify Y.

Part d - shape of decision boundary

The shape of the decision boundary of the model made in part c is quadratic as it is a quadratic discriminant analysis classification method.

Part e - confusion matrix

```
qda.class = predict(qda.fit, df)$class

# confusion matrix
table(qda.class, df$Y)
```

```
##
## qda.class  1  2
##         1 23  1
##         2  2 24
```

```
# accuracy (correct classification rate of model)
mean(qda.class == df$Y)
```

```
## [1] 0.94
```

The correct classification rate is 94%.

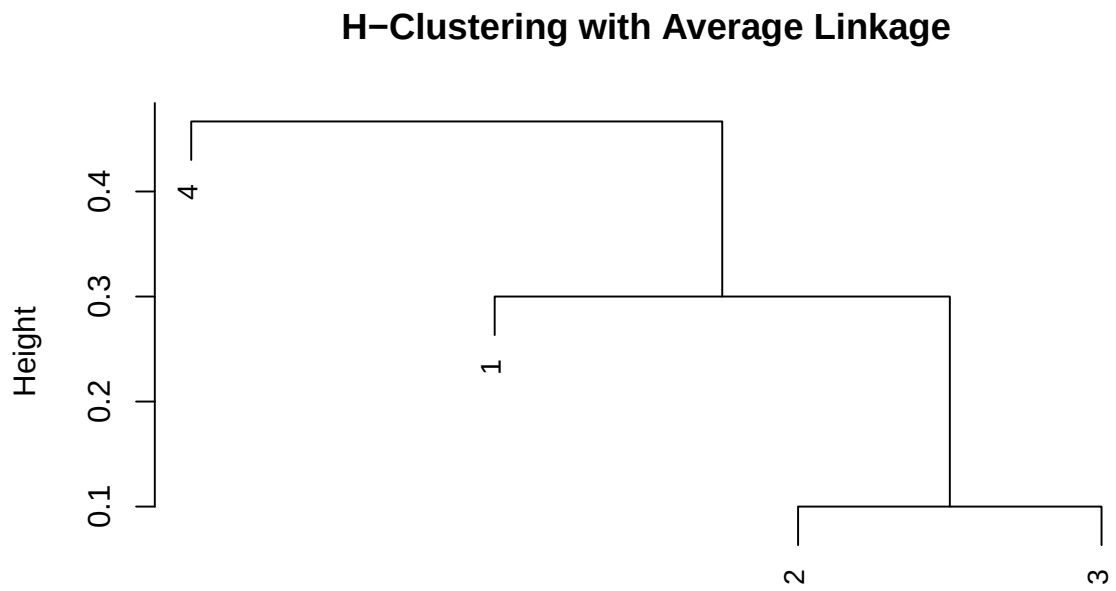
Part f - other classification method analysis

If we construct another model that results in a lower correct classification rate than the one obtained in the model from part (c) we CANNOT say that our model in part (c) is better than the new model. This is because we have not construct train and test datasets to see how well each model performs on unseen data. This is the idea that a low training error rate could result in a high test error rate if the model is tested on data it has seen before as it may overfit, conversely a model with high training error rate could result in a lower test error rate. Overall, we cannot confidently say that if a new model has a lower correct classification rate that it is worse than the model obtained in part (c).

Question 3

Part a - hierarchical clustering dendrogram

```
diss.mat = as.dist(matrix(data = c(0, 0.2, 0.4, 0.6,
                                   0.2, 0, 0.1, 0.5,
                                   0.4, 0.1, 0, 0.3,
                                   0.6, 0.5, 0.3, 0),
                          nrow=4, ncol=4, byrow=TRUE))
hc.avg = hclust(diss.mat, method="average")

plot(hc.avg, main="H-Clustering with Average Linkage", xlab="", sub="", cex=0.9)
```



Part b - Where to cut to get 2 cluster result

If we cut the tree at a height of 0.41 then we will result in 2 clusters, one containing $\{D\}$ and another containing $\{A, C, D\}$.

F Appendix: AAD Assignment 1

Refer to next page.

AAD Assignment 1 - Group 26

Omar, Eloise, Alina, Sue

2025-04-11

Contents

---

1 Descriptive analysis of the data set

1.1 Data loading and cleaning

There are 18640 data points in the original data set. There are 10 features being observed and/or recorded, including the response variable, Median House Value ("medianHouseValue").

```
data <- read.csv("Assignt1_data.csv")
dim(data)
```

```
## [1] 18640    10
```

```
stargazer(data[-1], summary = TRUE, type = "text")
```

```
##
## ===============================================================
## Statistic           N       Mean      St. Dev.     Min      Max
## ---------------------------------------------------------------
## longitude        18,640  -119.569      2.004    -124.350 -114.310
## latitude         18,640   35.630       2.136     32.550   41.950
## housingMedianAge 18,640   28.613      12.606       1        52
## aveRooms         18,640    5.437       2.535     0.846   141.909
## aveBedrooms      18,450    1.097       0.490     0.375    34.067
## population       18,640  1,426.684   1,135.967     3      35,682
## medianIncome     18,640    3.880       1.907     0.500    15.000
## medianHouseValue 18,640 207,241.900 115,651.600  14,999  500,001
## ---------------------------------------------------------------
```

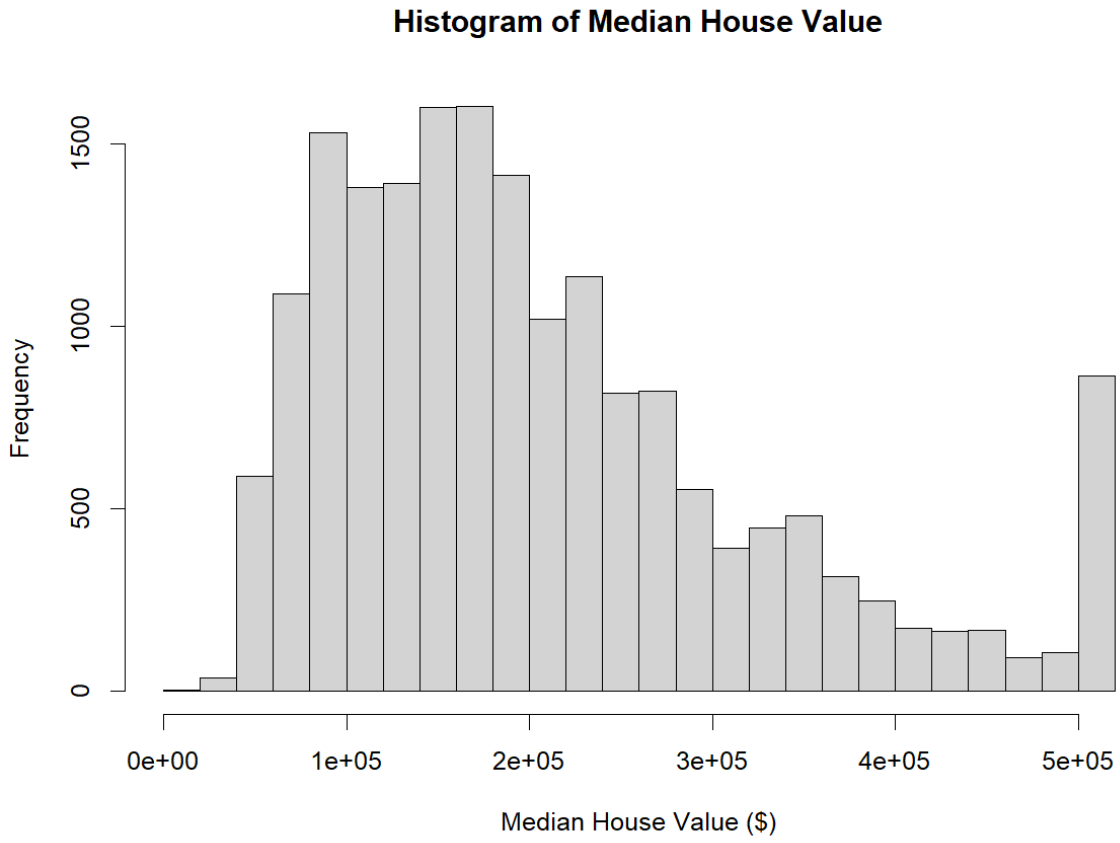
When inspecting the orginal data, we observe that there are 190 rows that have missing values (NA's) in the varaible "aveBedrooms". Since using data points with missing values can lead to biased results, and since 190 out of 18,640 data points is an immaterial number, we decided to remove these rows that contains NA's from the original data frame.

```r
data <- na.omit(data)
rownames(data) <- 1:nrow(data)
```

1.2 Preliminary analysis on the data

1.2.1 Histogram on the response variable and data censoring

```r
hist(data$medianHouseValue,
     xlab = "Median House Value ($)",
     ylab = "Frequency",
     main = "Histogram of Median House Value",
     breaks = 25)
```



From the histogram, it is evident that the medianHouseValue exhibits right-skewness. Interestingly, there is large a concentration of values at the right-end, where the frequency of values $500,000 is disproportionately high. This is likely due to data censoring, where all the medianHouseValue over a certain threshold is recorded at a capped value around $500,000.

Through further inspection we find out this cap for censoring is at $500,001.

```r
value <- max(data$medianHouseValue)
percentage <- mean(data$medianHouseValue == value) * 100
cat("Percentage of data with medianHouseValue =", value, "is", percentage, "%\n")
```

```
## Percentage of data with medianHouseValue = 500001 is 4.688347 %
```

```r
summary(data$medianHouseValue)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   14999  119800  180000  207277  265600  500001
```

Ultimately, given the max. value is 500,001 and the fact that the frequency of this exact value is disproportionately high (4.7%), we are led to conclude that 500,001 is the censoring cap applied. This is likely due to the data collection methods used, where houses valued above $500,000 were simply reported as exactly $500,001. The impact of this is that the data may under-represent the true values and variation in higher-value homes. Moving forward, we have decided to keep these censored observations in but will maintain caution - especially in our interpretation of correlations, and coming to terms with the fact that our model will not be able to provide reliable predictions for values above 500,001.

Other censoring

```r
summary(data$medianIncome)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.4999  2.5669  3.5446  3.8803  4.7608 15.0001
```

```r
value <- max(data$medianIncome)
percentage <- mean(data$medianIncome == value) * 100
cat("Percentage of data with medianIncome =", value, "is", percentage, "%\n")
```

```
## Percentage of data with medianIncome = 15.0001 is 0.2384824 %
```

It's clear that medianIncome is also censored. Both medianIncome and medianHouseValue are left censored and right censored. Since it is more significant, for this report we will focus on right censoring on medianHouseValue.

1.2.2 Correlations among variables

```r
numeric.data <-  data[sapply(data, is.numeric)]
head(numeric.data)
```

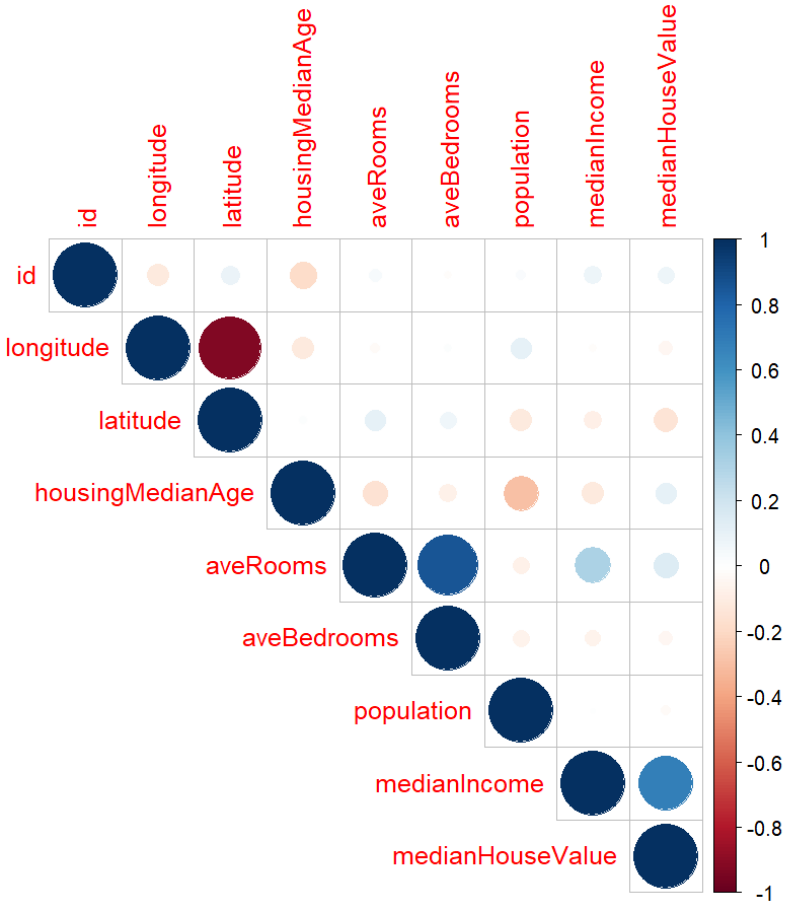
```
##   id longitude latitude housingMedianAge aveRooms aveBedrooms population
## 1  1   -122.23    37.88              41 6.984127   1.0238095        322
## 2  2   -122.22    37.86              21 6.238137   0.9718805       2401
## 3  3   -122.24    37.85              52 8.288136   1.0734463        496
## 4  4   -122.25    37.85              52 5.817352   1.0730594        558
## 5  5   -122.25    37.85              52 6.281853   1.0810811        565
## 6  6   -122.25    37.85              52 4.761658   1.1036269        413
##   medianIncome medianHouseValue
```

```
## 1       8.3252           452600
## 2       8.3014           358500
## 3       7.2574           352100
## 4       5.6431           341300
## 5       3.8462           342200
## 6       4.0368           269700
```

```
cor_medianHouseValue <- cor(data$medianHouseValue,
                            data[, c("longitude",
                                     "latitude",
                                     "housingMedianAge",
                                     "aveRooms",
                                     "aveBedrooms",
                                     "population",
                                     "medianIncome")])
print(cor_medianHouseValue)
```

```
##        longitude   latitude housingMedianAge aveRooms aveBedrooms  population
## [1,] -0.04646254 -0.1439446        0.1052129 0.148625 -0.04545692 -0.02394206
##      medianIncome
## [1,]     0.689536
```

```
corrplot(cor(data[, sapply(data, is.numeric)]), method = "circle",type =
         "upper")
```



From the correlation matrix plot we can see that among all the predictors, medianHouseValue has highest correlation with Median Income. This is pretty intuitive, as we expect high-income households purchasing more expensive houses.

Longitude and Latitude are highly negatively correlated, while Average Rooms and Average Bedrooms are highly positively correlated. This multicollinearity can lead to issues such as unstable coefficient estimates and inflated standard errors, as the model struggles to distinguish the individual effects of highly correlated predictors. Consequently, the interpretability of the regression results is compromised. We will address this issue later in section 2.4 Model improvements.

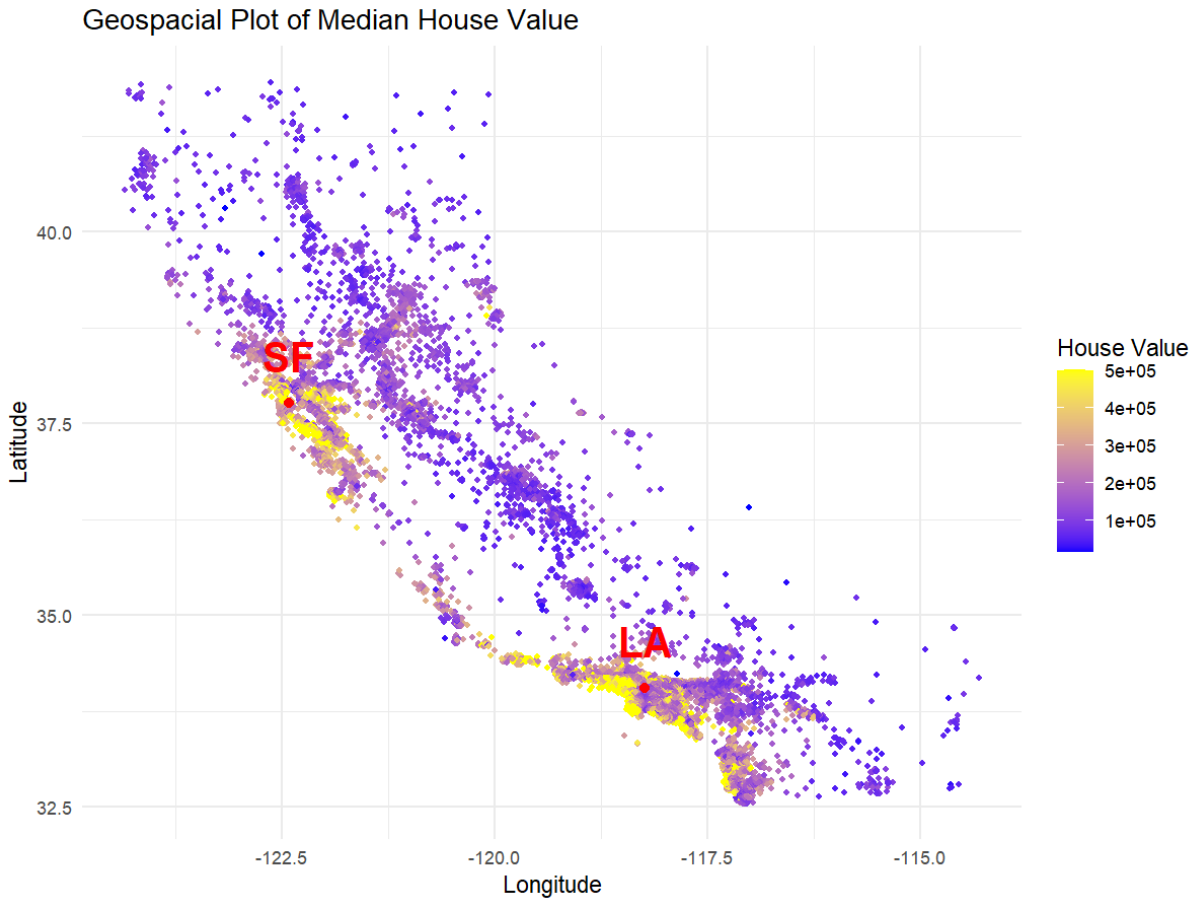
Thee rest of the correlations between the remaining variables with medianHouseValue have a magnitude below 0.3. However, they may still be very useful in predicting the response variable. But we do not give detailed discussion here.

Note that ID is just for labeling purposes and it is not a relevant predictor, so we don't care about the correlation between ID and other variables.

1.2.3 Geospacial plot of medianHouseValue

Housing values can vary based on their proximity to major cities. By analysing the provided longitude and latitude, we can deduce that the dataset originates from California. To visualise the relative variation in median house values, we create a heatmap that highlights the influence of major cities, specifically Los Angeles (LA) and San Francisco (SF).

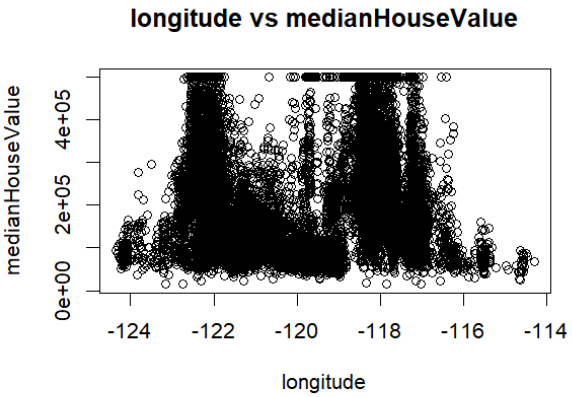
```r
cities <- data.frame(
  city = c("LA", "SF"),
  longitude = c(-118.24, -122.42),
  latitude = c(34.05, 37.77)
)

ggplot(data,
       aes(x = longitude,
           y = latitude,
           colour = medianHouseValue)) +
  geom_point(size = 1) +
  scale_colour_gradient(low = "blue", high = "yellow") +
  geom_point(data = cities,
             aes(x = longitude, y = latitude),
             colour = "red",
             size = 2) +
  geom_text(data = cities,
            aes(label = city),
            vjust = -1,
            colour = "red",
            size = 7,
            fontface = "bold") +
  labs(title = "Geospacial Plot of Median House Value",
       x = "Longitude",
       y = "Latitude",
       colour = "House Value") +
  theme_minimal()
```

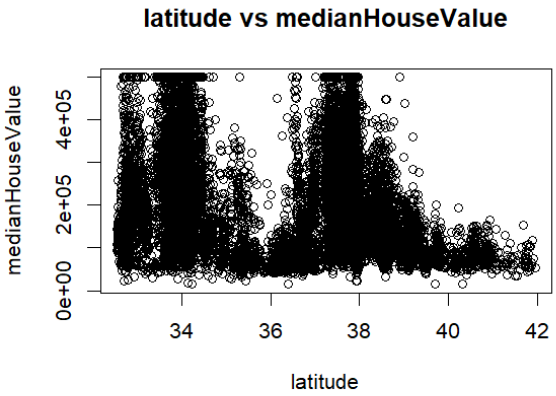


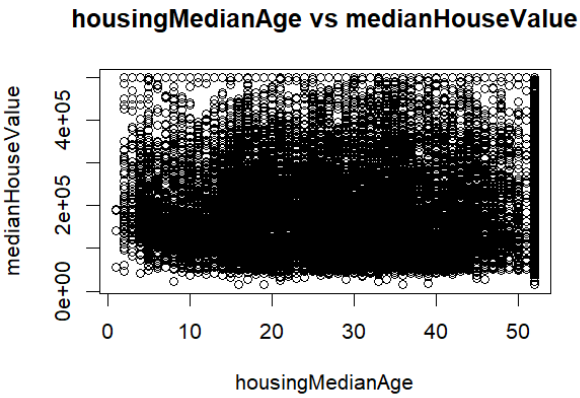
Clearly, median housing value is higher in the SF and LA regions. Therefore, it is reasonable to integrate a region's closeness to these major cities in our regression model later. They are also an appropriate proxy for censoring.

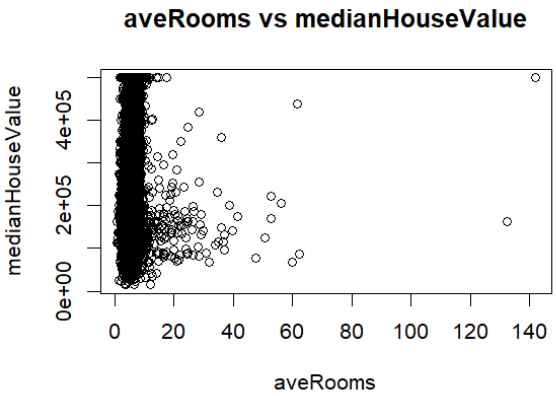
1.2.4 Plots of the response variable vs different features

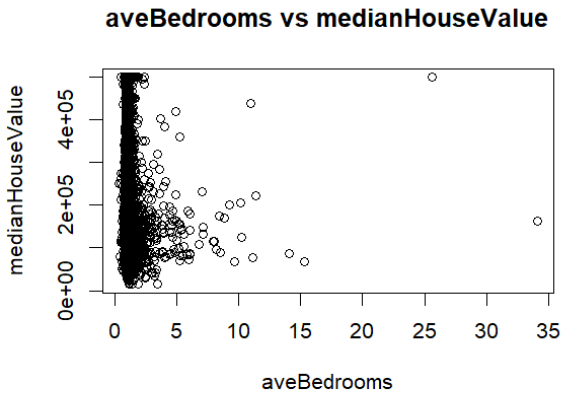
```
par(mfrow = c(2, 2))  # 2 rows, 2 columns of plots per "page"
target_col <- "medianHouseValue"
exclude <- c("id","oceanProximity",target_col)

for (col in names(data)) {
  if (!(col %in% exclude)) {
    plot(data[[col]], data[[target_col]],
         xlab = col,
         ylab = target_col,
         main = paste(col, "vs", target_col))
  }
}
```

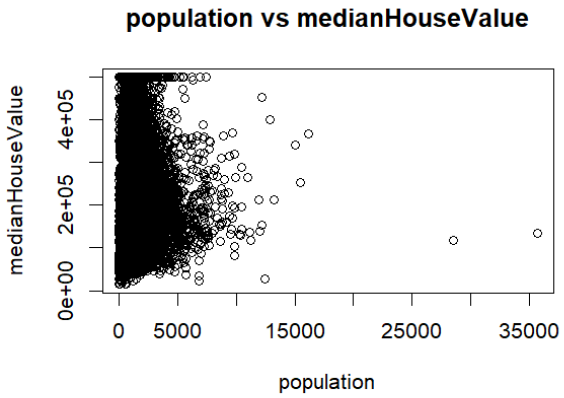

longitude vs medianHouseValue
medianHouseValue
0e+00
2e+05
4e+05
-124
-122
-120
-118
-116
-114
longitude

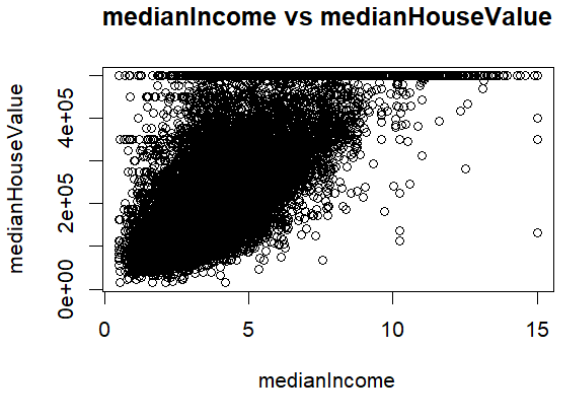

latitude vs medianHouseValue
medianHouseValue
0e+00
2e+05
4e+05
34
36
38
40
42
latitude


housingMedianAge vs medianHouseValue
medianHouseValue
0e+00
2e+05
4e+05
0
10
20
30
40
50
housingMedianAge


aveRooms vs medianHouseValue
medianHouseValue
0e+00
2e+05
4e+05
0
20
40
60
80
100
120
140
aveRooms





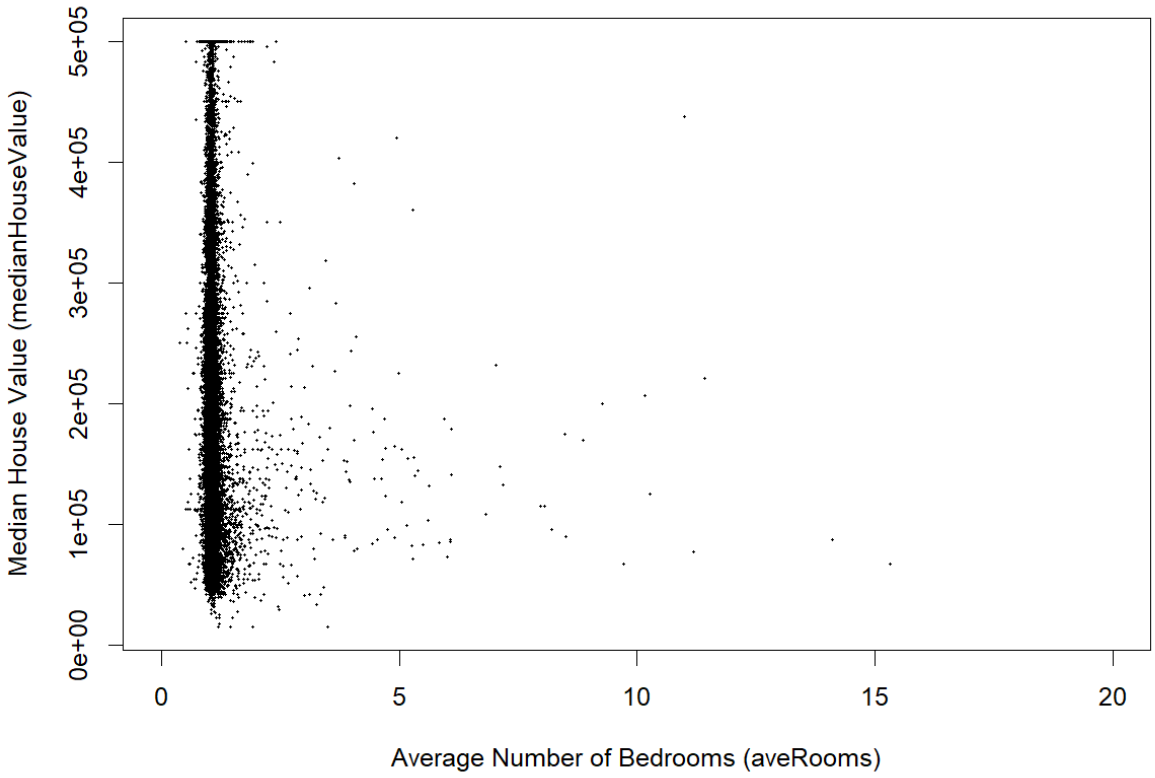


Median House Value vs Longitude/Latitude: there is no clear linear relationship or overall trend. But we can see that for some specific longtitude/latitude the housing value is higher. This is consistent with our findings in the geospacial plot - the peaks in the above plot has to do with the relative location to major cities, SF and LA. This motivates us to do some transformation on the raw longitude and latitude data while integrating relevant information in the MLR framwork later on.

Median House Value vs Housing Media Age: No clear pattern again just by looking at the plot. However the high density of data on the rightmost band (representing housingMedianAge = 52) shows another sign of data censoring.

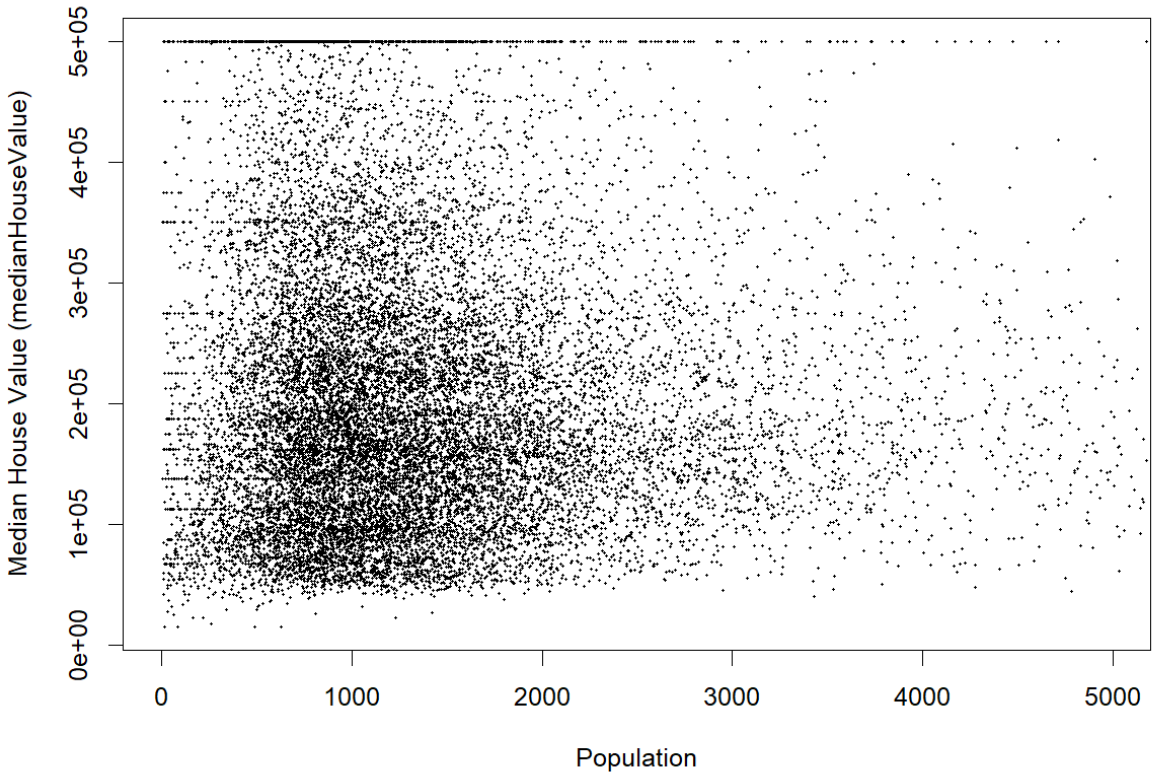
Median House Value vs Average Rooms / Average Bedrooms: The linear pattern is not clear when we plot against the entire data range for rooms. These two plots exposes some issue of high leverage points - some regions have average number of bedrooms over 30 and average number of rooms over 140. These abnormalities are worthy to be investigated and treated when we fit the MLR.

```
# Focus on the range of [0, 20] for aveRooms
plot(data$aveBedrooms, data$medianHouseValue,
     xlim = c(0, 20),
     pch = 16,
     xlab = "Average Number of Bedrooms (aveRooms)",
     ylab = "Median House Value (medianHouseValue)",
     cex = 0.3)
```



Median House Value vs Population: The relationship appear non-linear even if we zoom in to the population data range 0 to 5000:

```
plot(data$population, data$medianHouseValue,
    xlim = c(0, 5000),
     pch = 16,
     xlab = "Population",
     ylab = "Median House Value (medianHouseValue)",
     cex = 0.3)
```



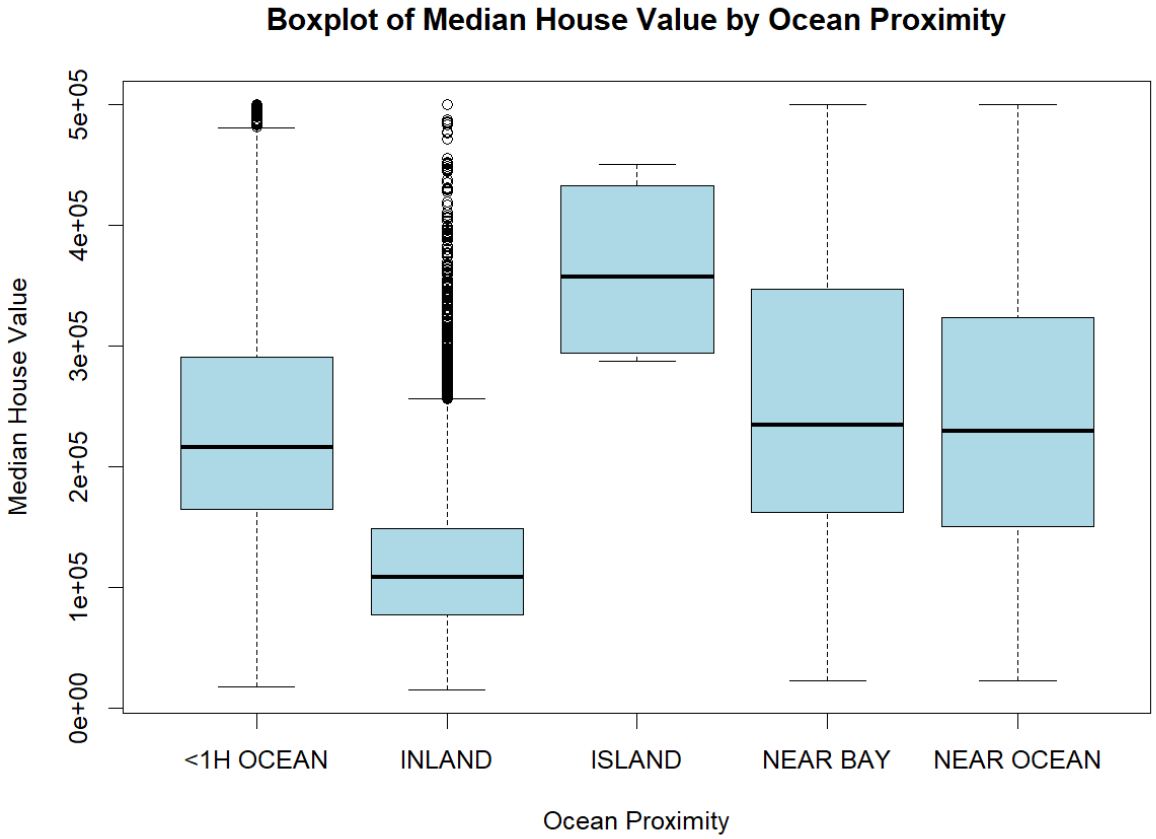
Median House Value vs Median Income: We can see some clear positive trend. This resonates with the correlation plot result, where these two features are positively correlated.

For those pairs where linear relationship is unclear, we will explore log transformations and polynomials in section 1.2.6 single linear regression.

1.2.5 Boxplot of Median House Value by Ocean Proximity

For the qualitative variable, Ocean Proximity, we can examine the differences across different classes via a boxplot:

```
boxplot(medianHouseValue ~ oceanProximity, data = data,
        main = "Boxplot of Median House Value by Ocean Proximity",
        xlab = "Ocean Proximity",
        ylab = "Median House Value",
        col = "lightblue")
```

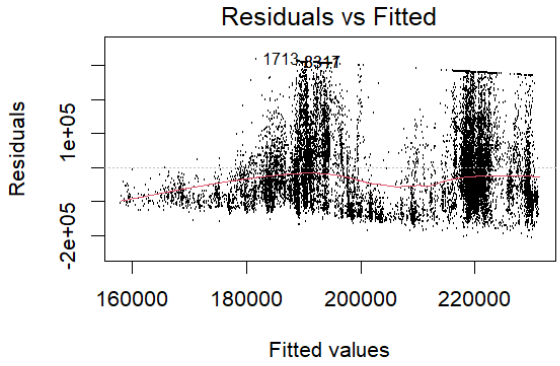


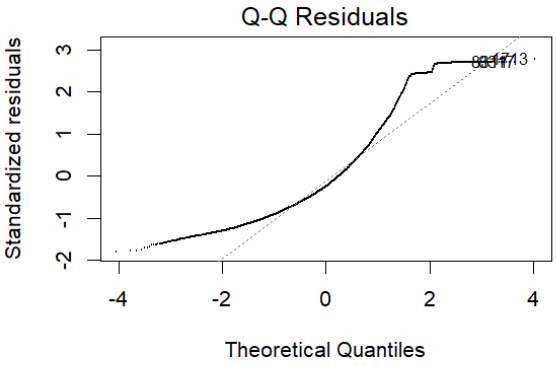
From the plot, island housing appears to be most valuable on average, whereas inland housing appears to be least valuable. The median value for other 3 classes (<1h ocean, near bay and near ocean) seems to be close to each other.

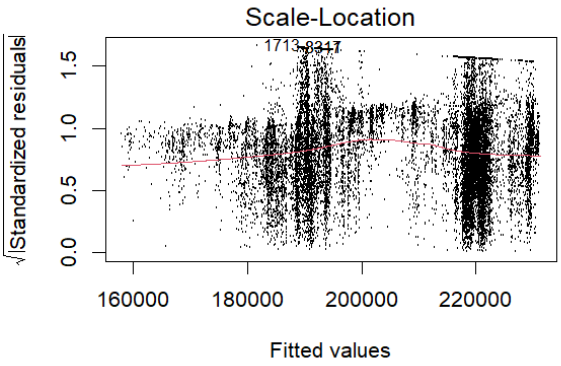
1.2.6 Non-linear considerations

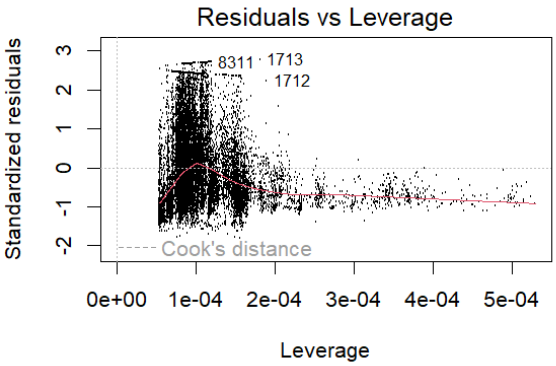
By performing single linear regressions, we find that individual pair of relationships are all statistically significant. However, some assumptions necessary for OLS are clearly violated (e.g. heteroscedasiticty, normality), and some relationship are clearly non-linear. We therefore consider polynomial and log transformations for the following pairwise relationships:

```
lm.latitude = lm(medianHouseValue ~ latitude, data = data)
lm.longitude = lm(medianHouseValue ~ longitude, data = data)
par(mfrow = c(2,2))
plot(lm.latitude, cex = 0.1)
```

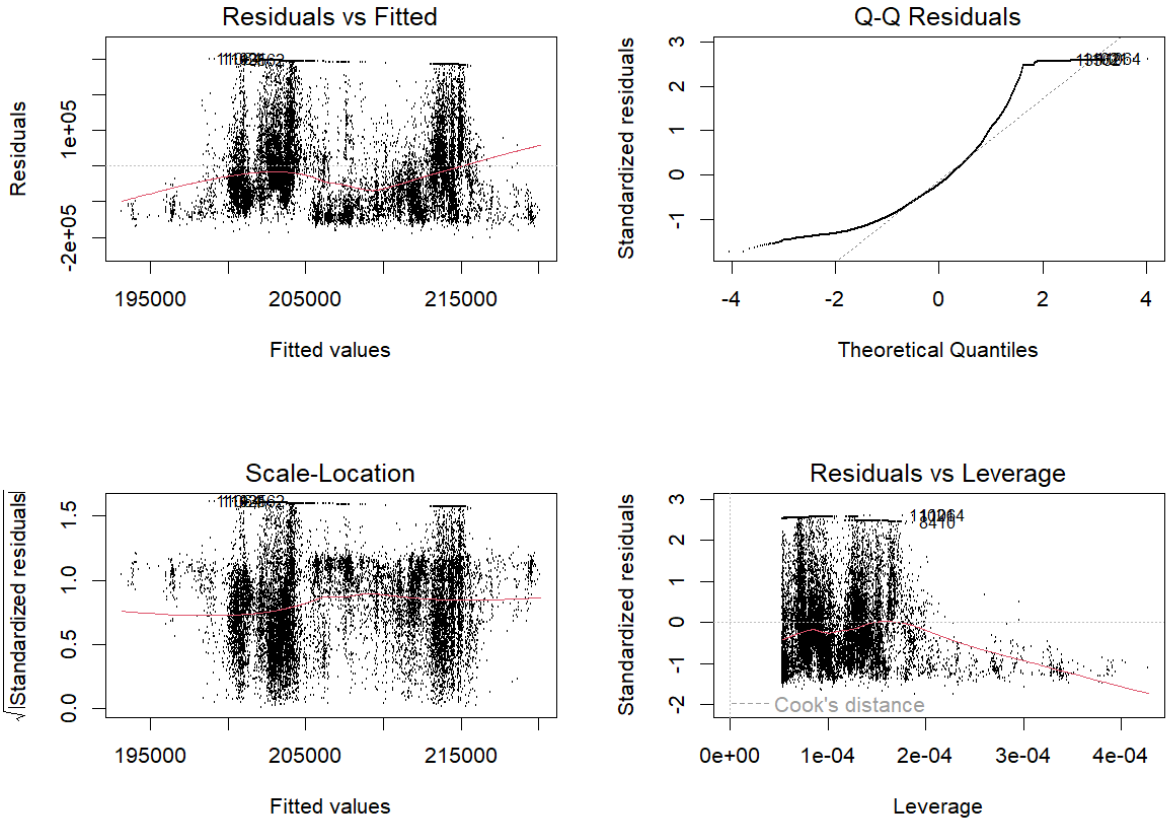









```
plot(lm.longitude, cex = 0.1)
```



Single regression of medianHouseValue on longitude and latitude shows clear non-linearity. This is evident from the residual plot, where the red reference line shows curved patterns.

```
# logitude and latitude up to degree 3:
lm.latitude.poly2 = lm(medianHouseValue ~ poly(latitude, 2, raw = T), data = data)
lm.latitude.poly3 = lm(medianHouseValue ~ poly(latitude, 3, raw = T), data = data)
lm.longitude.poly2 = lm(medianHouseValue ~ poly(longitude, 2, raw = T), data = data)
lm.longitude.poly3 = lm(medianHouseValue ~ poly(longitude, 3, raw = T), data = data)
# par(mfrow = c(2,2))
# plot(lm.latitude, cex = 0.1)
# plot(lm.latitude.poly2, cex = 0.1)
# plot(lm.latitude.poly3, cex = 0.1)

anova(lm.latitude, lm.latitude.poly2, lm.latitude.poly3)
```

```
## Analysis of Variance Table
##
## Model 1: medianHouseValue ~ latitude
## Model 2: medianHouseValue ~ poly(latitude, 2, raw = T)
## Model 3: medianHouseValue ~ poly(latitude, 3, raw = T)
##   Res.Df        RSS Df  Sum of Sq       F    Pr(>F)
## 1  18448 2.4188e+14
## 2  18447 2.3566e+14  1 6.2128e+12 486.911 < 2.2e-16 ***
## 3  18446 2.3537e+14  1 2.9719e+11  23.291 1.403e-06 ***
```

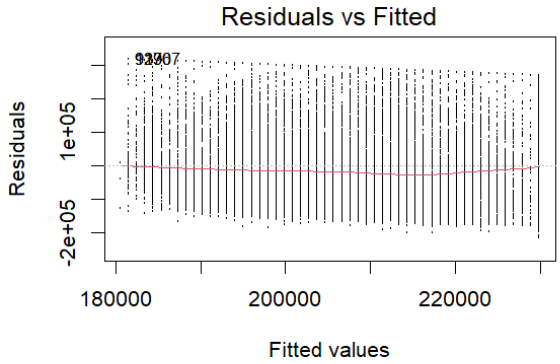
```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

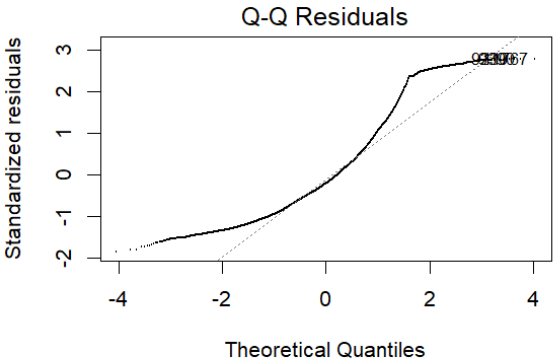
```
anova(lm.longitude, lm.longitude.poly2, lm.longitude.poly3)
```

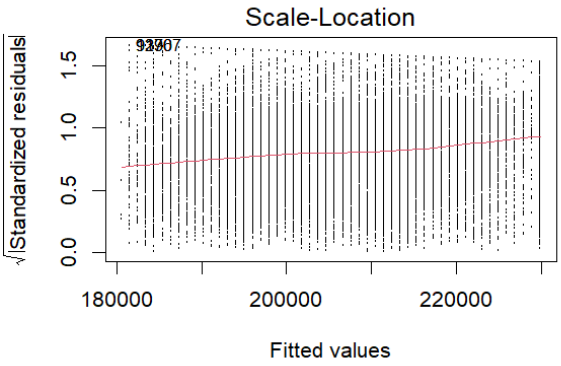
```
## Analysis of Variance Table
##
## Model 1: medianHouseValue ~ longitude
## Model 2: medianHouseValue ~ poly(longitude, 2, raw = T)
## Model 3: medianHouseValue ~ poly(longitude, 3, raw = T)
##   Res.Df        RSS Df  Sum of Sq       F    Pr(>F)
## 1  18448 2.4646e+14
## 2  18447 2.4623e+14  1 2.3032e+11  17.547 2.815e-05 ***
## 3  18446 2.4212e+14  1 4.1097e+12 313.102 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

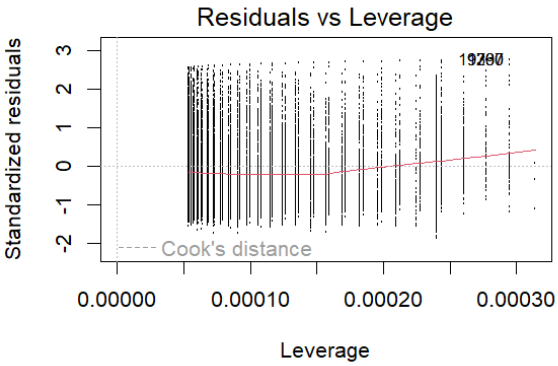
By fitting a polynomial regression up to degree 3, we observe an improvement in Adjusted R^2, and the residual plot pattern also improves. The ANOVA test further supports a significant model improvement. To avoid overfitting, we choose not to explore higher-degree polynomials.

```
lm.housingMedianAge <- lm(medianHouseValue ~ housingMedianAge, data = data)
lm.housingMedianAge.poly2 <- lm(medianHouseValue
                                ~ poly(housingMedianAge, 2, raw = T),
                                data = data)

par(mfrow = c(2,2)); plot(lm.housingMedianAge, cex = 0.1)
```

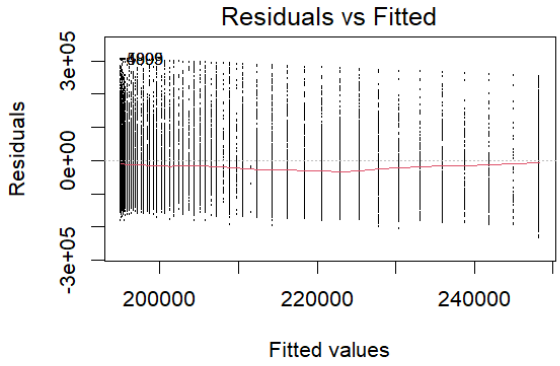


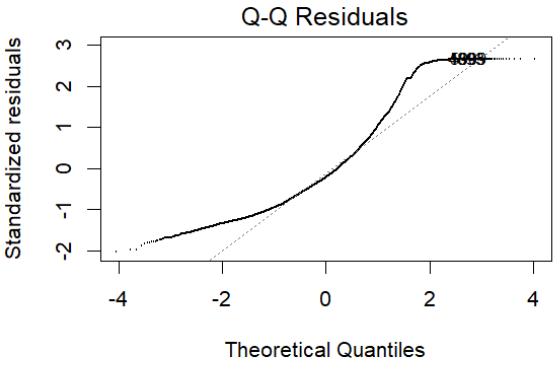


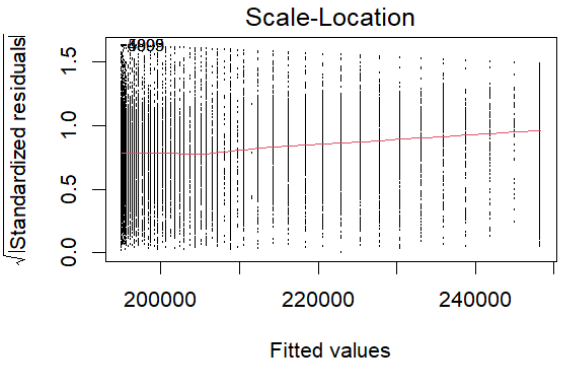


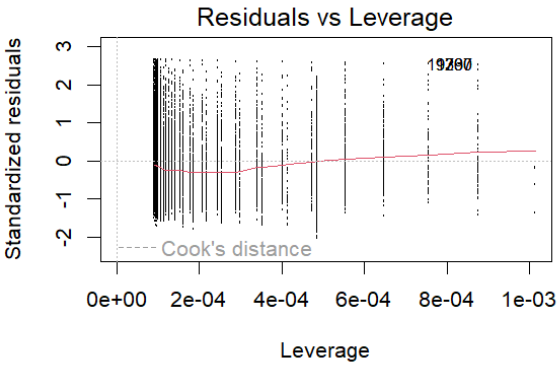



```
par(mfrow = c(2,2)); plot(lm.housingMedianAge.poly2, cex = 0.1)
```










```
summary(lm.housingMedianAge)
```

```
##
## Call:
## lm(formula = medianHouseValue ~ housingMedianAge, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -214856  -85248  -25850   58318  318407
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)      179663.30    2100.05   85.55   <2e-16 ***
## housingMedianAge    965.22      67.17   14.37   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 115100 on 18448 degrees of freedom
## Multiple R-squared:  0.01107,    Adjusted R-squared:  0.01102
## F-statistic: 206.5 on 1 and 18448 DF,  p-value: < 2.2e-16
```

```r
summary(lm.housingMedianAge.poly2)
```

```
##
## Call:
## lm(formula = medianHouseValue ~ poly(housingMedianAge, 2, raw = T),
##     data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -233248  -85747  -25328   59177  304946
##
## Coefficients:
##                                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)                       213548.81    3919.03   54.49  < 2e-16 ***
## poly(housingMedianAge, 2, raw = T)1  -1917.96     289.70   -6.62 3.68e-11 ***
## poly(housingMedianAge, 2, raw = T)2     49.72       4.86   10.23  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 114700 on 18447 degrees of freedom
## Multiple R-squared:  0.01665,    Adjusted R-squared:  0.01654
## F-statistic: 156.1 on 2 and 18447 DF,  p-value: < 2.2e-16
```

```r
anova(lm.housingMedianAge,
      lm.housingMedianAge.poly2)
```

```
## Analysis of Variance Table
##
## Model 1: medianHouseValue ~ housingMedianAge
## Model 2: medianHouseValue ~ poly(housingMedianAge, 2, raw = T)
##   Res.Df        RSS Df  Sum of Sq      F    Pr(>F)
## 1  18448 2.4426e+14
## 2  18447 2.4288e+14  1 1.3777e+12 104.64 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The normality assumption appears violated for housingMedianAge, as seen in the QQ plot. Although the residual pattern remains largely unchanged, the second-degree polynomial model improves Adjusted R^2 (from 0.011 to 0.017) and yields a statistically significant improvement in fit (ANOVA p < 2.2e-16). Therefore, we include the degree-2 polynomial in the model selection stage.

```r
lm.medianIncome <- lm(medianHouseValue ~ medianIncome, data = data)
# summary(lm.medianIncome)
# par(mfrow = c(2,2)); plot(lm.medianIncome, cex = 0.2)

lm.medianIncome.poly2 <- lm(medianHouseValue ~ poly(medianIncome, 2, raw = T),
                            data = data)
# summary(lm.medianIncome.poly2)
# par(mfrow = c(2,2)); plot(lm.medianIncome.poly2, cex = 0.2)

lm.medianIncome.poly3 <- lm(medianHouseValue ~ poly(medianIncome, 3, raw = T),
                            data = data)
```

```r
# summary(lm.medianIncome.poly3)
# par(mfrow = c(2,2)); plot(lm.medianIncome.poly3, cex = 0.2)

lm.medianIncome.poly4 <- lm(medianHouseValue ~ poly(medianIncome, 4, raw = T),
                            data = data)
anova(lm.medianIncome,
      lm.medianIncome.poly2,
      lm.medianIncome.poly3,
      lm.medianIncome.poly4)
```

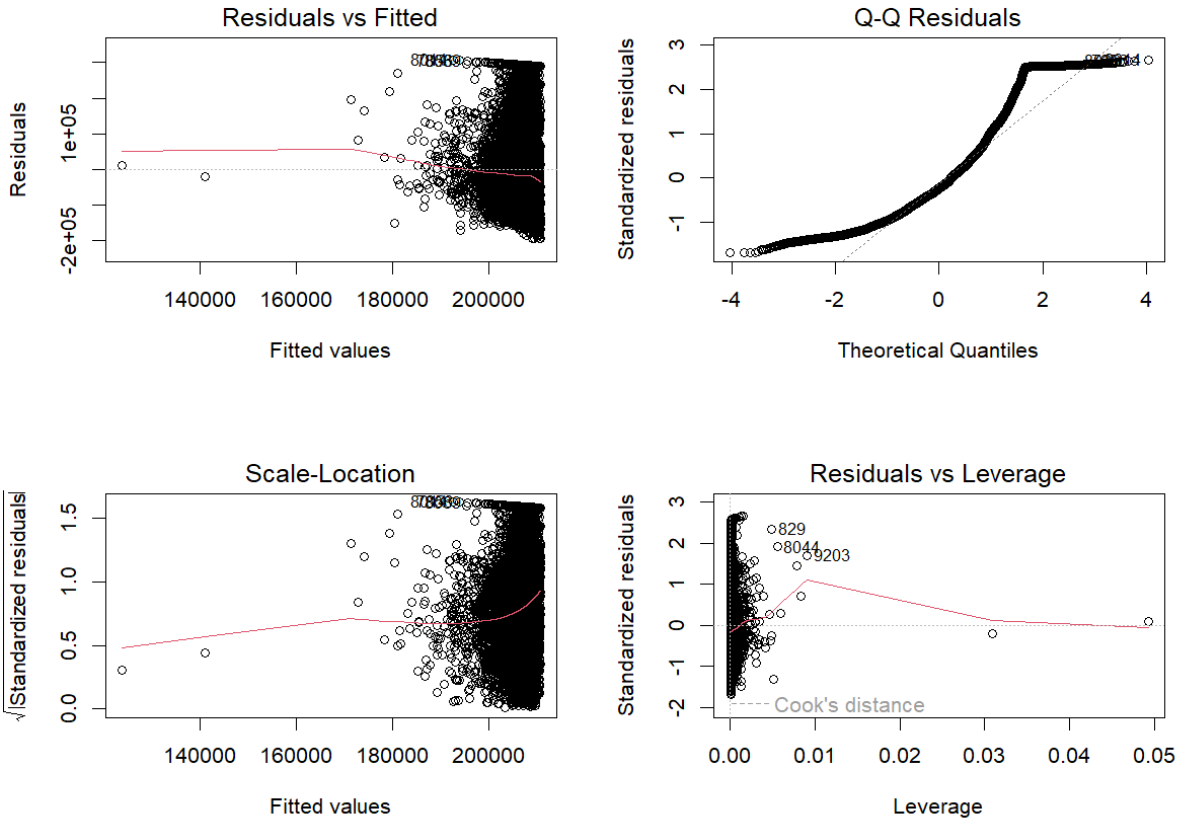
```
## Analysis of Variance Table
##
## Model 1: medianHouseValue ~ medianIncome
## Model 2: medianHouseValue ~ poly(medianIncome, 2, raw = T)
## Model 3: medianHouseValue ~ poly(medianIncome, 3, raw = T)
## Model 4: medianHouseValue ~ poly(medianIncome, 4, raw = T)
##   Res.Df        RSS Df  Sum of Sq        F  Pr(>F)
## 1  18448 1.2956e+14
## 2  18447 1.2844e+14  1 1.1210e+12 163.6184 < 2e-16 ***
## 3  18446 1.2639e+14  1 2.0433e+12 298.2294 < 2e-16 ***
## 4  18445 1.2637e+14  1 2.1319e+10   3.1117 0.07775 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Polynomial regression of medianHouseValue on medianIncome up to degree 3 is also shown to have a better fit than lower degree polynomial as well as linear model. In particular, with the evidence from the anova F-test, we conclude that degree-3 is significantly better than degree-2, whereas moving to higher degree (i.e. quartic) does not bring additional significant benefit.

```r
lm.pop <- lm(medianHouseValue ~ population, data = data)
summary(lm.pop)
```

```
##
## Call:
## lm(formula = medianHouseValue ~ population, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -195709  -87189  -26995   58504  307357
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.108e+05  1.366e+03 154.254  < 2e-16 ***
## population  -2.437e+00  7.492e-01  -3.253  0.00114 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 115700 on 18448 degrees of freedom
## Multiple R-squared:  0.0005732,  Adjusted R-squared:  0.000519
## F-statistic: 10.58 on 1 and 18448 DF,  p-value: 0.001145
```

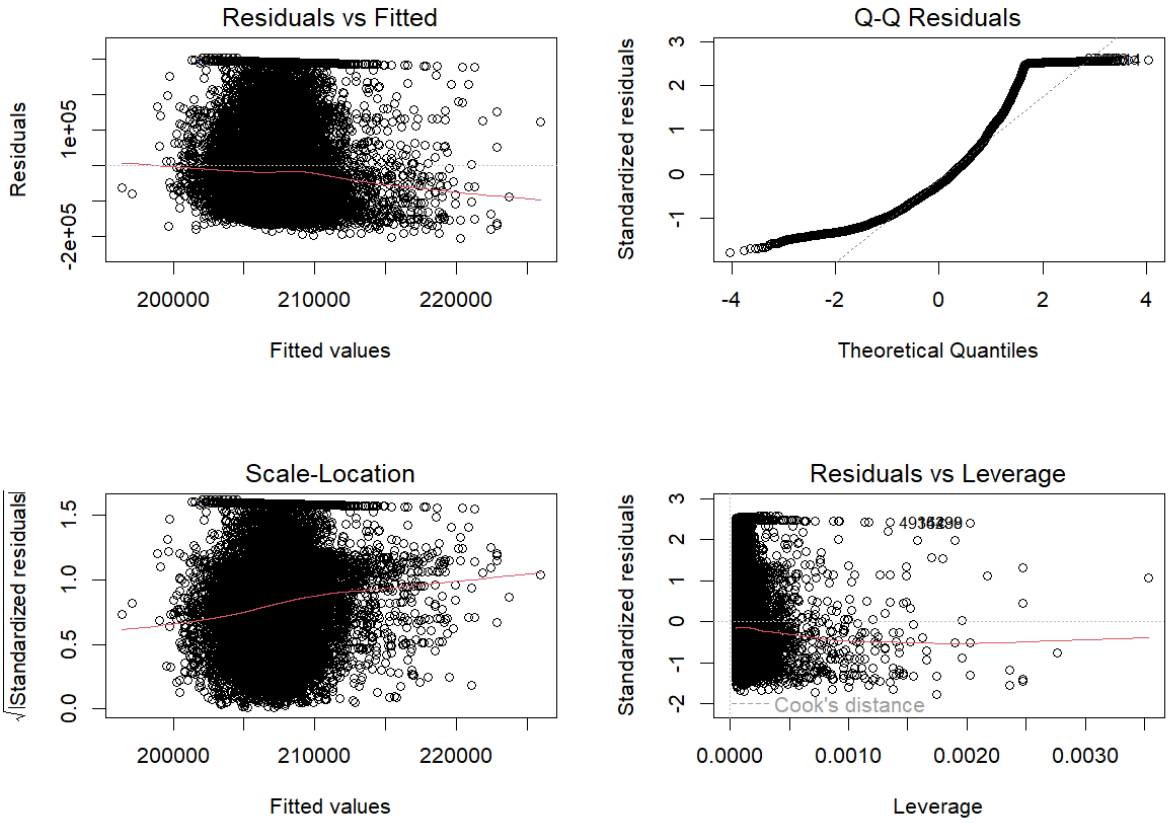
```r
par(mfrow = c(2,2)); plot(lm.pop)
```




```r
lm.log.pop <- lm(medianHouseValue ~ log(population), data = data)
summary(lm.log.pop)
```

```
##
## Call:
## lm(formula = medianHouseValue ~ log(population), data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -205287  -87246  -27031   58539  298671
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       229383       8132  28.209  < 2e-16 ***
## log(population)    -3147       1151  -2.734  0.00627 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 115700 on 18448 degrees of freedom
## Multiple R-squared:  0.0004049,  Adjusted R-squared:  0.0003507
## F-statistic: 7.472 on 1 and 18448 DF,  p-value: 0.006272
```

```r
par(mfrow = c(2,2)); plot(lm.log.pop)
```



The predictor "population" suffers from high leverage point and heteroscedasticity. To address this issue, a log transform is applied to the original data. From the pre-log-transform residual plot, the residuals are clustered at one end, whereas the post-log plot show better random scattering. This indicates that a log transform improves linearity between the response and the predictor, which motivates us to implement this improvement in our final model.

---

2 Multiple linear regression

2.1 Initial MLR model using all appropriate predictors

We will be using the all the given predictors except for id, since it is irrelevant.

```r
# Initial_data is a copy of data
initial_data <- read.csv("Assignt1_data.csv")
initial_data <- na.omit(initial_data)

initial_fit <- lm(medianHouseValue ~ . - id, data = initial_data)
summary(initial_fit)
```

```
##
## Call:
## lm(formula = medianHouseValue ~ . - id, data = initial_data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -632258  -45300  -11782   30245  443819
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)              -2.272e+06  9.759e+04 -23.278  < 2e-16 ***
## longitude                -2.655e+04  1.131e+03 -23.473  < 2e-16 ***
## latitude                 -2.473e+04  1.119e+03 -22.103  < 2e-16 ***
## housingMedianAge          8.205e+02  4.761e+01  17.235  < 2e-16 ***
## aveRooms                 -8.784e+03  6.219e+02 -14.126  < 2e-16 ***
## aveBedrooms               5.338e+04  2.973e+03  17.956  < 2e-16 ***
## population               -6.741e-01  4.942e-01  -1.364 0.172593
## medianIncome              4.209e+04  4.471e+02  94.142  < 2e-16 ***
## oceanProximityINLAND     -3.845e+04  1.931e+03 -19.906  < 2e-16 ***
## oceanProximityISLAND      1.290e+05  3.586e+04   3.597 0.000322 ***
## oceanProximityNEAR BAY    4.059e+03  2.090e+03   1.942 0.052115 .
## oceanProximityNEAR OCEAN  8.919e+03  1.714e+03   5.204 1.97e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 71650 on 18438 degrees of freedom
## Multiple R-squared:  0.6168, Adjusted R-squared:  0.6166
## F-statistic:  2698 on 11 and 18438 DF,  p-value: < 2.2e-16
```

2.2 Discussion of the initial model

The p-value of the F-test is practically zero, therefore we have sufficient evidence to reject the null hypothesis that all regression coefficients are zero. This means that the model has overall significance, and that at least one of the predictors are useful in explaining the variation in medianHouseValue.

Having concluded that this model (initial_fit) has overall significance, we can gauge the significance of individual predictors through the t-test p-values: At 5% level of significance, we cannot say that population is useful, as we do not have sufficient evidence to reject the null hypothesis that its corresponding coefficient is non-zero. The dummy variable, "NEAR BAY", associated to the categorical predictor, "oceanProximity", is also shown to be insignificant. This might be an indication that comparing to the base case (<1H OCEAN), NEAR BAY does not lead to significant change in Median House Price. A potential real-world explanation of why nearbay might not be influential is that "bay" is a very broad concept; some could be more favourable than others (eg. close to amenities OR prone to environmental issues) so its hard to deduce a clear relationship between proximity to bay and house value.

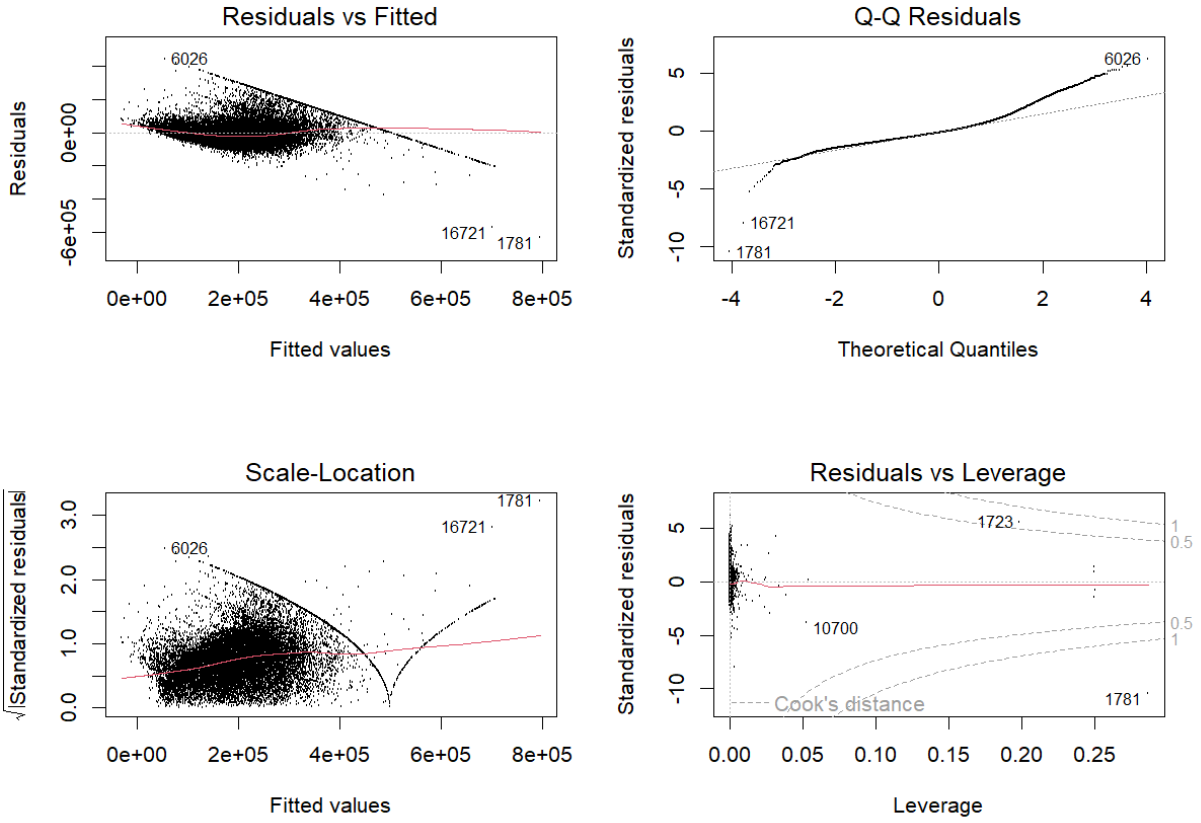
Moreover, the R-squared and adjusted R-squared both equal to 0.617 when corrected to 3 decimal places. This means that 61.7% of the variation in the predictors are useful in explaining the variation in the response variable (medianHouseValue).

Additionally, this initial model also reveals some clear relationships among the varaibles: Median income has a large, positive coefficient so its a strong positive predictor as we anticipated from the preliminary analysis.

longitude and latitude both have large negative coefficients which reiterates our findings before regarding geographic (major cities) relevance.

2.3 Checking issues in the initial model

```
par(mfrow = c(2,2))
plot(initial_fit, cex = 0.1)
```



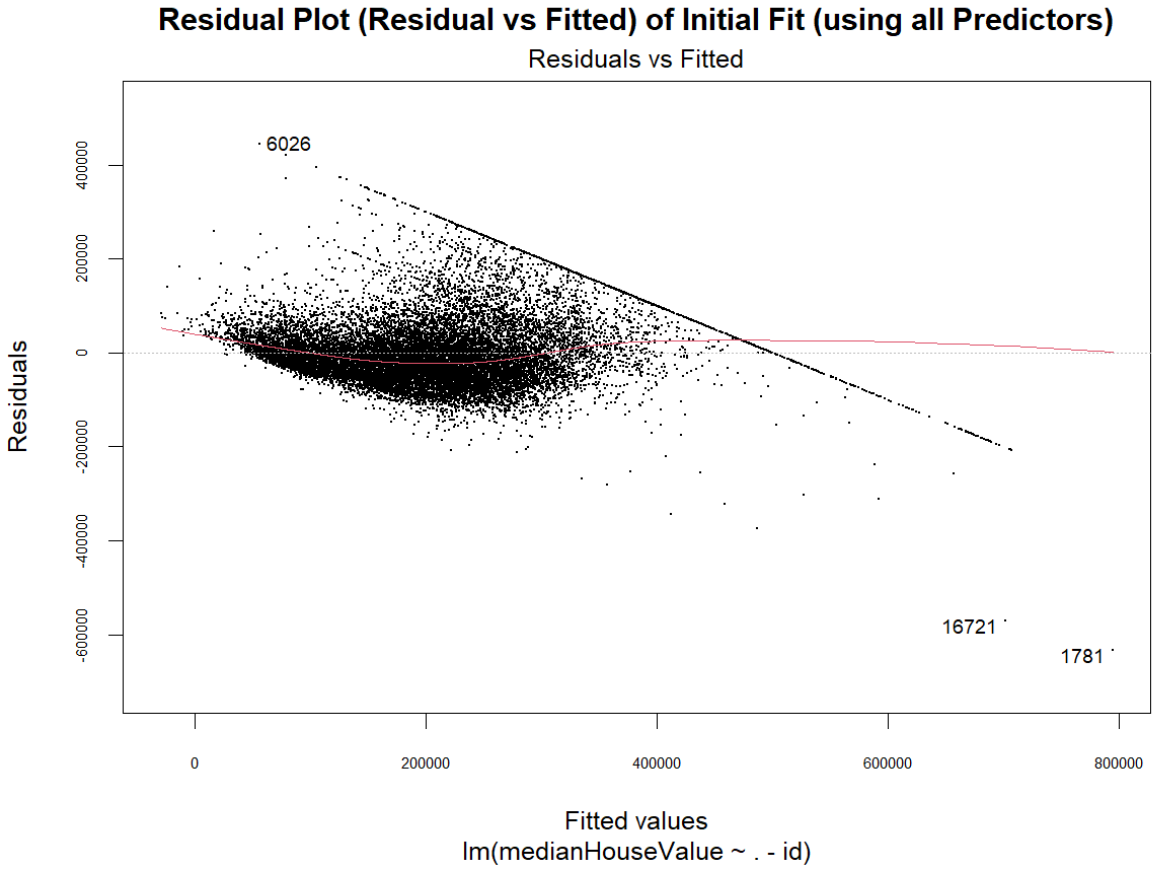
2.3.1 Residual Plot:

The residual plot shows that the points are not quite randomly scattered around zero. This means that the homoscedasiticity and linear assumption might be questionable. To address non-linearity, we can consider polynomial terms or interactions.

The funnel shape (the spread of residuals seems to increase as the fitted values increase) within the fitted value range (0, 400000) strengthens my doubt of heteroscedasticity. We may consider transforming some of our predictors later.

There are some outliers in the residuals that are far away from zero. These influential points may be high-leverage or outliers or both - should be investigated later.

```
par(mfrow = c(1, 1))
options(scipen = 999)
plot(initial_fit, which = 1, cex = 0.2, pch = 16, cex.axis = 0.6,
     main = "Residual Plot (Residual vs Fitted) of Initial Fit (using all Predictors)")
```



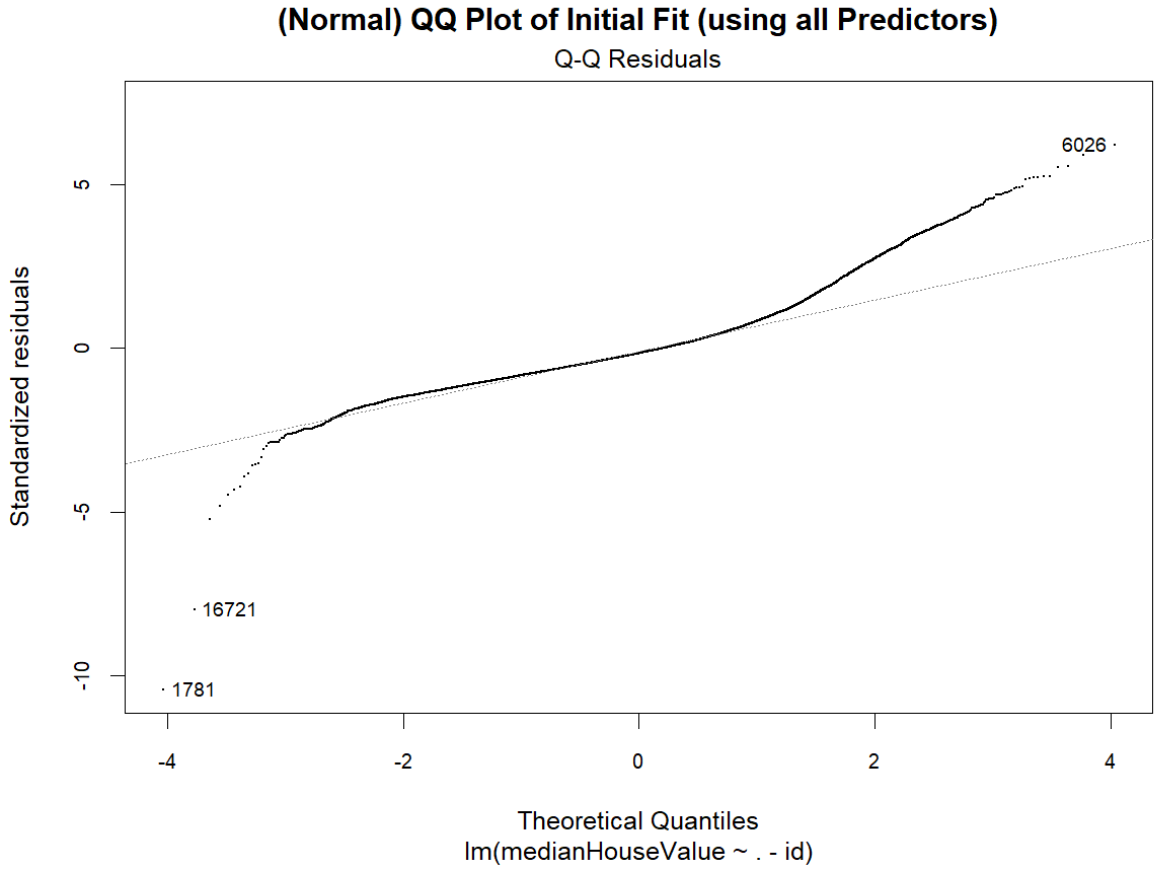
2.3.2 QQ Plot for Residuals:

A key assumption behind generalised linear model is that the error term is normally distributed. This is why t-statistics and F-statistics can be used in our previous testing.

T-statistics are robust under some mild deviation from normality, but under extreme non-normality, these statistics become less reliable.

In our plot, the points deviates from the reference line (dashed line) for larger and smaller quantiles (roughly outside of this range: (-2, 1.5)), indicating non-normality (especially high skewness) and influence of outliers especially at the tails.

```
plot(initial_fit, which = 2, cex = 0.2, pch = 16, cex.axis = 0.8,
     main = "(Normal) QQ Plot of Initial Fit (using all Predictors)")
```



2.3.3 Outliers

Outlier identification: as a rule of thumb, we consider those whose studentised residual has a magnitude greater than 3 as outliers here.

However we cannot just simply remove the outliers in this case. This is because these outliers could be attributable to model specification or other problems. This is partially addressed in 2.4.1 by bedroomsPerRoom. Note: outliers are not a big consideration in this data set because the response variable is both left and right censored, this is also reflected in the final model.

```
residuals_initial_fit <- residuals(initial_fit)
stdresiduals_initial_fit <- rstandard(initial_fit)
outlier_row_number <- which(abs(stdresiduals_initial_fit) > 3)

length(outlier_row_number) # gives how many outliers are there
```

```
## [1] 314
```

2.3.4 High Leverage Points

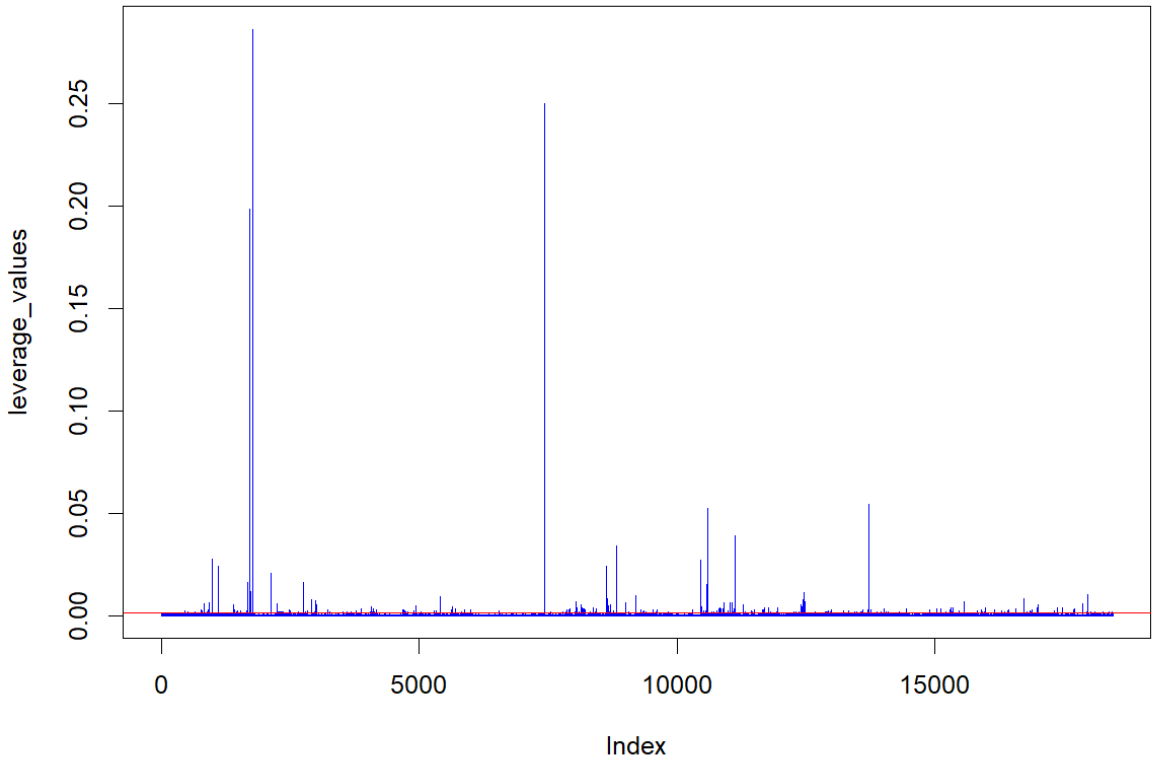
We will compute the leverage statistic hi and to see whether it is » (p + 1)/n. Where p is the number of predictors in the model and n is the sample size.

Having 3,317 high leverage points out of 18,450 data points means that about 18% of the data has high leverage. This indicates that our regression line can change dramatically with small changes in the predictors. One possible reason is that we are overfitting the data - and a reason to this is having too many predictor variables.

```
leverage_values <- hatvalues(initial_fit)

p_lvg <- length(coef(initial_fit))
n_lvg <- nrow(data)
threshold_lvg <- (p_lvg + 1) / n_lvg

highlvg_row_number <- which(leverage_values > threshold_lvg)
length(highlvg_row_number)
```

```
## [1] 3317
```

```
plot(leverage_values, type = "h", col = "blue")
abline(h = 2 * mean(leverage_values), col = "red")
```



2.3.5 Collinearity

The arbitrary threshold of severe collinearity is VIF greater or equal to 5. Here, all the predictors are shown to have a non-severe VIF. However, Longitude and Latitude shows relatively high VIF comparing to other

predictors - a cause of this is the high correlation between the two.

```
vif(initial_fit)
```

```
##                       GVIF Df GVIF^(1/(2*Df))
## longitude        18.464342  1        4.297015
## latitude         20.529183  1        4.530914
## housingMedianAge  1.295871  1        1.138363
## aveRooms          8.994891  1        2.999148
## aveBedrooms       7.637752  1        2.763648
## population        1.134318  1        1.065044
## medianIncome      2.610282  1        1.615637
## oceanProximity    4.089926  4        1.192517
```

2.4 Model Improvements

2.4.1 New predictors added

```
# New fit
data$bedroomsPerRoom <- data$aveBedrooms / data$aveRooms
data$incomePerRoom = data$medianIncome / data$aveRooms

# Dist from LA and SF
library(geosphere)
# Need to reference this
la_coords <- c(-118.24, 34.05)   # (longitude, latitude)
sf_coords <- c(-122.42, 37.77)

# Add distance to LA
data$distToLA <- apply(data[, c("longitude", "latitude")], 1, function(coord) {
  distGeo(coord, la_coords) / 1000  # convert meters to km
})

#Add distance to SF
data$distToSF <- apply(data[, c("longitude", "latitude")], 1, function(coord) {
  distGeo(coord, sf_coords) / 1000
})


# Compute direction angles
data$dirToLA <- atan2(data$latitude - la_coords[2], data$longitude - la_coords[1])
data$dirToSF <- atan2(data$latitude - sf_coords[2], data$longitude - sf_coords[1])

# Encode directions using dsin and cos
data$cosDirToLA <- cos(data$dirToLA)
data$sinDirToLA <- sin(data$dirToLA)

data$cosDirToSF <- cos(data$dirToSF)
data$sinDirToSF <- sin(data$dirToSF)

# Remove intermediate angle variables
```

```r
data$dirToLA <- NULL
data$dirToSF <- NULL

data$cityProximityScore <- 1 / (1 + data$distToLA) + 1 / (1 + data$distToSF)

#distance to centre
center_lat <- mean(data$latitude)
center_lon <- mean(data$longitude)
data$distToCenter <- sqrt((data$latitude - center_lat)^2 + (data$longitude - center_lon)^2)
```

A number of new predictors have been added that are transformations of previous predictors, going through sequentially these are the justifications for each addition.

- bedroomsPerRoom: this additional predictor allows for the information in aveRooms and aveBedrooms to be included without running into the issue of collinearity in the model since as seen above aveRooms and aveBedrooms are correlated. Additionally it mitigates the issue of high leverage points which were very apparent in both aveRooms and aveBedrooms by standardising. Given the response variable is censored there is no value add for these high leverage points. Finally it is an affluence metric for the house in question. Low values for bedrooms per room meaning that there are far more rooms than bedrooms indicating that the house is less cramped and there's more space, it would make sense that these houses are more valuable.
- incomePerRoom: this feature was included because it captures how much income supports each unit of housing space, how much money is "backing" each room, a relationship that neither raw income nor room count alone can fully express, and one that likely correlates more strongly with housing prices.
- Distances and directions from Los Angeles and San Fransisco. As seen in the Geospatial plot, California has two major cities and those cities have the highest housing prices. In particular they are good proxies for whether medianHouseValue has been censored, because the vast majority of expensive homes, those over $500,001 are in those cities. These engineered features introduce geospatial context by quantifying both the Euclidean distance and relative bearing from each observation to Los Angeles and San Francisco, two primary centers of economic activity in California. Distance to these cities is a strong predictor of housing prices: since proximity to urban centers often means better access to jobs, amenities, and higher demand. Directional features add geographic nuance by distinguishing not just how far a home is, but where it lies in relation to the city, which can reflect differences in development, terrain, or desirability. Encoding direction using sine and cosine avoids issues with angle discontinuity and makes it easier the model to learn spatial patterns.
- cityProximityScore: this feature combines inverse-distance relationships to Los Angeles and San Francisco, assigning higher scores to homes that are closer to either city. It creates a smooth, nonlinear decay of influence with distance, ensuring that proximity to urban centers has a diminishing but continuous effect. This score effectively quantifies urban accessibility, helping the model prioritise locations that are geographically well-positioned relative to high-value economic hubs.
- distToCenter: this feature introduces a global spatial reference by computing the Euclidean distance from each observation to the dataset's centroid, defined by the mean latitude and longitude. This addresses potential spatial drift or boundary bias, where model performance may degrade at the geographic edges of the dataset due to uneven sampling or regional sparsity. By capturing centrality relative to the full spatial domain, not just to high-demand cities, it helps the model learn broad spatial gradients and corrects for systematic variation in housing value that might arise from being on the dataset's periphery rather than near economic centers.

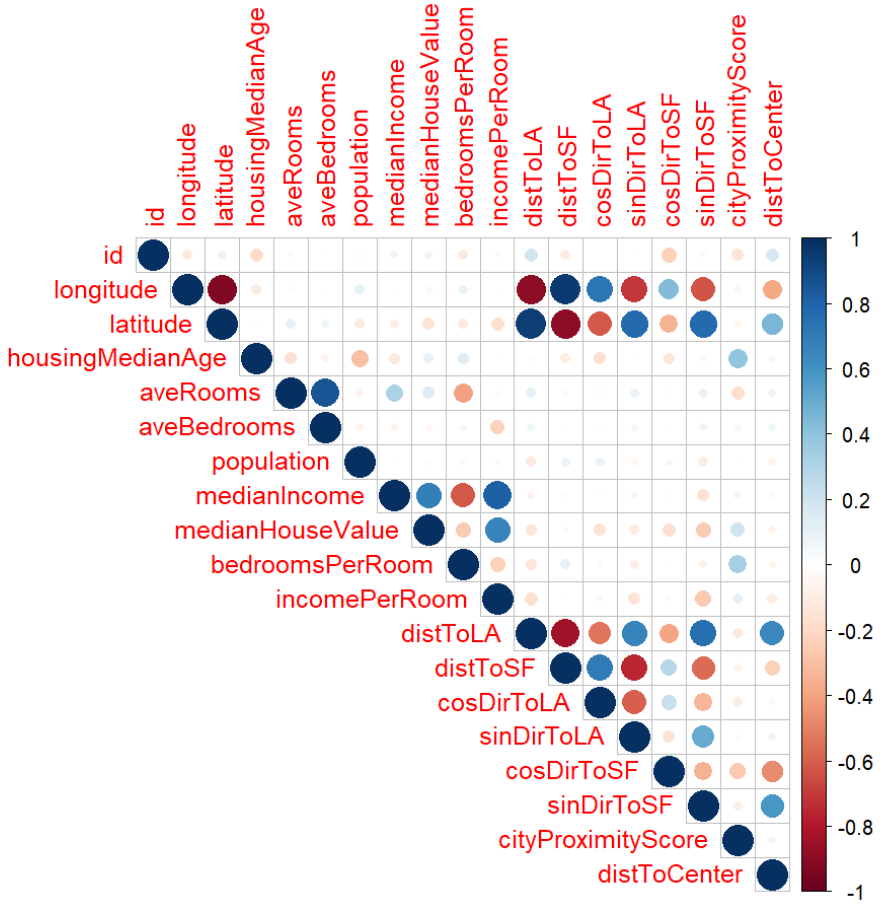
2.4.2 Choosing interaction terms and non-linear transformations

```r
# Select all numeric columns except 'ID'
numeric.data <- data[sapply(data, is.numeric)]
numeric.data$ID <- NULL

# Correlation between medianHouseValue and all other numeric variables
cor_medianHouseValue <- cor(data$medianHouseValue, numeric.data)
print(cor_medianHouseValue)
```

2.4.2.1 Choosing interaction terms from covariance matrix

```
##              id   longitude   latitude housingMedianAge aveRooms aveBedrooms
## [1,] 0.07198962 -0.04646254 -0.1439446        0.1052129 0.148625 -0.04545692
##       population medianIncome medianHouseValue bedroomsPerRoom incomePerRoom
## [1,] -0.02394206     0.689536                1      -0.2554662      0.665614
##        distToLA    distToSF cosDirToLA sinDirToLA cosDirToSF sinDirToSF
## [1,] -0.1309826 -0.03126119 -0.1547648 -0.1069453 -0.1746119  -0.251576
##      cityProximityScore distToCenter
## [1,]          0.2062844  -0.07148864
```

```r
# Correlation plot for all numeric variables
library(corrplot)
corrplot(cor(numeric.data), method = "circle",type =
          "upper")
```



Interaction choices for the model

Note: Some relate to covariance matrix others do not

- longitude:latitude
  - The variables longitude and latitude are not strongly correlated with medianHouseValue on their own but show strong negative correlation with each other, but together they define a unique spatial location. Their interaction enables the model to capture regional effects that are not purely east-west or north-south, but a combination of both.
  - The other reason why to include an interaction term is to reduce multi-collinearity between these two variables. This interaction represents the geographic positioning of each home, allowing the model to pick up spatial clusters in property value.
- bedroomsPerRoom:medianIncome
  - bedroomsPerRoom is negatively correlated with medianIncome. Pairing them allows the model to test whether the quality or crowding of space varies in importance across income levels. This interaction captures how the value of spaciousness or room density changes depending on neighbourhood affluence.
- medianIncome:housingMedianAge
  - medianIncome is strongly positively correlated with medianHouseValue, while housingMedianAge shows a weaker positive or near-zero correlation. The two variables are not strongly correlated

with each other, making them suitable for interaction. This interaction captures whether wealth is concentrated in newer or older neighbourhoods, revealing how the relationship between income and housing value shifts with the age of the housing stock.

- distToCenter:medianIncome
  - This interaction was chosen because it captures how the influence of income on housing value may vary with geographic centrality. High income in a central, urban location may signal access to premium markets, whereas high income in a peripheral location might reflect different lifestyle choices (e.g. rural affluence or luxury sprawl). This term allows the model to distinguish between urban wealth and suburban or rural wealth, identifying how location context changes the effect of income on house prices.
- latitude:incomePerRoom
  - latitude and incomePerRoom are not highly correlated with each other. Including the interaction helps model spatial economic patterns that are not captured by either variable alone. This interaction reflects how the value of economic density varies along the north–south axis of California.
- bedroomsPerRoom:distToLA
  - bedroomsPerRoom and distToLA are weakly correlated with each other and with medianHouseValue. Their interaction enables the model to explore whether the meaning of room layout or density shifts based on urban proximity. This expresses how space efficiency is valued differently depending on distance from Los Angeles.
- cityProximityScore:housingMedianAge
  - cityProximityScore is positively correlated with medianHouseValue, and weakly so with housingMedianAge. Their interaction tests whether proximity to major cities boosts or dampens the value of older housing stock. This models whether older homes near urban centres are more desirable or more heavily discounted.
- cityProximityScore:bedroomsPerRoom
  - These variables are weakly correlated with each other and allow the model to investigate if spatial efficiency (bedroom density) is more or less valuable near high-demand areas. This reflects how crowding or layout impacts price differently in urban versus more distant areas.
- housingMedianAge:incomePerRoom
  - These two have low correlation but both relate to housing quality and socioeconomic context. Their interaction allows for the effect of economic density to vary depending on housing stock age. This models how the combination of compact affluence and housing maturity affects property value.
- The four interaction terms between medianIncome and directional components (cosDirToLA, sinDirToLA, cosDirToSF, sinDirToSF) were included to model how the effect of income on housing value changes depending on where a home is situated relative to Los Angeles and San Francisco. Income is the strongest individual predictor of housing value, but its impact is not spatially uniform. These directional interactions allow the model to account for how the purchasing power and market influence of income varies depending on urban proximity, regional economies, and land use patterns. In essence, they help capture whether income drives prices more strongly in certain directions, reflecting localised economic geographies and how spatial context modifies the effect of wealth.

2.4.2.2 Choosing non-linear terms

- log(medianIncome / distToCenter)
  - Dividing income by distance to center captures the economic value of location, higher income closer to the center is associated with more valuable housing. This term measures the "effective income" adjusted for geographic desirability. Log scaling reduces the impact of extreme values and helps linearise the relationship in the context of multi-linear regression. It ensures the model captures diminishing returns, the impact of a change in income or distance is smaller at higher levels.
- I(medianIncome / housingMedianAge)
  - Combining income with housing age highlights areas where economic resources are mismatched with infrastructure age. High income in areas with older housing can indicate redevelopment potential, while low income and old housing may correlate with lower prices.
- log(incomePerRoom) where incomePerRoom = medianIncome / aveRooms
  - Dividing income by average rooms captures income per unit of housing capacity. This reflects how wealth is distributed relative to housing size and can act as a proxy for crowding or luxury. Logging controls for skewed distributions and allows the model to interpret multiplicative effects additively. It helps the model distinguish between different housing market conditions more effectively, especially at the extremes of income or room size.
- All three terms are related to medianIncome because: medianIncome is the strongest individual predictor of house prices in the California Housing dataset.It has the highest correlation with medianHousevalue. Housing prices are highly influenced by local purchasing power, and income reflects the ability of residents to pay for housing. Modifying and combining medianIncome with other variables can reveal more complex and informative relationships that are not captured by the raw variable alone.
- See **1.2.6** for the justification of other non-linear terms used in the final model (i.e. log and polynomial transformations).

2.4.3 Best subset selection with new predictors and interaction terms

New function - Must run to run next chunk. Creates method for regsubsets in predict().

```
predict.regsubsets=function(object,newdata,id,...){
  #... allows for other arguments to be passed into the function
  form=as.formula(object$call[[2]])
  mat=model.matrix(form,newdata)
  coefi=coef(object,id=id)
  xvars=names(coefi)
  mat[,xvars]%*%coefi
}
```

Checking whether all of the chosen predictors are worth adding. Do they result in the **best** model?

```
library(car)
# k-fold CV

formula_string <- "  medianHouseValue ~
```

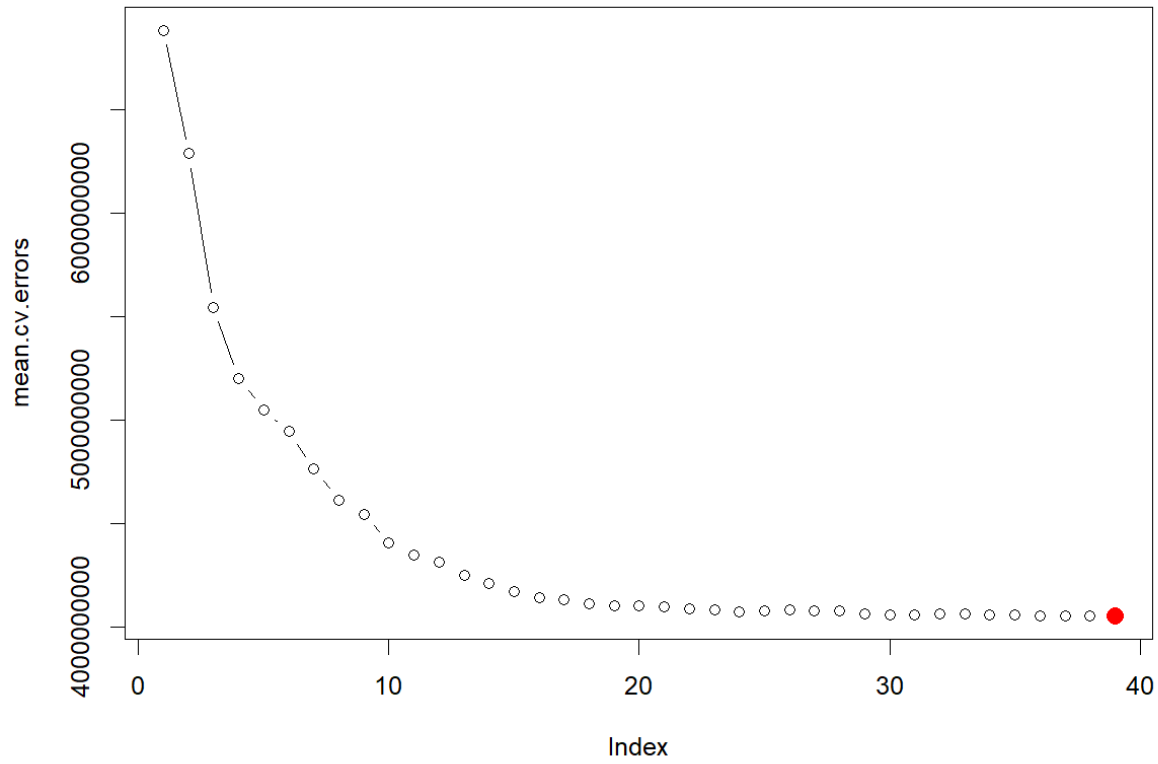
```
    . - id - oceanProximity - aveBedrooms - aveRooms - population +


    longitude:latitude +
    bedroomsPerRoom:medianIncome +
    medianIncome:housingMedianAge +
    distToCenter:medianIncome +
    latitude:incomePerRoom +
    bedroomsPerRoom:distToLA +
    medianIncome:log(population) +
    medianIncome:cosDirToLA +
    medianIncome:sinDirToLA +
    medianIncome:cosDirToSF +
    medianIncome:sinDirToSF +
    cityProximityScore:housingMedianAge +
    cityProximityScore:bedroomsPerRoom +
    housingMedianAge:incomePerRoom +



    I(medianIncome^2) +
    I(medianIncome^3) +
    I(housingMedianAge^2) +
    I(medianIncome / housingMedianAge) +
    I(latitude^2) +
    I(longitude^2) +
    I(latitude^3) +
    I(longitude^3) +


    log(population) +
    log(incomePerRoom) +
    log(medianIncome / distToCenter)"


lm.fit <- lm(formula = as.formula(formula_string),data = data)
summary(lm.fit)
```

```
##
## Call:
## lm(formula = as.formula(formula_string), data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -408775  -38162   -8192   26297  444997
##
## Coefficients:
##                                       Estimate     Std. Error t value
## (Intercept)                     -365958026.525  190786730.783  -1.918
## longitude                         -9304695.009    4688340.863  -1.985
## latitude                           1171910.406     595662.514   1.967
## housingMedianAge                     -2328.347        246.687  -9.438
## medianIncome                        -14082.986       6798.503  -2.071
```

```
## bedroomsPerRoom                          137551.352      32167.239   4.276
## incomePerRoom                           -343090.014      62679.259  -5.474
## distToLA                                    -310.932         33.858  -9.184
## distToSF                                    -467.020         35.233 -13.255
## cosDirToLA                                -19265.312       2402.240  -8.020
## sinDirToLA                                 53824.936       2908.701  18.505
## cosDirToSF                                -64289.055       4695.444 -13.692
## sinDirToSF                                -11800.223       4073.349  -2.897
## cityProximityScore                        -70396.767      55366.333  -1.271
## distToCenter                               12262.237       8609.351   1.424
## I(medianIncome^2)                           7139.209        634.332  11.255
## I(medianIncome^3)                           -416.072         24.983 -16.654
## I(housingMedianAge^2)                         14.487          3.646   3.973
## I(medianIncome/housingMedianAge)           -5212.574       3161.921  -1.649
## I(latitude^2)                               9340.986      17325.572   0.539
## I(longitude^2)                            -79974.600      39010.420  -2.050
## I(latitude^3)                                70.823        148.808   0.476
## I(longitude^3)                             -244.235        108.532  -2.250
## log(population)                           -13148.920       1372.858  -9.578
## log(incomePerRoom)                         20511.023       5166.048   3.970
## log(medianIncome/distToCenter)            -39790.608       7790.252  -5.108
## longitude:latitude                         18099.176       1352.990  13.377
## medianIncome:bedroomsPerRoom              115893.385       9909.695  11.695
## housingMedianAge:medianIncome                164.677         37.389   4.404
## medianIncome:distToCenter                   1194.777        485.961   2.459
## latitude:incomePerRoom                      4897.396       1765.195   2.774
## bedroomsPerRoom:distToLA                    -149.341         47.965  -3.114
## medianIncome:log(population)                2400.154        302.941   7.923
## medianIncome:cosDirToLA                    -2340.453        490.125  -4.775
## medianIncome:sinDirToLA                    -8633.563        657.784 -13.125
## medianIncome:cosDirToSF                     4353.736        932.040   4.671
## medianIncome:sinDirToSF                    -1604.394        775.812  -2.068
## housingMedianAge:cityProximityScore         3729.316        835.767   4.462
## bedroomsPerRoom:cityProximityScore       -169499.900     101452.926  -1.671
## housingMedianAge:incomePerRoom              1497.168        258.598   5.790
##                                                Pr(>|t|)
## (Intercept)                                     0.05511 .
## longitude                                       0.04720 *
## latitude                                        0.04915 *
## housingMedianAge                  < 0.0000000000000002 ***
## medianIncome                                    0.03833 *
## bedroomsPerRoom                    0.00001911266371441 ***
## incomePerRoom                      0.00000004463901963 ***
## distToLA                          < 0.0000000000000002 ***
## distToSF                          < 0.0000000000000002 ***
## cosDirToLA                         0.0000000000000112 ***
## sinDirToLA                        < 0.0000000000000002 ***
## cosDirToSF                        < 0.0000000000000002 ***
## sinDirToSF                                      0.00377 **
## cityProximityScore                              0.20358
## distToCenter                                    0.15438
## I(medianIncome^2)                 < 0.0000000000000002 ***
## I(medianIncome^3)                 < 0.0000000000000002 ***
## I(housingMedianAge^2)              0.00007124761918128 ***
```

```
## I(medianIncome/housingMedianAge)                0.09926 .
## I(latitude^2)                                   0.58979
## I(longitude^2)                                  0.04037 *
## I(latitude^3)                                   0.63413
## I(longitude^3)                                  0.02444 *
## log(population)                    < 0.0000000000000002 ***
## log(incomePerRoom)                  0.00007203953579710 ***
## log(medianIncome/distToCenter)      0.00000032928291737 ***
## longitude:latitude                 < 0.0000000000000002 ***
## medianIncome:bedroomsPerRoom       < 0.0000000000000002 ***
## housingMedianAge:medianIncome       0.00001066545658764 ***
## medianIncome:distToCenter                       0.01396 *
## latitude:incomePerRoom                          0.00554 **
## bedroomsPerRoom:distToLA                        0.00185 **
## medianIncome:log(population)        0.00000000000000245 ***
## medianIncome:cosDirToLA             0.00000180896994856 ***
## medianIncome:sinDirToLA            < 0.0000000000000002 ***
## medianIncome:cosDirToSF             0.00000301578362543 ***
## medianIncome:sinDirToSF                         0.03865 *
## housingMedianAge:cityProximityScore 0.00000816237931750 ***
## bedroomsPerRoom:cityProximityScore              0.09479 .
## housingMedianAge:incomePerRoom      0.00000000717233811 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63430 on 18410 degrees of freedom
## Multiple R-squared:  0.7001, Adjusted R-squared:  0.6995
## F-statistic:  1102 on 39 and 18410 DF,  p-value: < 0.00000000000000022
```

```r
nvmax <- length(coef(lm.fit)) - 1


k=10
set.seed(3)
folds=sample(1:k,nrow(data),replace=TRUE)
cv.errors=matrix(NA,k,nvmax, dimnames=list(NULL, paste(1:nvmax))) # NA means no data, NULL means no row


for(j in 1:k){
  best.fit=regsubsets(x = as.formula(formula_string),data = data[folds!=j,],nvmax=nvmax)
  for(i in 1:nvmax){
    pred=predict(best.fit,data[folds==j,],id=i)
    cv.errors[j,i]=mean( (data$medianHouseValue[folds==j]-pred)^2)
  }
}
mean.cv.errors=apply(cv.errors,2,mean)
par(mfrow=c(1,1))
best.model.size <- as.numeric(names(which.min(mean.cv.errors)))
plot(mean.cv.errors,type='b')
points(best.model.size, mean.cv.errors[best.model.size], col = "red", pch = 19, cex = 1.5)
```




```r
# Obtain the best subset model using the full data and CV selected id
reg.best=regsubsets(x = as.formula(formula_string),data = data, nvmax=nvmax)

best.predictors = names(coef(reg.best, best.model.size))[-1]  # remove intercept
# Create formula dynamically
formula.best = as.formula(paste("medianHouseValue ~", paste(best.predictors, collapse = " + ")))

# Fit the model using lm
model.best = lm(formula.best, data = data)
summary(model.best)
```

```
##
## Call:
## lm(formula = formula.best, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -408775  -38162   -8192   26297  444997
##
## Coefficients:
##                                    Estimate     Std. Error t value
## (Intercept)                    -365958026.525  190786730.783  -1.918
## longitude                        -9304695.009    4688340.863  -1.985
```

```
## latitude                                1171910.406     595662.514   1.967
## housingMedianAge                          -2328.347        246.687  -9.438
## medianIncome                             -14082.986       6798.503  -2.071
## bedroomsPerRoom                          137551.352      32167.239   4.276
## incomePerRoom                           -343090.014      62679.259  -5.474
## distToLA                                    -310.932         33.858  -9.184
## distToSF                                    -467.020         35.233 -13.255
## cosDirToLA                               -19265.312       2402.240  -8.020
## sinDirToLA                                53824.936       2908.701  18.505
## cosDirToSF                               -64289.055       4695.444 -13.692
## sinDirToSF                               -11800.223       4073.349  -2.897
## cityProximityScore                       -70396.767      55366.333  -1.271
## distToCenter                              12262.237       8609.351   1.424
## I(medianIncome^2)                          7139.209        634.332  11.255
## I(medianIncome^3)                          -416.072         24.983 -16.654
## I(housingMedianAge^2)                        14.487          3.646   3.973
## I(medianIncome/housingMedianAge)           -5212.574       3161.921  -1.649
## I(latitude^2)                              9340.986      17325.572   0.539
## I(longitude^2)                           -79974.600      39010.420  -2.050
## I(latitude^3)                               70.823        148.808   0.476
## I(longitude^3)                             -244.235        108.532  -2.250
## log(population)                          -13148.920       1372.858  -9.578
## log(incomePerRoom)                        20511.023       5166.048   3.970
## log(medianIncome/distToCenter)           -39790.608       7790.252  -5.108
## longitude:latitude                        18099.176       1352.990  13.377
## medianIncome:bedroomsPerRoom             115893.385       9909.695  11.695
## housingMedianAge:medianIncome               164.677         37.389   4.404
## medianIncome:distToCenter                  1194.777        485.961   2.459
## latitude:incomePerRoom                     4897.396       1765.195   2.774
## bedroomsPerRoom:distToLA                   -149.341         47.965  -3.114
## medianIncome:log(population)               2400.154        302.941   7.923
## medianIncome:cosDirToLA                   -2340.453        490.125  -4.775
## medianIncome:sinDirToLA                   -8633.563        657.784 -13.125
## medianIncome:cosDirToSF                    4353.736        932.040   4.671
## medianIncome:sinDirToSF                   -1604.394        775.812  -2.068
## housingMedianAge:cityProximityScore        3729.316        835.767   4.462
## bedroomsPerRoom:cityProximityScore      -169499.900     101452.926  -1.671
## housingMedianAge:incomePerRoom             1497.168        258.598   5.790
##                                                Pr(>|t|)
## (Intercept)                                     0.05511 .
## longitude                                       0.04720 *
## latitude                                        0.04915 *
## housingMedianAge                  < 0.0000000000000002 ***
## medianIncome                                    0.03833 *
## bedroomsPerRoom                   0.00001911266371441 ***
## incomePerRoom                     0.00000004463901963 ***
## distToLA                          < 0.0000000000000002 ***
## distToSF                          < 0.0000000000000002 ***
## cosDirToLA                        0.0000000000000112 ***
## sinDirToLA                        < 0.0000000000000002 ***
## cosDirToSF                        < 0.0000000000000002 ***
## sinDirToSF                                      0.00377 **
## cityProximityScore                              0.20358
## distToCenter                                    0.15438
```

```
## I(medianIncome^2)                   < 0.0000000000000002 ***
## I(medianIncome^3)                   < 0.0000000000000002 ***
## I(housingMedianAge^2)                0.00007124761918128 ***
## I(medianIncome/housingMedianAge)             0.09926 .
## I(latitude^2)                                0.58979
## I(longitude^2)                               0.04037 *
## I(latitude^3)                                0.63413
## I(longitude^3)                               0.02444 *
## log(population)                     < 0.0000000000000002 ***
## log(incomePerRoom)                   0.00007203953579710 ***
## log(medianIncome/distToCenter)       0.00000032928291737 ***
## longitude:latitude                  < 0.0000000000000002 ***
## medianIncome:bedroomsPerRoom        < 0.0000000000000002 ***
## housingMedianAge:medianIncome        0.00001066545658764 ***
## medianIncome:distToCenter                    0.01396 *
## latitude:incomePerRoom                       0.00554 **
## bedroomsPerRoom:distToLA                     0.00185 **
## medianIncome:log(population)         0.00000000000000245 ***
## medianIncome:cosDirToLA              0.00000180896994856 ***
## medianIncome:sinDirToLA             < 0.0000000000000002 ***
## medianIncome:cosDirToSF              0.00000301578362543 ***
## medianIncome:sinDirToSF                      0.03865 *
## housingMedianAge:cityProximityScore  0.00000816237931750 ***
## bedroomsPerRoom:cityProximityScore           0.09479 .
## housingMedianAge:incomePerRoom       0.00000000717233811 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63430 on 18410 degrees of freedom
## Multiple R-squared:  0.7001, Adjusted R-squared:  0.6995
## F-statistic:  1102 on 39 and 18410 DF,  p-value: < 0.00000000000000022
```

From best subset selection, all predictors provided above improve the model.

2.4.4 Adding in oceanProximity to model

Since oceanProximity is a factor variable, best selection cannot be performed on it. Either all factors relating to oceanProximity are included or none are. Given the p-value's for the factors relating to oceanProximity in the initial model (2.1) are significant, in particular "Inland", oceanProximity will be included in the model.

In additional to oceanProximity these two interaction terms have been added:

- oceanProximity:distToCenter
  - This term captures how the effect of distance to the geographic center varies by coastal category. For example, INLAND areas may become less valuable with distance from the center, while NEAR BAY areas may be valuable regardless of centrality.
  - This term allows the slope of distToCenter to change across oceanProximity groups. Instead of assuming a single, fixed effect of distance for all areas, the model can learn group-specific sensitivities to distance.
- oceanProximity:medianIncome

- Income may affect house prices differently across coastal regions. High income near the ocean could indicate luxury real estate, while the same income level inland may not produce the same price signal. This term allows the effect of income on price to vary by oceanProximity.
- The model estimates different coefficients for income in each region, capturing regional differences in how income translates into housing value.

```
# With oceanProximity
new.formula = update(formula.best, . ~ . + oceanProximity +
                       oceanProximity:distToCenter +oceanProximity:medianIncome)
model.best = lm(new.formula, data = data)

summary(model.best)
```

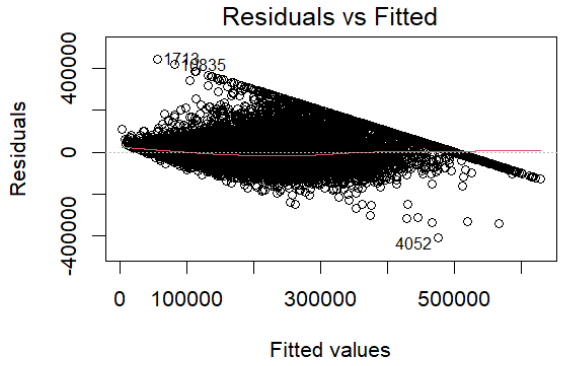
```
##
## Call:
## lm(formula = new.formula, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -408514  -37711   -8488   26111  444487
##
## Coefficients:
##                                          Estimate     Std. Error t value
## (Intercept)                          1221195367.532  308730841.015   3.956
## longitude                              31625764.372    7801341.024   4.054
## latitude                                2575546.922     674208.586   3.820
## housingMedianAge                          -2312.125        246.099  -9.395
## medianIncome                              -7452.574       7058.028  -1.056
## bedroomsPerRoom                          144592.331      32309.499   4.475
## incomePerRoom                           -343692.666      63133.380  -5.444
## distToLA                                    -361.874         36.713  -9.857
## distToSF                                    -324.034         45.098  -7.185
## cosDirToLA                                -23920.844       2528.925  -9.459
## sinDirToLA                                 50782.338       3048.435  16.658
## cosDirToSF                                -68159.116       4853.541 -14.043
## sinDirToSF                                -11680.409       4617.229  -2.530
## cityProximityScore                        -44241.047      55916.370  -0.791
## distToCenter                               42674.364       9635.788   4.429
## I(medianIncome^2)                           6598.428        647.080  10.197
## I(medianIncome^3)                           -394.358         25.434 -15.505
## I(housingMedianAge^2)                         14.117          3.641   3.877
## I(medianIncome/housingMedianAge)           -5921.504       3153.363  -1.878
## I(latitude^2)                             -38946.219      19661.659  -1.981
## I(longitude^2)                            267182.061      65655.990   4.069
## I(latitude^3)                                444.931        168.191   2.645
## I(longitude^3)                               741.101        184.529   4.016
## log(population)                           -13019.223       1369.370  -9.507
## log(incomePerRoom)                         19042.460       5171.140   3.682
## log(medianIncome/distToCenter)            -36045.347       7886.918  -4.570
## oceanProximityINLAND                       41610.139       8483.764   4.905
## oceanProximityISLAND                    -2416681.302    1320271.246  -1.830
## oceanProximityNEAR BAY                     -6038.556      35011.840  -0.172
## oceanProximityNEAR OCEAN                   18980.973       6702.481   2.832
```

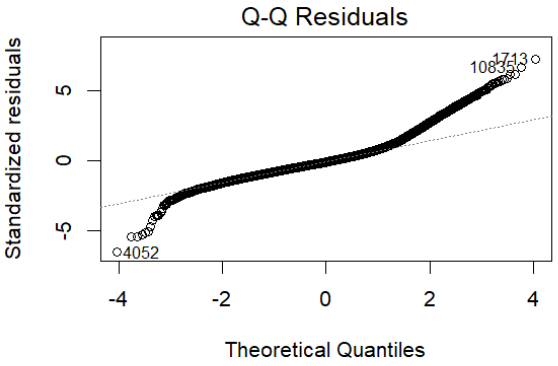
```
## longitude:latitude                          12863.459       1616.064   7.960
## medianIncome:bedroomsPerRoom               111176.839       9887.108  11.245
## housingMedianAge:medianIncome                 122.941         37.936   3.241
## medianIncome:distToCenter                     771.541        526.126   1.466
## latitude:incomePerRoom                       4836.782       1775.712   2.724
## bedroomsPerRoom:distToLA                     -120.748         48.247  -2.503
## medianIncome:log(population)                 2290.172        302.512   7.571
## medianIncome:cosDirToLA                     -1553.312        536.264  -2.897
## medianIncome:sinDirToLA                     -7518.266        705.836 -10.652
## medianIncome:cosDirToSF                      5639.558        963.433   5.854
## medianIncome:sinDirToSF                     -1121.863        891.543  -1.258
## housingMedianAge:cityProximityScore          3904.937        834.823   4.678
## bedroomsPerRoom:cityProximityScore        -198869.663     102624.752  -1.938
## housingMedianAge:incomePerRoom               1614.170        258.626   6.241
## distToCenter:oceanProximityINLAND          -12188.084       2369.785  -5.143
## distToCenter:oceanProximityISLAND         1327063.411     589026.767   2.253
## distToCenter:oceanProximityNEAR BAY          1615.530       9939.055   0.163
## distToCenter:oceanProximityNEAR OCEAN          60.914       1736.678   0.035
## medianIncome:oceanProximityINLAND           -4167.721       1007.566  -4.136
## medianIncome:oceanProximityISLAND         -300389.435     132274.194  -2.271
## medianIncome:oceanProximityNEAR BAY          1815.726       1078.701   1.683
## medianIncome:oceanProximityNEAR OCEAN        -143.401        845.553  -0.170
##                                                 Pr(>|t|)
## (Intercept)                          0.00007665005233137 ***
## longitude                            0.00005058050980433 ***
## latitude                                        0.000134 ***
## housingMedianAge                    < 0.0000000000000002 ***
## medianIncome                                    0.291028
## bedroomsPerRoom                      0.00000767882042247 ***
## incomePerRoom                        0.00000005278953201 ***
## distToLA                            < 0.0000000000000002 ***
## distToSF                             0.00000000000069740 ***
## cosDirToLA                          < 0.0000000000000002 ***
## sinDirToLA                          < 0.0000000000000002 ***
## cosDirToSF                          < 0.0000000000000002 ***
## sinDirToSF                                      0.011423 *
## cityProximityScore                              0.428837
## distToCenter                         0.00000953331448664 ***
## I(medianIncome^2)                   < 0.0000000000000002 ***
## I(medianIncome^3)                   < 0.0000000000000002 ***
## I(housingMedianAge^2)                           0.000106 ***
## I(medianIncome/housingMedianAge)                0.060419 .
## I(latitude^2)                                   0.047626 *
## I(longitude^2)                       0.00004732621765700 ***
## I(latitude^3)                                   0.008167 **
## I(longitude^3)                       0.00005938170691358 ***
## log(population)                     < 0.0000000000000002 ***
## log(incomePerRoom)                              0.000232 ***
## log(medianIncome/distToCenter)       0.00000490263236826 ***
## oceanProximityINLAND                 0.00000094379180381 ***
## oceanProximityISLAND                            0.067200 .
## oceanProximityNEAR BAY                          0.863069
## oceanProximityNEAR OCEAN                        0.004632 **
## longitude:latitude                   0.00000000000000182 ***
```

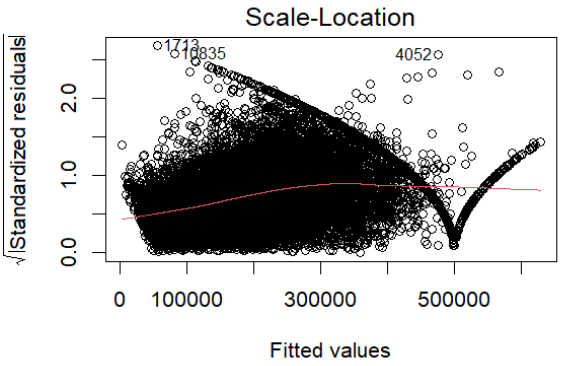
```
## medianIncome:bedroomsPerRoom          < 0.0000000000000002 ***
## housingMedianAge:medianIncome                      0.001194 **
## medianIncome:distToCenter                          0.142541
## latitude:incomePerRoom                             0.006459 **
## bedroomsPerRoom:distToLA                           0.012333 *
## medianIncome:log(population)          0.00000000000003893 ***
## medianIncome:cosDirToLA                            0.003777 **
## medianIncome:sinDirToLA               < 0.0000000000000002 ***
## medianIncome:cosDirToSF                0.00000000489202233 ***
## medianIncome:sinDirToSF                            0.208285
## housingMedianAge:cityProximityScore    0.00000292365690275 ***
## bedroomsPerRoom:cityProximityScore                 0.052659 .
## housingMedianAge:incomePerRoom         0.00000000044335998 ***
## distToCenter:oceanProximityINLAND      0.00000027298631852 ***
## distToCenter:oceanProximityISLAND                  0.024272 *
## distToCenter:oceanProximityNEAR BAY                0.870880
## distToCenter:oceanProximityNEAR OCEAN              0.972020
## medianIncome:oceanProximityINLAND      0.00003543258237127 ***
## medianIncome:oceanProximityISLAND                  0.023161 *
## medianIncome:oceanProximityNEAR BAY                0.092343 .
## medianIncome:oceanProximityNEAR OCEAN              0.865332
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63160 on 18398 degrees of freedom
## Multiple R-squared:  0.7029, Adjusted R-squared:  0.7021
## F-statistic: 853.4 on 51 and 18398 DF,  p-value: < 0.00000000000000022
```

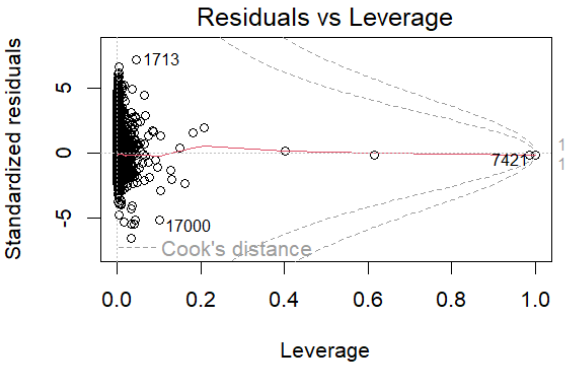
```
par(mfrow = c(2,2))
plot(model.best)
```

```
## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced
## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced
```










```
# High leverage points relate to ISLAND
data[data$oceanProximity == "ISLAND",]
```

```
##        id longitude latitude housingMedianAge aveRooms aveBedrooms population
## 7418 8316   -118.33    33.34                52 5.473318    1.371230       1100
## 7419 8317   -118.32    33.33                52 7.385417    1.777778        733
## 7420 8318   -118.32    33.34                52 6.225000    1.650000        341
## 7421 8319   -118.48    33.43                29 4.138728    1.236994        422
##      medianIncome medianHouseValue oceanProximity bedroomsPerRoom incomePerRoom
## 7418       2.8333           414700         ISLAND       0.2505299     0.5176568
## 7419       3.3906           300000         ISLAND       0.2407146     0.4590939
## 7420       2.7361           450000         ISLAND       0.2650602     0.4395341
## 7421       2.6042           287500         ISLAND       0.2988827     0.6292271
##      distToLA distToSF cosDirToLA sinDirToLA cosDirToSF sinDirToSF
## 7418 79.19181 615.5536 -0.1257543 -0.9920614  0.6783490 -0.7347399
## 7419 80.20382 616.9979 -0.1104315 -0.9938837  0.6784175 -0.7346766
## 7420 79.09950 616.0994 -0.1119675 -0.9937119  0.6792428 -0.7339136
## 7421 72.27561 599.3084 -0.3609941 -0.9325681  0.6721632 -0.7404031
##      cityProximityScore distToCenter
## 7418         0.01409202     2.605027
## 7419         0.01393282     2.618585
## 7420         0.01410496     2.609803
## 7421         0.01531292     2.456086
```

From the output it is clear that the model has improved since the reintroduction of oceanProximity and its respective interaction terms. The Adjusted R-squared has increased from 0.6995 to 0.7021.

From the Residuals vs Fitted plot it is clear that the fit has improved significantly since the original model and the Scale Location plot displays less heteroscedasticity than the previous model. However one cause for concern is the significant leverage (for two rows leverage is 1). However from further investigation it is clear that the the highly leveraged points relate to the four rows where oceanProximity is ISLAND. If they are removed ISLAND can't be used a predictor and hence neither than oceanProximity. This high leverage for a minority of houses is where oceanProximity is ISLAND is a trade-off of the final model. This should not be a big issue for the final test MSE because the predictions will be left and right censored.

2.5 Most significant predictors

```
# Using best selection
regfit.full=regsubsets(new.formula,data = data,nvmax = 3,really.big = TRUE)
reg.full.summary <- summary(regfit.full)

# Show which variables are included in the best model of size 3
which(reg.full.summary$which[3, ])[-1]  # '3' refers to 3-variable model and [-1] removes intercept
```

```
##                         medianIncome                oceanProximityINLAND
##                                    5                                  27
## housingMedianAge:cityProximityScore
##                                   42
```

```
summary(model.best)
```

```
##
## Call:
## lm(formula = new.formula, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -408514  -37711   -8488   26111  444487
##
## Coefficients:
##                                          Estimate     Std. Error t value
## (Intercept)                          1221195367.532  308730841.015   3.956
## longitude                              31625764.372    7801341.024   4.054
## latitude                                2575546.922     674208.586   3.820
## housingMedianAge                          -2312.125        246.099  -9.395
## medianIncome                              -7452.574       7058.028  -1.056
## bedroomsPerRoom                          144592.331      32309.499   4.475
## incomePerRoom                           -343692.666      63133.380  -5.444
## distToLA                                    -361.874         36.713  -9.857
## distToSF                                    -324.034         45.098  -7.185
## cosDirToLA                                -23920.844       2528.925  -9.459
## sinDirToLA                                 50782.338       3048.435  16.658
## cosDirToSF                                -68159.116       4853.541 -14.043
## sinDirToSF                                -11680.409       4617.229  -2.530
## cityProximityScore                        -44241.047      55916.370  -0.791
## distToCenter                               42674.364       9635.788   4.429
```

```
## I(medianIncome^2)                          6598.428        647.080  10.197
## I(medianIncome^3)                          -394.358         25.434 -15.505
## I(housingMedianAge^2)                        14.117          3.641   3.877
## I(medianIncome/housingMedianAge)          -5921.504       3153.363  -1.878
## I(latitude^2)                            -38946.219      19661.659  -1.981
## I(longitude^2)                           267182.061      65655.990   4.069
## I(latitude^3)                               444.931        168.191   2.645
## I(longitude^3)                              741.101        184.529   4.016
## log(population)                          -13019.223       1369.370  -9.507
## log(incomePerRoom)                        19042.460       5171.140   3.682
## log(medianIncome/distToCenter)           -36045.347       7886.918  -4.570
## oceanProximityINLAND                      41610.139       8483.764   4.905
## oceanProximityISLAND                  -2416681.302    1320271.246  -1.830
## oceanProximityNEAR BAY                   -6038.556      35011.840  -0.172
## oceanProximityNEAR OCEAN                 18980.973       6702.481   2.832
## longitude:latitude                       12863.459       1616.064   7.960
## medianIncome:bedroomsPerRoom            111176.839       9887.108  11.245
## housingMedianAge:medianIncome              122.941         37.936   3.241
## medianIncome:distToCenter                  771.541        526.126   1.466
## latitude:incomePerRoom                    4836.782       1775.712   2.724
## bedroomsPerRoom:distToLA                  -120.748         48.247  -2.503
## medianIncome:log(population)              2290.172        302.512   7.571
## medianIncome:cosDirToLA                  -1553.312        536.264  -2.897
## medianIncome:sinDirToLA                  -7518.266        705.836 -10.652
## medianIncome:cosDirToSF                   5639.558        963.433   5.854
## medianIncome:sinDirToSF                  -1121.863        891.543  -1.258
## housingMedianAge:cityProximityScore       3904.937        834.823   4.678
## bedroomsPerRoom:cityProximityScore     -198869.663     102624.752  -1.938
## housingMedianAge:incomePerRoom            1614.170        258.626   6.241
## distToCenter:oceanProximityINLAND       -12188.084       2369.785  -5.143
## distToCenter:oceanProximityISLAND       1327063.411     589026.767   2.253
## distToCenter:oceanProximityNEAR BAY        1615.530       9939.055   0.163
## distToCenter:oceanProximityNEAR OCEAN        60.914       1736.678   0.035
## medianIncome:oceanProximityINLAND         -4167.721       1007.566  -4.136
## medianIncome:oceanProximityISLAND       -300389.435     132274.194  -2.271
## medianIncome:oceanProximityNEAR BAY        1815.726       1078.701   1.683
## medianIncome:oceanProximityNEAR OCEAN      -143.401        845.553  -0.170
##                                                 Pr(>|t|)
## (Intercept)                          0.00007665005233137 ***
## longitude                            0.00005058050980433 ***
## latitude                                        0.000134 ***
## housingMedianAge                   < 0.0000000000000002 ***
## medianIncome                                    0.291028
## bedroomsPerRoom                      0.00000767882042247 ***
## incomePerRoom                        0.00000005278953201 ***
## distToLA                           < 0.0000000000000002 ***
## distToSF                             0.00000000000069740 ***
## cosDirToLA                         < 0.0000000000000002 ***
## sinDirToLA                         < 0.0000000000000002 ***
## cosDirToSF                         < 0.0000000000000002 ***
## sinDirToSF                                      0.011423 *
## cityProximityScore                              0.428837
## distToCenter                         0.00000953331448664 ***
## I(medianIncome^2)                  < 0.0000000000000002 ***
```

```
## I(medianIncome^3)                    < 0.0000000000000002 ***
## I(housingMedianAge^2)                             0.000106 ***
## I(medianIncome/housingMedianAge)                  0.060419 .
## I(latitude^2)                                     0.047626 *
## I(longitude^2)                     0.00004732621765700 ***
## I(latitude^3)                                     0.008167 **
## I(longitude^3)                     0.00005938170691358 ***
## log(population)                      < 0.0000000000000002 ***
## log(incomePerRoom)                                0.000232 ***
## log(medianIncome/distToCenter)     0.00004490263236826 ***
## oceanProximityINLAND               0.00000094379180381 ***
## oceanProximityISLAND                              0.067200 .
## oceanProximityNEAR BAY                            0.863069
## oceanProximityNEAR OCEAN                          0.004632 **
## longitude:latitude                 0.00000000000000182 ***
## medianIncome:bedroomsPerRoom         < 0.0000000000000002 ***
## housingMedianAge:medianIncome                     0.001194 **
## medianIncome:distToCenter                         0.142541
## latitude:incomePerRoom                            0.006459 **
## bedroomsPerRoom:distToLA                          0.012333 *
## medianIncome:log(population)       0.00000000000003893 ***
## medianIncome:cosDirToLA                           0.003777 **
## medianIncome:sinDirToLA              < 0.0000000000000002 ***
## medianIncome:cosDirToSF            0.00000000489202233 ***
## medianIncome:sinDirToSF                           0.208285
## housingMedianAge:cityProximityScore  0.00000292365690275 ***
## bedroomsPerRoom:cityProximityScore                0.052659 .
## housingMedianAge:incomePerRoom     0.00000000044335998 ***
## distToCenter:oceanProximityINLAND  0.00000027298631852 ***
## distToCenter:oceanProximityISLAND                 0.024272 *
## distToCenter:oceanProximityNEAR BAY               0.870880
## distToCenter:oceanProximityNEAR OCEAN             0.972020
## medianIncome:oceanProximityINLAND  0.00003543258237127 ***
## medianIncome:oceanProximityISLAND                 0.023161 *
## medianIncome:oceanProximityNEAR BAY               0.092343 .
## medianIncome:oceanProximityNEAR OCEAN             0.865332
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63160 on 18398 degrees of freedom
## Multiple R-squared:  0.7029, Adjusted R-squared:  0.7021
## F-statistic: 853.4 on 51 and 18398 DF,  p-value: < 0.00000000000000022
```

- Median Income
  - Median income is positively correlated with median house value, this holds true because as house buyers have a higher income they will have a greater tendency to purchase a house with a higher value to match what they can afford. Areas with higher income levels usually have residents with greater purchasing power. This translates into a higher demand for better and more expensive housing, which in turn drives up property values. The strong statistical significance of median income validates its role as an predictor of house values. Additionally, the p-value of this predictor in the model is essentially 0 ($<$ 2e-16) signifying its statistical significance in the model at all levels of significance.
- Ocean Proximity - Inland

  - This variable reflects the classification of a households ocean proximity. The other classifications are closer to the ocean and generally, as a result, have higher house prices due to desirability of ocean views and proximity to recreational activities with the ocean. In contrast, "inland" ocean proximity does not typically benefit from these types of coastal premiums. This difference is what makes this variable particularly significant in our model when classifying houses and determining their house values. Additionally, the p-value of this predictor in the model is essentially 0 (9.44e-07), signifying its statistical significance in the model at all levels of significance.

- Interaction between Housing Median Age and City Proximity Score
  - City proximity score is positively correlated with median house value, and weakly so with housing median age. Their interaction tests whether proximity to major cities boosts or dampens the value of older housing stock. Their interaction tests whether proximity to major cities boosts or dampens the value of older housing stock. In other words, the interaction term enables us to explore whether the effect of a house's age on its value depends on how close the property is to a major city. Additionally, the p-value of this predictor in the model is essentially 0 (2.92e-06), signifying its statistical significance in the model at all levels of significance.

---

3 Assessing the model performance

3.1 Training MSE

```
# Compute training MSE
# Initial model
# Need to reload data since new predictor columns have been added
initial_data <- read.csv("Assignt1_data.csv")
initial_data <- na.omit(initial_data)
initial_fit <- lm(medianHouseValue ~ . - id, data = initial_data)
initial_predictions <- predict(initial_fit, newdata = initial_data)
initial_train_mse <- mean((initial_data$medianHouseValue - initial_predictions)^2)
initial_train_mse
```

```
## [1] 5129801167
```

```
# Final Model
final_predictions <- predict(model.best, newdata=data)
final_train_mse <- mean((data$medianHouseValue - final_predictions)^2)
final_train_mse
```

```
## [1] 3977659474
```

3.2 Validation set MSE 80-20 split

```
# Set seed required for this approach to get the 80-20 split
set.seed(123)
```

```r
# Split data - 80% for training and 20% for validation
n <- nrow(data)
train_indicies <- sample(1:n, size=floor(0.8 * n))
train_set <- data[train_indicies, ]
validation_set <- data[-train_indicies, ]

train_set_init <- initial_data[train_indicies, ]
validation_set_init <- initial_data[-train_indicies, ]

# Initial model
initial_fit_train <- lm(medianHouseValue ~ . - id, data=train_set_init)
initial_val_predictions <- predict(initial_fit_train, newdata=validation_set_init)
initial_val_mse <- mean((validation_set$medianHouseValue - initial_val_predictions)^2)
initial_val_mse
```

```
## [1] 5211492438
```

```r
# Final model
final_fit_train <- lm(new.formula, data=train_set)
final_val_predictions <- predict(final_fit_train, newdata=validation_set)
final_val_mse <- mean((validation_set$medianHouseValue - final_val_predictions)^2)
final_val_mse
```

```
## [1] 3895087989
```

3.3 5 fold CV MSE

```r
set.seed(123)

# Define number of folds
k <- 5
n <- nrow(data)
folds <- sample(1:k, n, replace=TRUE)

# Initial model
# Vector to hold MSE for each fold
initial_cv_errors <- numeric(k)
final_cv_errors <- numeric(k)


for (i in 1:k) {

  train_indicies <- which(folds != i)
  test_indicies <- which(folds == i)

  train_data <- data[train_indicies, ]
  test_data <- data[test_indicies, ]

  train_data_init <- initial_data[train_indicies, ]
  test_data_init <- initial_data[test_indicies, ]
```

```
  # Initial Model
  initial_cv_model <- lm(medianHouseValue ~ . - id, data=train_data_init)
  initial_predictions <- predict(initial_cv_model, newdata=test_data_init)
  initial_cv_errors[i] <- mean((test_data_init$medianHouseValue - initial_predictions)^2)

  # Final Model
  final_cv_model <- lm(new.formula, data=train_data)
  final_predictions <- predict(final_cv_model, newdata=test_data)
  final_cv_errors[i] <- mean((test_data$medianHouseValue - final_predictions)^2)

}

initial_cv_5fold_mse <- mean(initial_cv_errors)
initial_cv_5fold_mse
```

```
## [1] 5153080811
```

```
final_cv_5fold_mse <- mean(final_cv_errors)
final_cv_5fold_mse
```

```
## [1] 4017021575
```

3.4 LOOCV MSE

```
set.seed(123)

# Initial Model
train_control <- trainControl(method = "LOOCV")
initial_loocv_model <- train(
  medianHouseValue ~ . - id,
  data = initial_data,
  method = "lm",
  trControl = train_control
)

# Extract RMSE
initial_loocv_rmse <- initial_loocv_model$results$RMSE


# Final Model
train_control <- trainControl(method = "LOOCV")
final_loocv_model <- train(
  new.formula,
  data = data,
  method = "lm",
  trControl = train_control
)
final_loocv_rmse <- final_loocv_model$results$RMSE

# Calculate MSE from RMSE
```

```r
initial_loocv_mse <- initial_loocv_rmse^2
initial_loocv_mse
```

```
## [1] 5163153328
```

```r
final_loocv_mse <- final_loocv_rmse^2
final_loocv_mse
```

```
## [1] 4013601583
```

3.5 Comparison

```r
initial_fit <- lm(medianHouseValue ~ . - id, data = data)
final_fit   <- lm(model.best, data = data)

AIC_initial <- AIC(initial_fit)
AIC_final   <- AIC(final_fit)

BIC_initial <- BIC(initial_fit)
BIC_final   <- BIC(final_fit)

cat("Initial Model: AIC =", AIC_initial, ", BIC =", BIC_initial, "\n")
```

```
## Initial Model: AIC = 462483.2 , BIC = 462663.1
```

```r
cat("Final Model:   AIC =", AIC_final, ", BIC =", BIC_final, "\n")
```

```
## Final Model:   AIC = 460282.9 , BIC = 460697.5
```

The final MLR model is better than the initial MLR model. This is clearly evident by the train mean squared error and the test error rates, under all of the methods covered above, being lower for the final MLR model when compared to the initial MLR model.

A well-performing model must strike a balance between bias (error from assumptions in the model) and variance (error from sensitivity from the data set), and this bias-variance trade-off was an important considerations when determining which model was more effective than the other. The initial MLR model was too simple and failed to capture important relationships present in the data, whereas the final model-by including additional predictors, interaction terms, and by refining the variable selection we reduced the variance of the model without introducing excessive bias. This improved balance lowered our test error rates as calculated above.

The use of the validation techniques (80-20 split validation set approach, 5-fold CV, and LOOCV) provided more robust estimates of the test error than the training mse. The final model's lower error rates in each of the 3 tests mentioned demonstrates our final models predictive performance is pretty reliable. Additionally, the 2 cross validation methods used to derive the test error rates for our model, are known for providing a close to unbiased estimate of the test error which helped when comparing the models. By the final model showing that it consistently achieves a lower test error rates in all of the approaches mentioned it provides even stronger evidence to support the strong predictive accuracy of the final model over the initial model.

The final model also had a lower AIC (Akaike Information Criterion, 460282.9 vs. 462,438.2) and a lower BIC (Bayesian Information Criterion, 460697.5 vs. 462,663.1) compared to the initial model, indicating that

the final model also achieves a better trade-off between fit and complexity. Although the difference may not seem very large it is significant to note that the final model is a lot more complex than the initial model but it still has a much better fit to the data, without introducing too much bias.

The final model was adapted to our knowledge of the data (such as ocean proximity and the city proximity score) to make sense from an economic and geographical perspective of the prediction task at hand. This supports the idea that the final model not only fits with the data better but it also aligns with theoretical expectations of how house values interact with these economic and geographic factors.

Finally, the final model was also created with use of the best subset selection method helps evaluates the most appropriate predictors for the model. This method evaluated which predictors add the most substantive explanatory power, as these predictors will be the most impactful to the predictive accuracy of our model. The results of this method of evaluating predictors also aligned with both statistical evidence and the knowledge of the data, as mentioned prior, further justifying the superiority of the predictors used in our final MLR over the initial MLR.

---

4 A Prediction Competition

4.1 Test MSE for the final (best) model

```
test_data <- read.csv("Assignt1_test full.csv")
test_data$bedroomsPerRoom <- test_data $aveBedrooms / test_data $aveRooms
test_data$distToCenter <- sqrt((test_data$latitude - center_lat)^2 +
                                 (test_data$longitude - center_lon)^2)
test_data$incomePerRoom = test_data$medianIncome / test_data$aveRoom
# Add distance to LA
test_data$distToLA <- apply(test_data[, c("longitude", "latitude")], 1, function(coord) {
  distGeo(coord, la_coords) / 1000  # convert meters to km
})

#Add distance to SF
test_data$distToSF <- apply(test_data[, c("longitude", "latitude")], 1, function(coord) {
  distGeo(coord, sf_coords) / 1000
})

# Compute direction angles
test_data$dirToLA <- atan2(test_data$latitude - la_coords[2],
                           test_data$longitude - la_coords[1])
test_data$dirToSF <- atan2(test_data$latitude - sf_coords[2],
                           test_data$longitude - sf_coords[1])


# Encode directions using sin and cos
test_data$cosDirToLA <- cos(test_data$dirToLA)
test_data$sinDirToLA <- sin(test_data$dirToLA)

test_data$cosDirToSF <- cos(test_data$dirToSF)
test_data$sinDirToSF <- sin(test_data$dirToSF)

# Remove intermediate angle variables
```

```r
test_data$dirToLA <- NULL
test_data$dirToSF <- NULL

test_data$cityProximityScore <- 1 / (1 + test_data$distToLA) + 1 / (1 + test_data$distToSF)
test_data$incomeFlag <- ifelse(test_data$medianIncome == 15.0001, 1, 0)

actual <- test_data$medianHouseValue

#Best model

pred <- predict(model.best, newdata = test_data  )
pred <- pmax(pmin(pred,500001),14999) # apply censoring
best_mse <- mean((pred - actual)^2)

cat("This is the mean squared error of our final MLR model for the competition \n",best_mse)
```

```
## This is the mean squared error of our final MLR model for the competition
##  3763439107
```

We loaded the test dataset and setup our final MLR model to predict the median housing prices. Firstly, we calculated our new predictors to the new data. Secondly, we used our final MLR model and predicted the median house values using the test data, from which we could easily calculate our mean squared error for the competition as shown below. We also applied censoring on the median housing prices to replicate the style of the dataset.

G Appendix: AAD Assignment 2

Refer to next page.

Group-26 AAD-Assignment-2

Omar, Eloise, Alina, Sue

2025-05-21

Contents

2.1 Nonlinear model vs non-parametric model

2.1.1 GAM vs KNN approach

GAM:

Pros

- **Flexibility to capture non-linearity**. In a GAM we replace each linear term $\beta_k x_k$ with a smooth function $f_k(x_k)$, meaning we can automatically model non-linear relationships between each predictor and the response without having to manually try out different transformation on each variable individually.
- **Non-linear fits may be more accurate**. The ability of creating this non-linear predictor and response relationship may make our model more accurate when predicting the medianHousingValue.
- **Interperability from additivity**. Since a GAM is additive, we can examine the effect of each X_j on Y individually while holding all of the other variables fixed. Therefore, we can understand each variables individual effect on house value.
- **Control over smoothness**. Each f_j comes with an associated smoothing parameter (or degrees of freedom), making it straightforward to trade bias and variance (e.g. via cross-validation).

Cons

- **Additivity assumption restriction**. If there are strong synergistic effects like between location (longitude/latitude) and income –an additive model will not capture them unless explicit interaction terms $X_j \times X_k$ are included or low-dimensional interactions function of the form $f_{jk}(X_j, X_k)$ are manually introduced.
- **Computational cost**. Fitting this model to over ~$20,600$ observations and multiple smoothers can be very slow when compared to fitting a single parametric method.

KNN:

Pros

- **Completely non-parametric**. KNN makes no assumptions about the form of $f(X)$, allowing the model to potentially fit better than a parametric model. At a point x_0 KNN averages the responses of the K closest training blocks: $\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i$, where N_0 is the set of the K nearest neighbours.
- **Control of bias-variance**. A small K yields a very flexible, low-bias but high-variance fit; large K yields a smoother, lower-variance fit.

Cons

- **Dimensoniality constraints**. As the number of predictors grows, the "nearest" neighbours tend to be far away in a high-dimensional space, so KNN's performance degrades rapidly as the number of predictors grow.
- **Distance metric sensitivity**. With a KNN you must scale the numeric features and encode the categorical features (e.g. oceanProximity) carefully. Otherwise poorly scaled or encoded features can dominate the distance of the nearest neighbours calculation.
- **Computationally intensive with predictions**. For each new group of predictions all ~$20,600$ must be computed to find the K nearest, which can be very intensive and slow.
- **Low interperability**. There is no simple way to explain a KNN prediction beyond pointing to the raw neighbours and their average.

In this housing-price context:

- **GAM:** it is likely to give an interpretable model with, as we can analyse the partial-effects (e.g. how median income or ocean proximity individually affects price), and we can capture smooth non-linear trends.

- **KNN:** can capture complex interactions, between predictors, automatically, but with eight predictors (including a categorical one) it may run into high-dimensionality issues, making distance-based averaging unstable and slow.

2.1.2 GAM vs KNN Regression Model

We will use the cleaned data (omitting all data points that contains "NA") instead of the original raw dataset. We split the data into a training set (80% of the original data) and a test set (20% of the original data).

```
# Load the data
data <- read.csv("Assignt2_data.csv")
data <- na.omit(data)
```

```
# Separate the training and test set out
set.seed(1)
train_index <- createDataPartition(data$medianHouseValue, p = 0.8, list = FALSE)
train_data <- data[train_index, ]
test_data  <- data[-train_index, ]
```

We use the same training and test set for this whole report.

Basic KNN Regression Model

The basic KNN regression model was trained using 5-fold cross-validation on the training dataset, with k tuned from 1 to 10. Preprocessing steps included centering and scaling to ensure features contribute equally to the distance metric.

```
# Basic KNN

knn_reg_model_1 <- train(
  medianHouseValue ~ .-id,
  data = train_data,
  method = "knn",
  preProcess = c("center", "scale"),
  tuneGrid = data.frame(k = 1:10),  # test k = 1 to 10
  trControl = trainControl(method = "cv", number = 5)
)

# Best k
print(knn_reg_model_1)
```

```
## k-Nearest Neighbors
##
## 16348 samples
##     9 predictor
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 13080, 13078, 13077, 13079, 13078
## Resampling results across tuning parameters:
```

```
##
##    k   RMSE      Rsquared   MAE
##    1  81963.63  0.5511857  54420.18
##    2  71586.59  0.6278105  48377.51
##    3  68378.57  0.6530505  46328.44
##    4  66666.50  0.6677736  45236.72
##    5  65783.58  0.6754535  44650.50
##    6  65241.29  0.6803072  44329.60
##    7  65008.23  0.6824027  44252.06
##    8  64925.97  0.6832080  44216.47
##    9  64762.67  0.6848104  44167.47
##   10  64546.74  0.6869904  44061.75
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 10.
```

```r
# KNN MSE calculation
knn_pred_1 <- predict(knn_reg_model_1, newdata = test_data)
knn_MSE_1 <- mean((knn_pred_1 - test_data$medianHouseValue)^2)
cat("The test MSE of basic KNN model is", knn_MSE_1, "\n")
```

```
## The test MSE of basic KNN model is 3983322872
```

KNN Model Improvements

Addition and modification of features:

- Following the arguments in Assignment1, we've added the following features to the KNN regression model:
  - **bedroomsPerRoom**: captures housing density within units; higher ratios may indicate crowding, which could affect property value.
  - **incomePerRoom**: reflects economic status relative to housing size, providing a measure of affordability or wealth.
  - **distToLA** and **distToSF**: quantify proximity to major cities, which often correlates with higher housing prices.
  - **cosDirToLA/SF** and **sinDirToLA/SF**: encode directional information relative to LA and SF to help KNN detect spatial orientation patterns.
  - **cityProximityScore**: combines distance to both cities into a single metric, giving higher scores to locations near either city.
- We also encoded the categorical data to numeric form specifically for KNN so it can properly use it for distance calculations.
  - Converted the categorical variable `oceanProximity` to a factor to ensure proper handling of categories.
  - Applied one-hot encoding using `dummyVars()` to convert categorical variables into binary indicator variables.
  - Transformed the one-hot encoded matrices back into data frames for compatibility with modeling functions.

How we used the data:

- The KNN regression model was trained using the encoded and preprocessed training data (`train_data_encoded`), which includes both numeric and one-hot encoded categorical variables.
- We standardized features by centering and scaling (`preProcess = c("center", "scale")`) to ensure that distance calculations are not affected by different sizes of units.
- We selected the best `k` by performing cross validation (5-fold) over the range 1 to 15 (`tuneGrid = data.frame(k = 1:15)`).

```r
# Import and clean data to get data_imp for model improvement
data_imp <- read.csv("Assignt2_data.csv")
data_imp <- na.omit(data_imp)

# bedroomsPerRoom
data_imp$bedroomsPerRoom <- data_imp$aveBedrooms / data_imp$aveRooms

# incomePerRoom
data_imp$incomePerRoom = data_imp$medianIncome / data_imp$aveRooms

# Compute distances
la_coords <- c(-118.24, 34.05)   # (longitude, latitude)
sf_coords <- c(-122.42, 37.77)

data_imp$distToLA <- apply(data_imp[, c("longitude", "latitude")], 1, function(coord) {
  distGeo(coord, la_coords) / 1000
})
data_imp$distToSF <- apply(data_imp[, c("longitude", "latitude")], 1, function(coord) {
  distGeo(coord, sf_coords) / 1000
})

# Compute directions
data_imp$dirToLA <- atan2(data_imp$latitude - la_coords[2], data_imp$longitude - la_coords[1])
data_imp$dirToSF <- atan2(data_imp$latitude - sf_coords[2], data_imp$longitude - sf_coords[1])

# Cosine and sine of direction
data_imp$cosDirToLA <- cos(data_imp$dirToLA)
data_imp$sinDirToLA <- sin(data_imp$dirToLA)
data_imp$cosDirToSF <- cos(data_imp$dirToSF)
data_imp$sinDirToSF <- sin(data_imp$dirToSF)
data_imp$dirToLA <- NULL
data_imp$dirToSF <- NULL

# Composite proximity score
data_imp$cityProximityScore <- 1 / (1 + data_imp$distToLA) + 1 / (1 + data_imp$distToSF)

# Add these to test and train
# Using the same training and test sets from before
train_data_imp <- data_imp[train_index, ]
test_data_imp  <- data_imp[-train_index, ]
```

```r
# KNN with model improvements cont.

# Hot encoding the variables
```

```r
# Convert oceanProximity to factor
train_data_imp$oceanProximity <- as.factor(train_data_imp$oceanProximity)
test_data_imp$oceanProximity <- as.factor(test_data_imp$oceanProximity)

# One-hot encode using dummyVars
dummies <- dummyVars(~ ., data = train_data_imp)
train_data_encoded <- predict(dummies, newdata = train_data_imp)
test_data_encoded <- predict(dummies, newdata = test_data_imp)

# Convert to data frame
train_data_encoded <- as.data.frame(train_data_encoded)
test_data_encoded <- as.data.frame(test_data_encoded)

# KNN regression

set.seed(1)
knn_reg_model_2 <- train(
  medianHouseValue ~ .-id,
  data = train_data_encoded,
  method = "knn",
  preProcess = c("center", "scale"),
  tuneGrid = data.frame(k = 1:15),  # test k = 1 to 15
  trControl = trainControl(method = "cv", number = 5)
)

# Best k
print(knn_reg_model_2)
```

```
## k-Nearest Neighbors
##
## 16348 samples
##    22 predictor
##
## Pre-processing: centered (21), scaled (21)
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 13079, 13079, 13079, 13077, 13078
## Resampling results across tuning parameters:
##
##   k   RMSE      Rsquared   MAE
##    1  67229.74  0.6842298  43172.00
##    2  58960.44  0.7434965  38523.64
##    3  56746.61  0.7595084  37237.09
##    4  55717.26  0.7673094  36641.96
##    5  55099.02  0.7721771  36325.34
##    6  54659.74  0.7757243  36103.21
##    7  54361.30  0.7782753  36030.08
##    8  54432.13  0.7777877  36060.89
##    9  54362.43  0.7785184  36049.82
##   10  54411.06  0.7782571  36095.97
##   11  54567.02  0.7771765  36175.67
##   12  54768.53  0.7756700  36345.60
##   13  54960.45  0.7742252  36484.03
##   14  55048.35  0.7737057  36533.66
```

```
##   15  55165.68  0.7728772  36639.39
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.
```

```
# KNN MSE calculation
knn_pred_2 <- predict(knn_reg_model_2, newdata = test_data_encoded)
knn_MSE_2 <- mean((knn_pred_2 - test_data_encoded$medianHouseValue)^2)
cat("The KNN MSE (improved) is", knn_MSE_2, "\n")
```

```
## The KNN MSE (improved) is 3060856986
```

Basic GAM

- We used the same 80/20 train-test split as before to ensure consistency.
- The GAM model was trained using smoothing splines on original predictors to capture possible non-linear relationships.
- The select = TRUE argument enables automatic feature selection by penalising and shrinking irrelevant smooth terms toward zero. This is analogous to tuning the hyperparameter k in the Basic KNN regression, as both mechanisms helps to control model complexity and prevent overfitting.

```
# GAM

# train the GAM
set.seed(1)
gam_model_1 <- mgcv::gam(medianHouseValue ~ s(latitude) +
                           s(longitude) +
                    s(housingMedianAge) +
                      s(aveRooms) +
                    s(aveBedrooms) +
                      s(population) +
                    s(medianIncome) +
                      factor(oceanProximity),
                    data = train_data,
                    select = TRUE)

# Calculate MSE
gam_pred_1 <- predict(gam_model_1, newdata = test_data)
gam_MSE_1 <- mean((gam_pred_1 - test_data$medianHouseValue)^2)
cat("The Basic GAM MSE is", gam_MSE_1, "\n")
```

```
## The Basic GAM MSE is 4256244455
```

Improved GAM

- Similar to the improved KNN Regression, the improved GAM model incorporates new features such as city proximity scores, distances, and directional terms to better capture spatial and socioeconomic effects.
- We also included a tensor product smooth, te(longitude, latitude), to flexibly capture complex spatial interactions between longitude and latitude.

```r
# GAM improvements
train_data_imp = data_imp[train_index, ]
set.seed(1)
gam_model_2 <- mgcv::gam(medianHouseValue ~
                      s(housingMedianAge) +
                      s(aveRooms) +
                      s(aveBedrooms) +
                      s(population) +
                      s(medianIncome) +
                      te(longitude, latitude) +
                      s(cityProximityScore) +
                      s(bedroomsPerRoom) +
                      s(incomePerRoom) +
                      s(distToSF) +
                      s(distToLA) +
                      s(cosDirToLA) +
                      s(cosDirToSF) +
                      s(sinDirToLA) +
                      s(sinDirToSF),
                    data = train_data_imp,
                    select = TRUE)


# Calculate MSE
test_data_imp = data_imp[-train_index, ]
gam_pred_2 <- predict(gam_model_2, newdata = test_data_imp)
gam_MSE_2 <- mean((gam_pred_2 - test_data_imp$medianHouseValue)^2)
cat("The GAM MSE (improved) is", gam_MSE_2, "\n")
```

```
## The GAM MSE (improved) is 3386284239
```

2.1.3 Which model performs better?

Without adding any new features, KNN regression has a lower test MSE (3.98 billion) that is about 7% lower than the GAM (test MSE = 4.26 billion). The baseline models were tuned - KNN with 5-fold cross-validation to select the optimal k, and GAM with automatic feature selection.

After introducing additional features and encoding the categorical variables, the KNN model's MSE dropped substantially by about 23% (to 3.06 billion). The MSE for GAM also reduced by 20% (to 3.39 billion). However, the KNN model still outperforms the GAM by a 10% lower test MSE.

KNN outperforms GAM both before and after the model improvements because its completely non-parametric nature that makes no assumptions about the functional form and averages responses of nearest neighbours (recall section 2.1.1 KNN Pros). This flexibility allows it to capture complex interactions introduced by new features such as cityProximityScore and the directional terms.

On the other hand, GAM is limited by its additive structure and smoothing assumptions, which restrict its ability to capture strong synergistic effects unless interactions are explicitly modelled (section 2.1.1 GAM Cons). Additionally, GAM's exponential family assumption (Gaussian here) on the response variable limits its flexibility to fully model complex, censored, and non-normal housing price data, making it less suitable compared to nonparametric models like KNN.

2.2 Classification models

2.2.1 New censoring boolean column

```r
# Add censoring column and drop id and medianHouseValue
data_c <- data %>%
  na.omit() %>%
  mutate(censoring = ifelse(medianHouseValue == 500001, 1, 0)) %>%
  dplyr::select(-id, -medianHouseValue) %>%
  mutate(oceanProximity = as.factor(oceanProximity))



# This dataframe does the same changes as above but is with our model improvements
data_c_imp <- data_imp %>%
  na.omit() %>%
  mutate(censoring = ifelse(medianHouseValue == 500001, 1, 0)) %>%
  dplyr::select(-id, -medianHouseValue) %>%
  mutate(oceanProximity = as.factor(oceanProximity))
```

2.2.2 Two classification methods

Logistic Regression:

In this case logistic regression can model the log-odds of hitting the $\$500,001$ cap (censoring $= 1$) as a linear function of the predictors:

$$\log \frac{Pr(\text{censor} = 1|X_1, ..., X_p)}{1 - Pr(\text{censor} = 1|X_1, ..., X_p)} = \beta_0 + \beta_1 \text{ medianIncome} + \beta_2 \text{ housingMedianAge} + ...$$

This method is reasonable because:

1. **Ideal for binary classification problems**: Designed specifically for binary response variables, logistic regression is suitable since in this case censoring take values of either 1 (censored) or 0 (not censored).
2. **No strong distributional assumptions on $\boldsymbol{X}$**: Some of our variables may be skewed or heteroscedastic; logistic regression simply cares that the logit is linear in $\boldsymbol{X}$ (where $\boldsymbol{X} = (X_1, ..., X_p)$).
3. **Interpretable effects**: e.g. e^{β_1} tells us how one unit $(\$1,000)$ increases in medianIncome multiplies the odds of a house price being censored.
4. **Flexibility**: It estimates the probability that a given input belongs to a particular class, which also provides us with the flexibility of choosing the threshold for which optimises the testing correct rate with the assistance of the ROC curve.

KNN-CV:

In this case KNN-CV is a non-parametric approach that classifies each block group by majority voting among its k closest neighbours in the predictor space. Formally, for a test point x_0, let N_0 be the indices of the k nearest training observations (by Euclidean distance):

$$Pr(\text{censor} = 1|x_0) = \frac{1}{k} \sum_{i \in N_0} I(\text{censor}_i = 1)$$

and we predict the class with the larger estimated probability. We choose k via cross-validation to pick the optimal bias-variance trade-off.

This method is reasonable because:

1. **Captures nonlinearity**: If censoring depends on complex interactions and requires a non-linear decision boundary, KNN can adapt without specifying a specific model form.
2. **No parametric assumptions**: No distributional assumptions required–so it's robust to skewed, heteroscedastic variables.
3. **Tunable bias-variance trade-off via cross-validation**: The ability to choose k allows us to control the smoothness.
    - Small $k \implies$ very flexible (low bias, high variance)
    - Large $k \implies$ smoother (high bias, low variance)
    - Using cross-validation empirically estimates the error for each k, letting us pick a k value the minimises both over and under fitting

2.2.3 Which classification method performs better?

How we have used the given data:

- We used a random 80/20 training and test split of the dataset to produce the test error rates.
- We dropped the *id* and *medianHouseValue* columns to ensure we only use legitimate predictors when training and testing our model.
- The logistic model thresholds at 0.5, while KNN makes decisions based on the majority class among the nearest neighbours in the feature space.

```
# Logistic Regression without Model Improvements

train_data_c <- data_c[train_index, ]
test_data_c <- data_c[-train_index, ]

logit_fit_1 <- glm(censoring ~ longitude +
                     latitude +
                     medianIncome +
                     housingMedianAge +
                     oceanProximity +
                     aveRooms +
                     aveBedrooms,
                   data = train_data_c,
                   family = binomial)

logit_pred_1 <- as.numeric(predict(logit_fit_1,
                       newdata = test_data_c,
                       type = "response") > 0.5) # use 0.5 for now

error_rate <- mean(logit_pred_1 != test_data_c$censoring)

print(paste("Logistic Regression Prediction Error Rate:", round(error_rate, 3)))
```

```
## [1] "Logistic Regression Prediction Error Rate: 0.031"
```

```
# Confusion matrix
confusion_matrix <- confusionMatrix(as.factor(logit_pred_1), as.factor(test_data_c$censoring))
print(confusion_matrix)
```

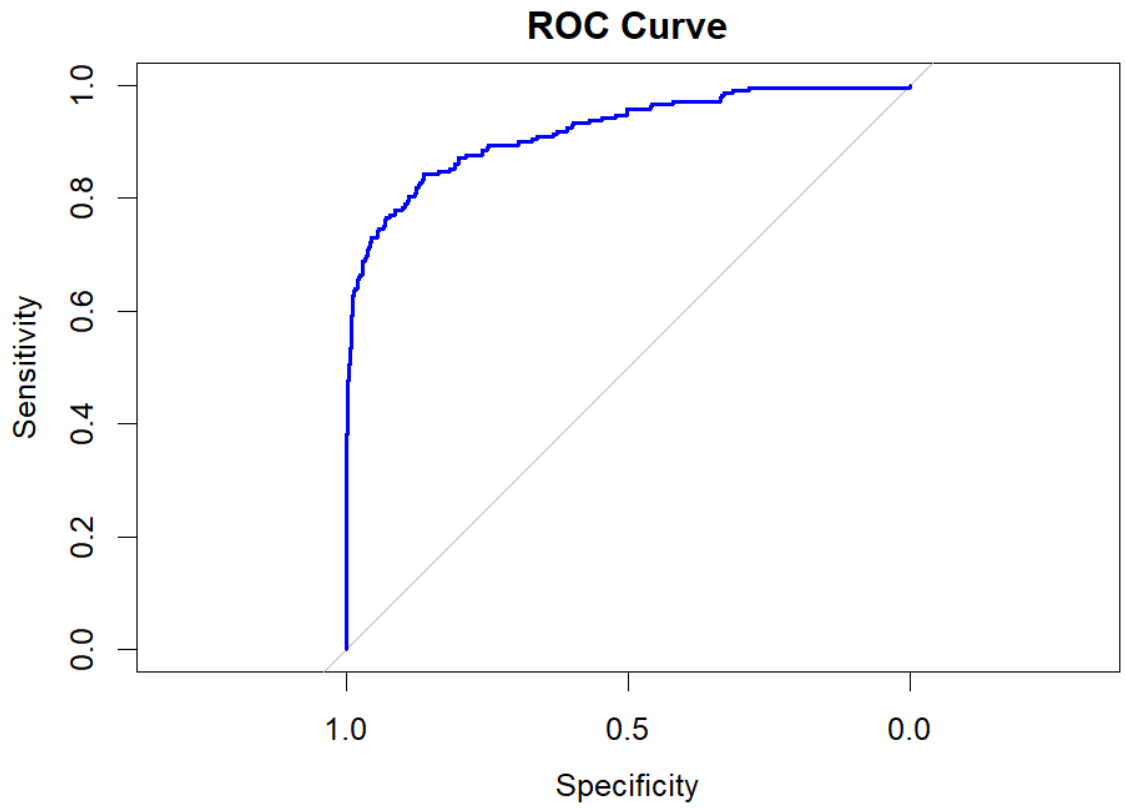
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3857  107
##          1   20  101
##
##                Accuracy : 0.9689
##                  95% CI : (0.9631, 0.974)
##     No Information Rate : 0.9491
##     P-Value [Acc > NIR] : 4.094e-10
##
##                   Kappa : 0.599
##
##  Mcnemar's Test P-Value : 2.325e-14
##
##             Sensitivity : 0.9948
##             Specificity : 0.4856
##          Pos Pred Value : 0.9730
##          Neg Pred Value : 0.8347
##              Prevalence : 0.9491
##          Detection Rate : 0.9442
##    Detection Prevalence : 0.9704
##       Balanced Accuracy : 0.7402
##
##        'Positive' Class : 0
##
```

```
# ROC
logit_pred_probs_1 <- predict(logit_fit_1, newdata = test_data_c, type = "response")
roc_curve_1 <- roc(test_data_c$censoring, logit_pred_probs_1)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot(roc_curve_1, main = "ROC Curve", col = "blue", lwd = 2)
```




```r
# Logistic Regression with Model Improvements

train_data_c_imp <- data_c_imp[train_index, ]
test_data_c_imp <- data_c_imp[-train_index, ]

logit_fit_2 <- glm(censoring ~ longitude +
                     latitude +
                     medianIncome +
                     housingMedianAge +
                     oceanProximity +
                     aveRooms +
                     aveBedrooms +
                     cityProximityScore +
                     bedroomsPerRoom +
                     incomePerRoom +
                     distToSF +
                     distToLA +
                     cosDirToLA +
                     cosDirToSF +
                     sinDirToLA +
                     sinDirToSF,
                   data = train_data_c_imp,
                   family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
logit_pred_2 <- as.numeric(predict(logit_fit_2,
                      newdata = test_data_c_imp,
                      type = "response") > 0.5) # use 0.5 for now

error_rate <- mean(logit_pred_2 != test_data_c_imp$censoring)

print(paste("Logistic Regression Prediction Error Rate:", round(error_rate, 3)))
```

```
## [1] "Logistic Regression Prediction Error Rate: 0.029"
```

```
# Confusion matrix
confusion_matrix <- confusionMatrix(as.factor(logit_pred_2), as.factor(test_data_c_imp$censoring))
print(confusion_matrix)
```

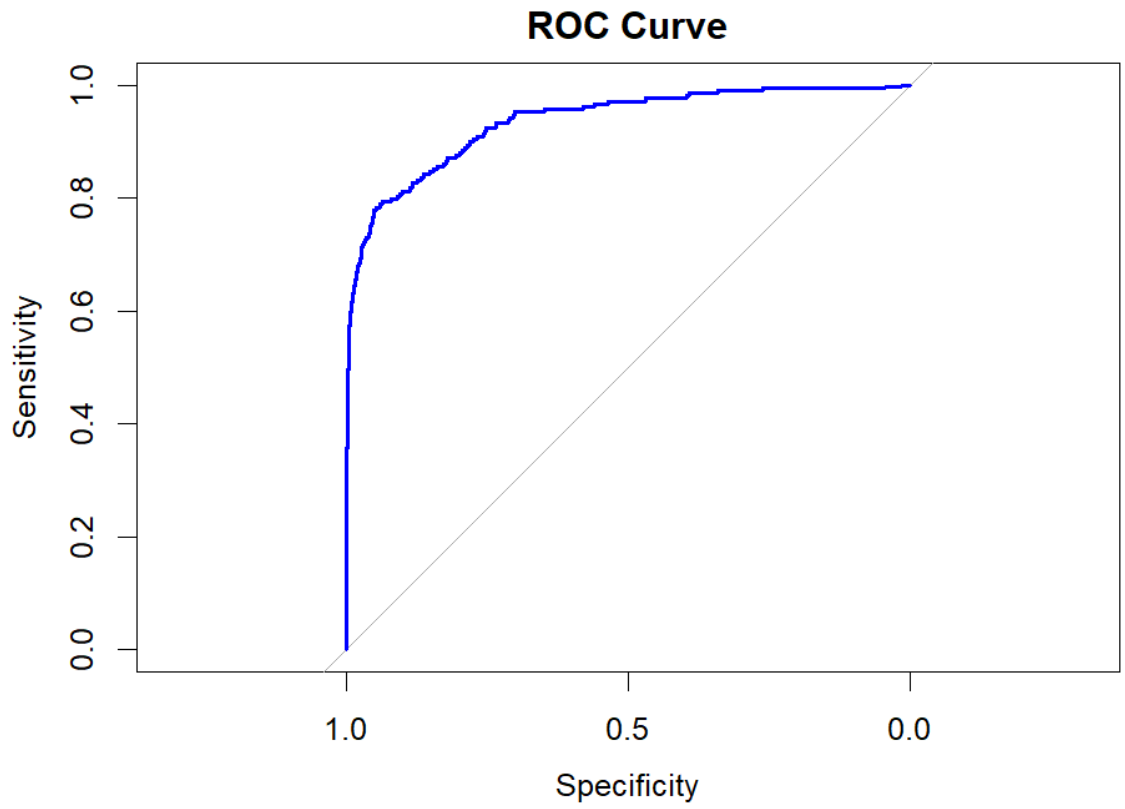
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3854   95
##          1   23  113
##
##                Accuracy : 0.9711
##                  95% CI : (0.9655, 0.976)
##     No Information Rate : 0.9491
##     P-Value [Acc > NIR] : 2.686e-12
##
##                   Kappa : 0.6426
##
##  Mcnemar's Test P-Value : 6.315e-11
##
##             Sensitivity : 0.9941
##             Specificity : 0.5433
##          Pos Pred Value : 0.9759
##          Neg Pred Value : 0.8309
##              Prevalence : 0.9491
##          Detection Rate : 0.9435
##    Detection Prevalence : 0.9667
##       Balanced Accuracy : 0.7687
##
##        'Positive' Class : 0
##
```

```
# ROC
logit_pred_probs_2 <- predict(logit_fit_2, newdata = test_data_c_imp, type = "response")
roc_curve_2 <- roc(test_data_c_imp$censoring, logit_pred_probs_2)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```r
plot(roc_curve_2, main = "ROC Curve", col = "blue", lwd = 2)
```




```r
## KNN-CV Without Model Improvements

# Prepare the predictors (X) and response (Y)
train.X <- model.matrix(~ longitude +
                          latitude +
                          housingMedianAge +
                          aveRooms +
                          aveBedrooms +
                          population +
                          medianIncome +
                          oceanProximity,
                        data = train_data_c)[, -1]

test.X <- model.matrix(~ longitude +
                         latitude +
                         housingMedianAge +
                         aveRooms +
                         aveBedrooms +
                         population +
                         medianIncome +
                         oceanProximity,
                       data = test_data_c)[, -1]
```

```r
train.Y <- train_data_c$censoring
test.Y <- test_data_c$censoring

# Use cross-validation on the training set to choose the best k
knn_rp <- rep(0, 15)
for (k in 1:15) {
  set.seed(5)
  knn.pred.cv <- knn.cv(train = train.X, cl = train.Y, k = k)
  knn_rp[k] <- mean(knn.pred.cv == train.Y)
}

k.cv <- which.max(knn_rp)
cat("Best k selected by CV is:", k.cv, "\n")
```

```
## Best k selected by CV is: 5
```

```r
# Fit and predict on the test set using the best k
set.seed(5)
knn_pred_c_1 <- knn(train = train.X, test = test.X, cl = train.Y, k = k.cv)

# Compute confusion matrix and error rate
confusionMatrix(as.factor(knn_pred_c_1), as.factor(test.Y))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3876  199
##          1    1    9
##
##                Accuracy : 0.951
##                  95% CI : (0.944, 0.9575)
##     No Information Rate : 0.9491
##     P-Value [Acc > NIR] : 0.2994
##
##                   Kappa : 0.0783
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.99974
##             Specificity : 0.04327
##          Pos Pred Value : 0.95117
##          Neg Pred Value : 0.90000
##              Prevalence : 0.94908
##          Detection Rate : 0.94884
##    Detection Prevalence : 0.99755
##       Balanced Accuracy : 0.52151
##
##        'Positive' Class : 0
##
```

```r
(knn_err_c_1 = mean(knn_pred_c_1 != test.Y))
```

```
## [1] 0.04895961
```

```r
cat("The KNN error rate is", round(knn_err_c_1, 4), "\n")
```

```
## The KNN error rate is 0.049
```

```r
## KNN-CV with Model Improvements

# Prepare the predictors (X) and response (Y)
train.X <- model.matrix(~ longitude +
                          latitude +
                          medianIncome +
                          housingMedianAge +
                          oceanProximity +
                          aveRooms +
                          aveBedrooms +
                          cityProximityScore +
                          bedroomsPerRoom +
                          incomePerRoom +
                          distToSF +
                          distToLA +
                          cosDirToLA +
                          cosDirToSF +
                          sinDirToLA +
                          sinDirToSF,
                        data = train_data_c_imp)[, -1]

test.X <- model.matrix(~  longitude +
                          latitude +
                          medianIncome +
                          housingMedianAge +
                          oceanProximity +
                          aveRooms +
                          aveBedrooms +
                          cityProximityScore +
                          bedroomsPerRoom +
                          incomePerRoom +
                          distToSF +
                          distToLA +
                          cosDirToLA +
                          cosDirToSF +
                          sinDirToLA +
                          sinDirToSF,
                       data = test_data_c_imp)[, -1]

train.Y <- train_data_c_imp$censoring
test.Y <- test_data_c_imp$censoring

# Use cross-validation on the training set to choose the best k
knn_rp <- rep(0, 15)
```

```r
for (k in 1:15) {
  set.seed(5)
  knn.pred.cv <- knn.cv(train = train.X, cl = train.Y, k = k)
  knn_rp[k] <- mean(knn.pred.cv == train.Y)
}

k.cv <- which.max(knn_rp)
cat("Best k selected by CV is:", k.cv, "\n")
```

```
## Best k selected by CV is: 5
```

```r
# Fit and predict on the test set using the best k
set.seed(5)
knn_pred_c_2 <- knn(train = train.X, test = test.X, cl = train.Y, k = k.cv)

# Compute confusion matrix and error rate
confusionMatrix(as.factor(knn_pred_c_2), as.factor(test.Y))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 3859  107
##          1   18  101
##
##                Accuracy : 0.9694
##                  95% CI : (0.9636, 0.9745)
##     No Information Rate : 0.9491
##     P-Value [Acc > NIR] : 1.419e-10
##
##                   Kappa : 0.603
##
##  Mcnemar's Test P-Value : 3.519e-15
##
##             Sensitivity : 0.9954
##             Specificity : 0.4856
##          Pos Pred Value : 0.9730
##          Neg Pred Value : 0.8487
##              Prevalence : 0.9491
##          Detection Rate : 0.9447
##    Detection Prevalence : 0.9709
##       Balanced Accuracy : 0.7405
##
##        'Positive' Class : 0
##
```

```r
(knn_err_c_2 = mean(knn_pred_c_2 != test.Y))
```

```
## [1] 0.03059976
```

```r
cat("The KNN error rate is", round(knn_err_c_2, 4), "\n")
```

```
## The KNN error rate is 0.0306
```

Note: We did not generate ROC curves for KNN because it does not provide continuous probability scores but rather hard class labels. ROC analysis requires varying the classification threshold across a range of values, which is not possible with the standard KNN output. While probabilistic adaptations of KNN exist, they were not necessary here since our objective was classification accuracy, not probability estimation.

2.2.4 Justifying the better classification method

Logistic regression outperformed KNN-CV in our 80/20 hold-out test, with test errors of 0.0290 and 0.0306 respectively. Logistic regression is a parametric model, it assumes a specific relationship between predictors and the target, resulting in higher bias but lower variance. This rigidity works in its favour when the underlying structure of the data is close to linear, as it avoids fitting noise and reduces the risk of overfitting. KNN, on the other hand, is non-parametric and makes very few assumptions about the data's form. It is flexible and can capture complex, non-linear patterns, but that flexibility comes at the cost of high variance, especially in high-dimensional settings.

While KNN's flexibility can be useful, it becomes increasingly unstable in high-dimensional spaces where data points are more sparsely distributed and neighbourhoods lose their meaning. Specifically, the distance between the nearest and farthest neighbours shrinks, making it hard to define what counts as "close," which undermines the core logic of KNN. Even with cross-validation to choose the optimal k, KNN was still overfitting, likely due to this dimensionality issue. Logistic regression, despite its linear constraint, handled the complexity of the data more gracefully by not reacting too much to the noise.

The lower test error achieved by logistic regression shows that its higher bias was actually beneficial here, it provided a smoother, more stable model that generalised well. So while KNN theoretically offers more flexibility, in this case it became a liability, and logistic regression ultimately struck the better balance. To note, the test error is only marginally larger for the KNN, and both are relatively low so it is clear that both models are effective, just for this random seed, logistic regression is the winner.

2.3 A hybrid approach

2.3.1 Dicussing feasibility of the approach

To address the censoring issue in the dataset, a hybrid approach that combines regression and classification models is both feasible and potentially very effective. The best regression model from 2.1.3 is used to predict house values, while the best classifier from 2.2.2 identifies whether a given observation is censored; i.e. whether the true house value exceeds the $500,001 threshold. This setup allows us to adjust regression predictions when the classifier indicates censoring, helping to reduce the bias that occurs when censored values are treated as if they were exact rather than right censored.

The procedure involves four main steps. First, we use the KNN regression model to predict house prices. Second, a logistic regression classifier estimates the probability of censoring. Third, we adjust the regression predictions by setting them to 500001 for observations predicted to be censored. Finally, we evaluate the performance of this adjusted model using test MSE. This combined method leverages the strengths of both models and is a practical way to improve prediction accuracy in the presence of censoring.

2.3.2 test MSE of medianHousingValue using the approach

```
# Step 1: Predict house prices with KNN regression
knn_pred_2 <- predict(knn_reg_model_2, newdata = test_data_encoded)

# Step 2: Predict censoring with logistic regression
logit_pred_2 <- as.numeric(predict(logit_fit_2, newdata = test_data_c_imp, type = "response") > 0.5)

# Step 3: Adjust predicted house prices
# If censored predicted, clip to 500001
adjusted_preds <- ifelse(logit_pred_2 == 1, 500001, knn_pred_2)

# Step 4: Compute MSE using all observations
final_mse <- mean((adjusted_preds - test_data_encoded$medianHouseValue)^2)

cat("Final Test MSE:", round(final_mse, 2), "\n")
```

```
## Final Test MSE: 3077760238
```

2.3.3 Comparison of the accuracy of this procedure to model in 2.1.3

The hybrid model resulted in a slightly higher test MSE (3.07 billion) compared to the original KNN regression model from 2.1.3 (3.06 billion). These results were obtained using a random 80/20 train-test split of the data the same split used throughout the course of this report. Given how close the two MSE values are, it's possible that on a different random sample, the hybrid model could perform slightly better. Overall, the difference is minimal, suggesting that both models offer comparable predictive performance in this context.

The reason the hybrid model does not consistently outperform KNN likely stems from how it handles censoring using a hard classification decision. Specifically, it clips predicted values to 500001 whenever the logistic classifier predicts an observation is censored. This is because logistic regression applies a strict decision boundary (at 0.5) to make this call, which introduces a sharp threshold with no gradual transition. This can be problematic near the boundary, where small changes in input may flip the classification, yet the model treats the output as definitively censored or not. If the classifier incorrectly predicts censoring, the hard cap forces the prediction to 500001, which can be significantly off from the true (uncensored) value. In contrast, the KNN model always outputs a continuous estimate. When the classifier misclassifies an uncensored point as censored, the true value is often still closer to the KNN prediction than to the fixed 500001. As a result, KNN may show better MSE performance, despite not explicitly addressing censoring.

H Appendix: R Programming Essentials for Actuarial Data Analytics

In this appendix, summarise key R skills for the exam.

H.1 Key R Functions for Final Exam

Note this list is not exhaustive, just about the relevant pre-installed packages on the final exam.

Package: splines

`bs()` (B-spline basis) **Purpose:** Generate a B-spline basis matrix for a numeric predictor to model piecewise polynomials.

Usage:

```
bs(x, df = NULL, knots = NULL, degree = 3, intercept = FALSE)
```

Arguments:

- `x`: numeric vector of predictor values.
- `df`: target degrees of freedom; sets number of basis columns.
- `knots`: numeric vector of interior knot positions.
- `degree`: polynomial degree (default = 3 for cubic).
- `intercept`: include intercept column (logical; default FALSE).

Return Value: A design matrix with columns equal to the number of basis functions.

Example:

```
library(splines)
X_bs <- bs(wage, df = 6)
head(X_bs)
```

`ns()` (Natural spline basis) **Purpose:** Generate a natural cubic spline basis, enforcing linear tails beyond boundary knots.

Usage:

```
ns(x, df = NULL, knots = NULL, Boundary.knots = range(x))
```

Arguments:

- `x`: numeric vector of predictor values.
- `df`: degrees of freedom.
- `knots`: interior knot locations.
- `Boundary.knots`: boundary knot vector of length 2 (default = `range(x)`).

Return Value: A matrix of natural spline basis functions.

Example:

```
X_ns <- ns(wage, df = 5)
plot(wage, X_ns[,1])
```

Package: MASS

`lda()` (Linear Discriminant Analysis) **Purpose:** Fit a linear discriminant model for classification.

Usage:

```
lda(formula, data, prior = NULL, ...)
```

Arguments:

- `formula`: symbolic description, e.g., `Direction ~Lag1 + Lag2`.
- `data`: data frame containing variables.
- `prior`: numeric vector of prior class probabilities (default = sample proportions).

Return Value: An object of class `lda`, containing means, scaling, and classification rules.

Example:

```
library(MASS)
model_lda <- lda(Direction ~ ., data = Smarket, prior = c(0.5,0.5))
predict(model_lda, Smarket[1:5, ])
```

`qda()` (Quadratic Discriminant Analysis) **Purpose:** Fit a quadratic discriminant model.

Usage:

```
qda(formula, data, prior = NULL, ...)
```

Arguments: Same as in `lda()`.

Return Value: An object of class `qda` with group means and covariance matrices.

Example:

```
model_qda <- qda(Direction ~ ., data = Smarket)
predict(model_qda, Smarket[1:5, ])$class
```

Package: class

`knn()` (K-Nearest Neighbors) **Purpose:** Perform classification via k-nearest neighbors.

Usage:

```
knn(train, test, cl, k = 1, prob = FALSE)
```

Arguments:

- `train`: matrix or data frame of training predictors.
- `test`: matrix/data frame of test predictors.
- `cl`: factor of true class labels for training set.
- `k`: number of neighbors (default = 1).
- `prob`: return proportion of votes for winning class (logical).

Return Value: A factor of predicted classes (and, if `prob=TRUE`, an attribute `prob`).

Example:

```
library(class)
pred_knn <- knn(train_X, test_X, train_Y, k = 5)
```

Package: e1071

`svm()` (Support Vector Machine) **Purpose:** Train SVM models for classification and regression.

Usage:

```
svm(formula, data, kernel = "radial", cost = 1, gamma = if(is.vector(x)) 1 / ncol(x) else 1 / nrow(x), ..
```

Arguments:

- `formula`, `data`: as in `lda()`.
- `kernel`: kernel type ("linear", "radial", "polynomial", "sigmoid").
- `cost`: penalty parameter C of the error term.
- `gamma`: kernel coefficient for non-linear kernels.

Return Value: An object of class `svm` containing support vectors and coefficients.
Example:

```
library(e1071)
model_svm <- svm(Direction ~ ., data = Smarket, kernel = "linear", cost = 1)
pred_svm <- predict(model_svm, Smarket)
```

`tune()` (Model Tuning) **Purpose:** Perform grid search over hyperparameters (e.g., cost, gamma).
Usage:

```
tune(svm, train.x, train.y = NULL, data, ranges, ...)
```

Arguments:
- `svm`: the function to tune (e.g. `svm`).
- `train.x`, `train.y`: predictors and response.
- `data`: data frame if formula interface.
- `ranges`: named list of parameter ranges, e.g. `list(cost = c(0.1,1,10), gamma = c(0.5,1))`.

Return Value: An object of class `tune` with best model and performance.
Example:

```
tune_out <- tune(svm, Direction ~ ., data = Smarket,
ranges = list(cost = c(0.1,1,10)))
```

Package: boot

`boot()` (Bootstrap Sampling) **Purpose:** Generate bootstrap replicates of a statistic.
Usage:

```
boot(data, statistic, R, ...)
```

Arguments:
- `data`: vector or matrix of original data.
- `statistic`: function of form `function(data, indices)` returning statistic.
- `R`: number of bootstrap replicates.
- `...`: additional args passed to `statistic`.

Return Value: An object of class `boot` containing replicates.
Example:

```
library(boot)
boot_fn <- function(data, i) mean(data[i])
boot_obj <- boot(wage, boot_fn, R = 1000)
```

`cv.glm()` (Cross-Validation for GLMs) **Purpose:** Estimate prediction error via k-fold or leave-one-out cross-validation on a fitted generalized linear model.
Usage:

```
cv.glm(data, glmfit, K = nrow(data))
```

Arguments:
- `data`: data frame containing the variables used in the GLM.
- `glmfit`: a fitted `glm` object.
- `K`: number of folds for cross-validation (default = number of observations, i.e., LOOCV).

Return Value: A list containing components `delta`, the raw and adjusted cross-validation estimates of prediction error.

Example:

```
glm_fit <- glm(Y ~ X, data = df, family = gaussian)
cv_err <- cv.glm(df, glm_fit, K = 10)
```

H.2 Basic Data Operations

- **Reading data:** `read.table("file.txt", header=TRUE)` or `read.csv("file.csv")` to import data. If the exam provides data as a CSV, use `read.csv`. The `header` argument indicates if the first row has column names.
- **Viewing data:** `head(data)` (first 6 rows), `dim(data)` (dimensions, i.e., number of rows and columns), `names(data)` (column names), `str(data)` (structure of the data frame).
- **Subsetting:** Use `data[rows, cols]`. You can use numeric indices, names, or logical vectors:
  - `data[1:5, ]` – first 5 rows, all columns.
  - `data[, c("Age","Salary")]` – select two columns by name.
  - `data[data$Age > 30, ]` – all observations with Age $>$ 30.
  - The `subset()` function can also be used:
    `subset(data, Age>30 & Salary<=50000, select=c(Name, Salary))`.
- **Creating new variables:** `data$High <- ifelse(data$Sales > 8, "Yes","No")` creates a new factor column in `data` indicating High sales.
- **Factors vs numeric:** Use `as.factor()` to convert a numeric vector to a factor (useful for qualitative response variables in models), and `as.numeric()` to convert factors to their underlying integer codes if needed. Many modeling functions treat factors specially by creating dummy variables.

H.3 Statistical Modeling in R

- **Linear regression:** `lm(y ~ x1 + x2 + ... + xp, data=...)`. After fitting, use `summary(lm_fit)` to get coefficient estimates, standard errors, t-values, and R^2.
  Use `predict(lm_fit, newdata=..., interval="confidence")` for confidence intervals on the mean prediction, or `interval="prediction"` for prediction intervals on a new observation.
- **GLM (Logistic Regression):** `glm(y ~ x1 + x2, data=..., family=binomial)`. Use `summary(glm_fit)` to get coefficient estimates and z-values. To get predicted probabilities, use `predict(glm_fit, type="response")`. Then convert to class labels by comparing to 0.5 or another threshold: e.g., `ifelse(prob > 0.5, "Yes","No")`.
- **LDA/QDA (MASS package):** `lda()` and `qda()` from `MASS`.
  Example: `lda_fit <- lda(Direction ~ Lag1+Lag2, data=Smarket, subset=train)`. Predict with `predict(lda_fit, newdata=test_data)` which returns a list with `class` (predicted class) and `posterior` (probabilities for each class). Construct a confusion matrix with `table(predict, actual)` and get accuracy by `mean(predict==actual)`.
- **KNN (class package):** Prepare matrices of predictors and factor of labels. Example:

  ```
  library(class)
  knn_pred <- knn(train.X, test.X, train.Y, k=5)
  ```

  Then use `table(knn_pred, test.Y)` to see the results.
- **Decision Trees (tree package):** Fit by `tree_fit <- tree(Target ~ predictors, data=...)`. Plot the tree with `plot(tree_fit); text(tree_fit, pretty=0)`. `pretty=0` displays category names for splits instead of numeric codes. Prune with cost-complexity:

```
cv_tree <- cv.tree(tree_fit, FUN=prune.misclass)
prune_tree <- prune.misclass(tree_fit, best=<size>)
```

Then plot the pruned tree. Use `predict(prune_tree, newdata, type="class")` for classifications.

- **Support Vector Machine (e1071 package):**
  `svm_fit <- svm(y ~ ., data=..., kernel="radial", gamma=..., cost=..., scale=TRUE)`. `scale=TRUE` standardizes features (important if on different scales). After fitting, use `predict(svm_fit, newdata)` to get classes. Use
  `tune(svm, y ~ ., data=..., ranges=list(cost=..., gamma=...))` for cross-validation to find optimal parameters.
- **PCA (prcomp in base R):** `pr.out <- prcomp(mydata, scale.=TRUE)`. Then:
  - `pr.out$sdev`: standard deviations of PCs.
  - `pr.out$rotation`: matrix of PC loadings (each column is a principal component loading vector).
  - `pr.out$x`: the PC scores (each observation's coordinates in PC space).

  To determine how many PCs to keep, compute:

```
pr.var <- pr.out$sdev^2
pve <- pr.var / sum(pr.var)
```

  This gives proportion of variance explained by each component (PVE). You can print or plot `pve` and `cumsum(pve)`.
- **Clustering:**
  - **K-means:** `kmeans(scale(data), centers=K, nstart=20)`. `nstart` (number of random starts) is recommended to avoid local optima. The result has `$cluster` (cluster assignment for each observation) and `$centers` (cluster means).
  - **Hierarchical:** Use `dist()` to compute distance matrix, then `hclust()` on that. Example:

```
d <- dist(scale(mydata))
hc <- hclust(d, method="complete")
plot(hc)
cutree(hc, k=4)  # cut tree into 4 clusters
```

    You can try different linkage: `"single"`, `"average"`.
- **Setting a seed:** Use `set.seed(123)` for reproducibility (especially important for random splits in CV, initializations in k-means, etc.).

H.4 Model Evaluation and Utility Functions

- **Cross-validation:** Use `cv.glm` for GLMs, or manual for others. E.g., for K-fold partition, one approach:

```
folds <- sample(1:K, n, replace=TRUE)
cv_errors <- rep(0, K)
for(k in 1:K){
    fit <- lm(mpg ~ poly(horsepower, 2), data=Auto[folds!=k, ])
    pred <- predict(fit, Auto[folds==k, ])
    cv_errors[k] <- mean((Auto$mpg[folds==k] - pred)^2)
}
mean(cv_errors)
```

But high-level libraries (like `caret` or functions like `tune()`) can simplify this for certain models.

- **Confusion matrix and metrics:** After obtaining predictions and actuals, use `table(predicted, actual)`.
  Accuracy $=$
  $$\frac{\text{TP}+\text{TN}}{\text{Total}} = \texttt{mean(pred==actual)}$$
  Sensitivity $=$
  $$\frac{\text{TP}}{\text{TP}+\text{FN}}$$
  Specificity $=$
  $$\frac{\text{TN}}{\text{TN}+\text{FP}}$$
  In R, you may calculate these manually or use the `caret::confusionMatrix()` function on factors of predictions and actuals (which reports many metrics).
- **General plotting:**

H.5 R Plotting and Annotation Commands

General Plotting with `plot()`

```
plot(x, y,
    type      = "p",     % "p"=points, "l"=lines, "b"=both, "o"=overlaid,
                            % "h"=histogram-like verticals, "s"/"S"=steps
    col       = "black", % point/line color (e.g. "red", ...)
    pch       = 16,      % plotting symbol (1=empty circle, 16=solid circle)
    cex       = 1.0,     % point size multiplier
    lwd       = 1,       % line width multiplier
    lty       = 1,       % line type (1=solid, 2=dashed, 3=dotted, ...)
    xlab      = "X label",
    ylab      = "Y label",
    main      = "Title",
    sub       = "Subtitle",
    xlim      = c(xmin, xmax),
    ylim      = c(ymin, ymax),
    asp       = NA,      % aspect ratio y/x
    axes      = TRUE,    % draw axes?
    frame.plot= axes,    % draw box if axes=FALSE
    ann       = TRUE     % annotate with xlab, ylab
    )
```

`points()`, `lines()` and `abline()`

```
points(x2, y2,
    pch = 17, # symbol code
    col = "blue",
    cex = 1.5) # adds points to existing plot

lines(xg, yg,
    type = "l", # "l", "b", "p", "o", etc.
    col  = "red",
    lty  = 2, # dashed
    lwd  = 2) # adds lines or points+lines
```

```r
abline(lm_obj,
    col = "darkgreen",
    lwd = 2) # add regression line from lm object

abline(h = y0, # horizontal line at y = y0
    col = "gray", lty = 3)}

abline(v = x0, # vertical line at x = x0
    col = "gray", lty = 3)}

abline(a = a0, b = b0, # line y = a0 + b0 * x
    col = "purple", lwd = 1.5)}
```

text(), mtext() and legend()

```r
text(x, y,
    labels = row.names(df),
    pos    = 4, # 1=below,2=left,3=above,4=right
    offset = 0.5,
    col    = "darkred",
    cex    = 0.8)

mtext("Caption or source",
    side = 1, # 1=bottom, 2=left, 3=top, 4=right
    line = 4, # margin line
    adj  = 1, # 0=left,1=right
    cex  = 0.8)}

legend("topright",
    legend = c("Observed","Fit"),
    col    = c("black","red"),
    pch    = c(16,NA),
    lty    = c(NA,1),
    lwd    = c(NA,2),
    pt.cex = c(1.2,NA),
    bty    = "n", # no box
    inset  = 0.02,
    cex    = 0.9)}
```

par() for Layout and Styles

```r
par(mfrow = c(2,2)) # 2x2 plotting panels
par(mar = c(4.5,4.5,3,1)) # margins (bottom,left,top,right)
par(oma = c(1,1,1,1)) # outer margin
par(cex.lab = 1.1, # axis label size
    cex.main = 1.2, # main title size
    cex.axis = 0.9) # axis tick label size
par(las = 1) # axis labels always horizontal
```

axis() and grid()

```r
axis(side = 1,
    at     = c(0,5,10),
```

```
    labels = c("0","5","10"),
    las    = 1,
    cex.axis = 0.8)}

grid(nx = NULL, ny = NULL,
    col = "lightgray",
    lty = "dotted",
    lwd = 1) # add background grid
```

- Scatterplot: `plot(x, y)`. Use `col` to color points (can map classes to colors), `pch` to change point type.
- Add lines: `abline(lm_fit)` adds a regression line; use `lines(x_grid, predictions)` for non-linear fits.
- Multiple plots: `par(mfrow=c(2,2))` to arrange 4 plots, etc.
- Identify points: `plot(...); text(x, y, labels=row.names(data))` to label points by name or index (useful for spotting outliers on plots).