

BAN404 Statistical Learning - Exam Companion

Brede N. Espelid

Spring 2025

Contents

1	General R Setup and Tips	6
1.1	Commonly Used Libraries	6
1.2	Key R Operations	6
2	Lecture Summaries and Key Concepts (ISLR)	7
2.1	Lecture 1: Introduction to Statistical Learning	7
2.1.1	Defining Business Analytics	7
2.1.2	Categories of Business Analytics	7
2.1.3	Essential Components for Business Analytics Problems	7
2.1.4	Core Concepts in Statistical Learning	7
2.1.5	Goals of Estimating f : Prediction vs. Inference	8
2.1.6	Types of Learning	8
2.1.7	Assessing Model Accuracy	8
2.1.8	The Bias-Variance Trade-Off	8
2.1.9	The Curse of Dimensionality	9
2.1.10	Understanding Loss Functions	9
2.1.11	Lab: Introduction to R	10
2.1.12	Exercise: ISLR 2.8 (College Data)	10
2.2	Lecture 2: Linear Regression and K-Nearest Neighbors (KNN)	10
2.2.1	Revisiting Overfitting with Polynomial Regression	10
2.2.2	Simple Linear Regression	11
2.2.3	Assessing Accuracy of Coefficient Estimates	11
2.2.4	Assessing Overall Model Accuracy	12
2.2.5	Multiple Linear Regression	13
2.2.6	Prediction and Confidence/Prediction Intervals	13
2.2.7	Model Extensions and Potential Problems	14
2.2.8	K-Nearest Neighbors (KNN) Regression	15
2.2.9	ISLR Chapter 3 Lab Highlights	16
2.2.10	Key Insights from ISLR Chapter 3 Exercises	17
2.3	Lecture 3: Classification - Logistic Regression	18
2.3.1	Introduction to Classification	18
2.3.2	Measuring Prediction Quality in Classification	18
2.3.3	Why Not Linear Regression for Classification?	18
2.3.4	Logistic Regression Model	19
2.3.5	Estimating Coefficients: Maximum Likelihood Estimation (MLE)	19
2.3.6	Making Predictions and Evaluating Accuracy	20
2.3.7	Multiple Logistic Regression	21

2.3.8	Logistic Regression for ≥ 2 Response Classes (Multinomial Logistic Regression)	21
2.3.9	Exercise: ISLR 4.13a-d (or old 4.10a-d)	21
2.4	Lecture 4: Classification - Discriminant Analysis & KNN	22
2.4.1	Linear Discriminant Analysis (LDA)	22
2.4.2	Quadratic Discriminant Analysis (QDA)	24
2.4.3	Comparison of Classification Methods (Logistic, LDA, QDA, KNN)	24
2.4.4	Evaluating Classification Models (Revisited)	25
2.4.5	K-Nearest Neighbors (KNN) Classifier	26
2.4.6	Exercises: ISLR 4.13e-i (or old 4.10e-i)	27
2.5	Lecture 5: Resampling Methods - Cross-Validation	27
2.5.1	Introduction to Resampling Methods	27
2.5.2	Cross-Validation (CV)	28
2.5.3	The Validation Set Approach	28
2.5.4	Leave-One-Out Cross-Validation (LOOCV)	29
2.5.5	k-Fold Cross-Validation	30
2.5.6	Pros and Cons of Different CV Approaches	31
2.5.7	The Right and Wrong Way to do Cross-Validation	32
2.5.8	Exercise: ISLR 5.5 (Cross-validation for Logistic Regression)	32
2.6	Lecture 6: Resampling Methods - The Bootstrap	33
2.6.1	Introduction to the Bootstrap	33
2.6.2	The Bootstrap Procedure for Estimating Standard Error	33
2.6.3	Example: Estimating SE of Investment Allocation	33
2.6.4	R Implementation of Bootstrap	34
2.6.5	Bootstrap Confidence Intervals	35
2.6.6	When is Bootstrap Useful?	36
2.6.7	Limitations of Bootstrap	36
2.6.8	Conceptual Exercise Insights	36
2.6.9	Exercise: ISLR 5.9 (Bootstrap for Boston data)	37
2.7	Lecture 7: Linear Model Selection Methods	38
2.7.1	Introduction to Model Selection	38
2.7.2	Three Types of Model Selection Techniques	38
2.7.3	Best Subset Selection	39
2.7.4	Stepwise Selection Methods	40
2.7.5	Choosing the Optimal Model (Criteria)	41
2.7.6	Exercise: ISLR 6.8	42
2.8	Lecture 8: Regularization (Shrinkage) Methods and PCR	43
2.8.1	Introduction to Shrinkage Methods	43
2.8.2	Ridge Regression	43
2.8.3	The Lasso (Least Absolute Shrinkage and Selection Operator)	45
2.8.4	Comparing Lasso and Ridge	46
2.8.5	Selecting the Tuning Parameter λ	46
2.8.6	Dimension Reduction Methods	46
2.8.7	Principal Components Analysis (PCA) and Regression (PCR)	46
2.8.8	Partial Least Squares (PLS)	48
2.8.9	Exercise: ISLR 6.11	48
2.9	Lecture 9: Non-linear Models - Polynomials, Splines, Local Regression	49
2.9.1	Introduction to Non-linear Models	49
2.9.2	Polynomial Regression	49
2.9.3	Step Functions (Piecewise Constant Regression)	49

2.9.4	Basis Functions	50
2.9.5	Regression Splines (Piecewise Polynomials)	50
2.9.6	Smoothing Splines	51
2.9.7	Local Regression (LOESS/LOWESS)	52
2.9.8	Kernel Estimators (Nadaraya-Watson)	53
2.9.9	Exercises: ISLR 7.6, 7.9, (7.10, 7.11 for GAMs)	54
2.10	Lecture 10: Generalized Additive Models (GAMs)	54
2.10.1	Introduction to GAMs	54
2.10.2	Fitting GAMs: Backfitting Algorithm	55
2.10.3	GAMs for Regression	55
2.10.4	GAMs for Classification	55
2.10.5	Pros and Cons of GAMs	56
2.10.6	Exercises: ISLR 7.10, 7.11	56
2.11	Lecture 11: Tree-Based Methods	57
2.11.1	Introduction to Decision Trees	57
2.11.2	Building Regression Trees	57
2.11.3	Tree Pruning	58
2.11.4	Classification Trees	59
2.11.5	Pros and Cons of Decision Trees	60
2.11.6	Exercise: ISLR 8.8a-c	61
2.12	Lecture 12: Bagging, Random Forests, Boosting	62
2.12.1	Bagging (Bootstrap Aggregation)	62
2.12.2	Random Forests	63
2.12.3	Variable Importance Measures (for Bagging/RF)	63
2.12.4	Boosting	64
2.12.5	Exercises: ISLR 8.8d-e, 8.11	65
2.13	Lecture 13: Support Vector Machines (SVMs)	65
2.13.1	Introduction to Support Vector Machines	65
2.13.2	Hyperplanes	66
2.13.3	Classification by Separating Hyperplanes	66
2.13.4	The Maximal Margin Classifier (MMC)	66
2.13.5	Support Vector Classifier (SVC) / Soft Margin Classifier	67
2.13.6	Support Vector Machines (SVMs) - Non-linear Boundaries	67
2.13.7	R Implementation of SVMs	68
2.13.8	SVMs with More than Two Classes	69
2.13.9	Relationship to Logistic Regression	69
2.13.10	Exercise: ISLR 9.7	69
2.14	Lecture 14: Unsupervised Learning	70
2.14.1	Introduction to Unsupervised Learning	70
2.14.2	Principal Component Analysis (PCA)	70
2.14.3	Clustering Methods	72
2.14.4	K-Means Clustering	72
2.14.5	Hierarchical Clustering	74
2.14.6	Practical Issues in Clustering	75
2.14.7	Exercises: ISLR 12.8, 12.9 (.	75
3	Previous Exams	75
3.1	Exam Spring 2021	75
3.1.1	1a: Bootstrap histogram for volatility	75
3.1.2	1b: 95% CI for volatility (normal assumption)	76

3.1.3	1c: 95% CI for volatility (percentile method)	76
3.1.4	1d: Regression of Squared Returns	77
3.1.5	1e: Bootstrap SE of regression coefficient	77
3.1.6	2a: Data Filtering and Variable Removal	78
3.1.7	2b: Descriptive statistics and variable formatting	78
3.1.8	2c: Cross-validation for linear models	79
3.1.9	2d: Generalized Additive Model (GAM)	80
3.1.10	2e: Specify appropriate GAM and compute testMSE	81
3.1.11	3a: Data prep, descriptive stats for <code>clm</code>	82
3.1.12	3b: Logistic regression for <code>clm</code>	83
3.1.13	3c: Validation set for logistic regression, thresholding	83
3.1.14	3d: Boosted trees for <code>clm</code>	84
3.2	Exam Spring 2022	85
3.2.1	1a: Data preparation and splitting	85
3.2.2	1b: Descriptive statistics for promising predictors	86
3.2.3	1c: Logistic regression	87
3.2.4	1d: Classification tree	88
3.2.5	1e: Cross-validation for optimal threshold for tree	88
3.2.6	1f: Analysis with <code>LoyalCH</code>	89
3.2.7	2a: Data preparation and splitting	90
3.2.8	2b: Descriptive statistics for promising predictors	91
3.2.9	2c: Linear regression for <code>price</code>	91
3.2.10	2d: Linear regression of <code>log(price)</code>	92
3.2.11	2e: GAM for <code>price</code>	93
3.2.12	2f: Explanation of backfitting for GAMs	93
3.2.13	2g: Bagged trees for <code>price</code>	94
3.2.14	2h: Explanation of bagging	95
3.3	Exam Spring 2023	95
3.3.1	1a: Data preparation and splitting	95
3.3.2	1b: Descriptive methods for <code>bill_avg</code>	96
3.3.3	1c: Best OLS for <code>bill_avg</code>	97
3.3.4	1d: LASSO regression for <code>bill_avg</code>	98
3.3.5	1e: Evaluate LASSO model	99
3.3.6	1f: Regression tree for <code>bill_avg</code>	99
3.3.7	1g: Evaluate regression tree	100
3.3.8	1h: Random forest for <code>bill_avg</code>	100
3.3.9	1i: Evaluate random forest	101
3.3.10	1j: Features of customers with high/low <code>bill_avg</code>	101
3.3.11	2a: Data formatting, descriptive statistics for <code>churn</code>	101
3.3.12	2b: Bootstrap CI for $P(\text{churn}=1)$	102
3.3.13	2c: Logistic regression for <code>churn</code>	103
3.3.14	2d: Evaluate logistic regression, consider removing variables	104
3.3.15	2e: Random forest for <code>churn</code>	105
3.3.16	2f: Typical features of customers who churn	105
3.4	Exam Spring 2024	106
3.4.1	1a: Explain R-function	106
3.4.2	1b: Optimal K for function f using LOOCV	106
3.4.3	1c: Plot predictions with optimal K	107
3.4.4	1d: Modify function f for multiple predictors	108
3.4.5	1e: Implement backfitting for this method	109

3.4.6	2a: Recode categorical variables and split data	110
3.4.7	2b: Motivation for data usage (Inference: "Why dissatisfied?")	111
3.4.8	2c: Descriptive statistics for	111
3.5	2d: Logistic Regression (Why dissatisfied?)	112
3.6	2e: Classification Tree (Why dissatisfied?)	113
3.7	2f: Random Forest (Why dissatisfied?)	113
3.8	2g: Why are some customers dissatisfied?	113
3.9	2h: Assumptions for Prediction Task (Pre-flight variables)	113
3.10	2i: Motivation for Train/Test Split in Prediction	113
3.11	2j: Logistic Regression (Prediction)	114
3.12	2k: Random Forest (Prediction)	114
3.13	2l: Future Data Collection	114
4	Tutorials	114
4.1	Tutorial 1	114
4.1.1	Task 1: Linear Regression and KNN (One Predictor)	114
4.1.2	Task 2: Data Splitting and Basic Linear Model	116
4.1.3	Task 3: Exercise 3.10a-f from ISLR Book	117
4.2	Tutorial 2	117
4.2.1	Task 1: Logistic Regression and KNN for Binary Classification	117
4.2.2	Task 3: Logistic Regression vs LDA (Default data)	119
4.2.3	Task 4 : Cross-Validation Methods (Auto data)	120
4.3	Tutorial 3	120
4.3.1	Task 1: Shrinkage Methods (College data)	120
4.3.2	Task 2: Non-linear Models (Small Dataset)	122
4.3.3	Task 3: Extension of KNN (Nadaraya-Watson)	123
4.4	Tutorial 4	123
4.4.1	Task 1 : Regression Trees (Hitters data)	123
4.4.2	Task 2: Bagging and Random Forest (Hitters data)	124
4.4.3	Task 3: Boosting (Hitters data)	125
4.4.4	Task 4: Support Vector Machines (Simulated Data 'xy.csv')	125
4.4.5	Task 5: PCA (advertising.csv)	126
4.4.6	Task 6: Cluster Analysis (USArrests data)	126
5	Assignment: Compulsory Assignment BAN404 (Spring 2025x)	126
5.1	Initial Data Loading and Preparation	127
5.2	Task 1: Descriptive Statistics and Predictor Investigation	127
5.3	Task 2: Linear Regression and LASSO	129
5.4	Task 3: KNN Regression	130
5.5	Task 4: KNN with Multiple Predictors and GAM	131

1 General R Setup and Tips

1.1 Commonly Used Libraries

- ISLR or ISLR2: For datasets from the book.
- `boot`: For bootstrap functions (`boot()`, `cv.glm()`).
- `glmnet`: For Ridge and LASSO (`glmnet()`, `cv.glmnet()`).
- `tree`: For classification and regression trees (`tree()`).
- `randomForest`: For Random Forests (`randomForest()`, `varImpPlot()`).
- `gbm`: For Gradient Boosting Machines (`gbm()`).
- `gam`: For Generalized Additive Models (`gam()`, `s()`, `lo()`).
- `e1071`: For Support Vector Machines (`svm()`).
- `MASS`: For LDA, QDA (`lda()`, `qda()`).
- `tidyverse` (or individual packages like `dplyr`, `ggplot2`): For data manipulation and plotting.
- `fastDummies`: For creating dummy variables (`dummy_cols()`).
- `insuranceData`: For `dataCar` dataset (Exam 2021).
- `Ecdat`: For `Computers` dataset (Exam 2022).

1.2 Key R Operations

- Setting seed: `set.seed(number)` for reproducibility.
- Splitting data (50/50 example):

```
1 n <- nrow(mydata)
2 set.seed(123)
3 train_indices <- sample(1:n, floor(n/2))
4 train_data <- mydata[train_indices, ]
5 test_data <- mydata[-train_indices, ]
```

- Converting to factors: `mydata$variable <- as.factor(mydata$variable)`
- Model fitting:
 - Linear Regression: `lm(y ~ x1 + x2, data=train_data)` *LogisticRegression* :
- Evaluation:
 - MSE: `mean((predictions - actuals)^2)`
 - Confusion Matrix: `table(predicted_class, actual_class)` *Accuracy* :
- LASSO/Ridge: Remember to create a model matrix (e.g., `x <- model.matrix(y ~ -1, data=train_data)`) and a response vector

2 Lecture Summaries and Key Concepts (ISLR)

2.1 Lecture 1: Introduction to Statistical Learning

2.1.1 Defining Business Analytics

- *Analytics* (Davenport & Harris, 2007): "the extensive use of data, statistical and quantitative analysis, explanatory and predictive models and fact based management to drive decisions and actions."
- *Business Analytics*: Applying analytics in the context of firms making the best possible decisions efficiently using available data.

2.1.2 Categories of Business Analytics

- **Descriptive Analytics**: Understanding past and present through data exploration (e.g., salary structures). Not model-based, not prescriptive.
- **Predictive Analytics**: Forecasting future events or unknown properties (e.g., stock prices, probability of tax evasion). This course primarily focuses here.
- **Prescriptive Analytics**: Recommending actions based on optimization or statistical models (e.g., resource allocation, best medication choice).

2.1.3 Essential Components for Business Analytics Problems

- Management Science / Operations Research: Formulating decision problems from business questions.
- Data Analysis / Statistics: Choosing data and methods to estimate/predict quantities of interest.
- Programming / Database Management: Implementing solutions.

2.1.4 Core Concepts in Statistical Learning

- Definition: A vast set of tools for understanding data.
- Example: Improving sales based on advertising budgets (TV, Radio, Newspaper).
 - Inputs (Predictors, Independent Variables, Features): X_1, X_2, \dots, X_p .
 - Output (Response, Dependent Variable): Y .
- The Data Generating Process (DGP): $Y = f(X) + \epsilon$
 - f : Systematic information that X provides about Y . Unknown.
 - ϵ : Random error. Zero mean, independent of X . Represents irreducible error.

2.1.5 Goals of Estimating f : Prediction vs. Inference

- **Prediction Focus:**

- Goal: Estimate Y using $\hat{Y} = \hat{f}(X)$.
- Accuracy depends on reducible and irreducible error.
- *Reducible error*: Error from \hat{f} not being a perfect estimate of f . Can be reduced with better models.
- *Irreducible error*: Due to ϵ . Inherent variability/noise.

- **Inference Focus:**

- Goal: Understand the relationship between X and Y .
- Key questions: Which predictors are associated? Strength of association? Functional form of f ?

2.1.6 Types of Learning

- **Supervised Learning:** Both predictors X_i and response Y_i are observed.

- Regression Problems: Y is quantitative (e.g., price, blood pressure).
- Classification Problems: Y is qualitative/categorical (e.g., disease/no disease, purchase/no purchase).

- **Unsupervised Learning:** Only predictors X_i are observed.

- Goal: Discover structure (e.g., clustering, PCA).

2.1.7 Assessing Model Accuracy

- **Measuring Quality of Fit - Regression:**

- *Mean Squared Error (MSE)*: $MSE_{Train} = \text{avg}(y_i - \hat{f}(x_i))^2$.
- Goal is to minimize $MSE_{Test} = E(Y_0 - \hat{f}(x_0))^2$ for new unseen data (x_0, Y_0) .

- **Measuring Quality of Fit - Classification:**

- *Training Error Rate*: $\frac{1}{n} \sum_{i=1}^n I(y_i \neq \hat{y}_i)$.
- *Test Error Rate*: $\text{Avg}(I(y_0 \neq \hat{y}_0))$ for new unseen data.
- *Bayes Classifier*: A theoretical ideal that minimizes test error rate. Assigns to class k for which $P(Y = k|X = x_0)$ is largest.
- *K-Nearest Neighbors (KNN) Classifier*: Practical method, estimates $P(Y = k|X = x_0)$ based on fraction of K nearest neighbors belonging to class k .

2.1.8 The Bias-Variance Trade-Off

- For a given x_0 , the expected test MSE: $E(y_0 - \hat{f}(x_0))^2 = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon)$.
- **Bias**: Error from simplifying a complex real-world problem. More flexible models typically have lower bias.

- **Variance:** How much \hat{f} would change with different training data. More flexible models typically have higher variance.
- **Overfitting vs. Underfitting:**
 - *Underfitting:* High bias, model too simple (e.g., linear model for non-linear data). Low training and test performance typically.
 - *Overfitting:* Low bias (on training data), high variance. Model fits training noise. Low training error, high test error. (Slides p. 17-22 illustrate with polynomial regression on limited training data, showing poor generalization to test data).

2.1.9 The Curse of Dimensionality

- Phenomenon: Test error tends to increase as dimensionality (p) grows, unless the additional predictors are truly associated with the response.
- Reason: In high dimensions, data becomes sparse. Nearest neighbors can be very far away, making local methods (like KNN) less effective.
- Volume of space increases exponentially: s^p . To maintain density, n must grow exponentially.
- Most points in a high-dimensional hypercube are close to the boundary. (R code ‘cube.R’, ‘hypercube.R’, ‘edges.R’).

Listing 1: R code: Fraction of points near boundary (edges.R concept)

```

1 GetRatio <- function(dim){
2   1 - (0.9)^dim # (1 - 20.05)^dim is volume of inner hypercube
3 }
4 p_vals <- 1:10 # For demonstration, ISLR uses up to p=10 for nearest
   neighbor distance plots
5 ratios <- sapply(p_vals, GetRatio)
6 plot(p_vals, ratios, type="b", xlab="Number of Dimensions (p)",
7      ylab="Fraction of Volume Near Boundary (within 10% of range)",
8      main="Curse of Dimensionality: Volume Concentration")
9 abline(h=0.5, col="red", lty=2) # Shows when over 50% of volume is "near
   " boundary
10 text(5, 0.4, "p=5, ~41% near boundary", pos=4)
11 text(10, 0.6, "p=10, ~65% near boundary", pos=4)

```

2.1.10 Understanding Loss Functions

- Squared Error Loss: $L(Y, \hat{f}(X)) = (Y - \hat{f}(X))^2$. Minimizing expected squared error leads to prediction $\hat{f}(x) = E(Y|X = x)$.
 - Unconditional case (predict with constant θ): $\theta_{opt} = E(Y)$. Sample estimate \bar{y} .
- Absolute Error Loss: $L(Y, \hat{f}(X)) = |Y - \hat{f}(X)|$. Minimizing expected absolute error leads to prediction $\hat{f}(x) = \text{median}(Y|X = x)$.
 - Unconditional case: $\theta_{opt} = \text{median}(Y)$. Sample estimate is sample median.
- Implication: The "best" prediction can change depending on how we define "loss" or "error."

2.1.11 Lab: Introduction to R

- Basic commands, graphics, data indexing, loading data, summaries.
- Relevant files: ‘ch2-lab.R’ (from GitHub), ‘ex2.8.Rmd’ (Canvas).

2.1.12 Exercise: ISLR 2.8 (College Data)

- Application of descriptive statistics and basic plotting in R.
- See ‘ex2.8.Rmd’ (Canvas) or ‘ch2-applied.R’ (GitHub) for solution ideas.

2.2 Lecture 2: Linear Regression and K-Nearest Neighbors (KNN)

2.2.1 Revisiting Overfitting with Polynomial Regression

- **Context:** Example using a small training dataset (10 observations, x, y) to fit polynomial models of increasing degrees ($k = 1, 2, 3, 4$). The true underlying relationship is quadratic ($k = 2$).
- **Training MSE Behavior** (Slides L2 p. 5):
 - * Linear model ($k = 1$): `trMSE1 = 14.37876`
 - * Quadratic model ($k = 2$): `trMSE2 = 0.4588453`
 - * Cubic model ($k = 3$): `trMSE3 = 0.4320397`
 - * Quartic model ($k = 4$): `trMSE4 = 0.4003628`
 - * *Observation:* Training MSE decreases as model flexibility (polynomial degree) increases. The 4th order polynomial, being most flexible, has the lowest training MSE. This is typical: more flexible models can better fit the noise in the training data.

Listing 2: R code for Training MSE (Conceptual - from Slides L2 p.5 values)

```
1 # Assuming y_train are the true training y values
2 # and reg1, reg2, reg3, reg4 are lm objects for k=1,2,3,4
3 # fitted ONLY on the first 10 training observations.
4 # y_pred_reg1 <- predict(reg1, newdata=training_data_first_10)
5 # trMSE1 <- mean((y_train_first_10 - y_pred_reg1)^2)
6 # ... similar for reg2, reg3, reg4
```

- **Test MSE Behavior** (Slides L2 p. 6-8):
 - * The full dataset actually has 20 observations. The first 10 were training, the next 10 are used as a hold-out test set.
 - * Linear model ($k = 1$): `teMSE1 = 3522.564` (Significant underfitting)
 - * Quadratic model ($k = 2$): `teMSE2 = 0.9843595` (Best test performance, matches true DGP)
 - * Cubic model ($k = 3$): `teMSE3 = 173.0981` (Overfitting starts)
 - * Quartic model ($k = 4$): `teMSE4 = 8582.551` (Severe overfitting)
- **Visualizing Over/Underfitting** (Slides L2 p. 7): The plot shows the linear model (black) clearly missing the curve. The quadratic model (blue) tracks the full dataset well. The 3rd/4th order polynomial (green, if shown for $k=4$ on slide 7, or specifically for $k=4$ on slide 22 of L1 slides) fits the initial 10 points well but deviates wildly for the test points.

- **Conclusion** (Slide L2 p. 9): Models that are too flexible (e.g., high-degree polynomials here) can achieve low training error by fitting noise, but generalize poorly to unseen test data (high test error). The goal is to find a model complexity that balances fitting the signal without fitting the noise, typically leading to the lowest test error.

2.2.2 Simple Linear Regression

- **Model:** $Y = \beta_0 + \beta_1 X + \epsilon$. Y is response, X is predictor, β_0 is intercept, β_1 is slope, ϵ is error term.
- **Estimating Coefficients using Least Squares:**
 - * Objective: Minimize the Residual Sum of Squares (RSS).

$$RSS = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2 = \sum_{i=1}^n (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2$$

(Slide L2 p. 28 shows the minimization objective $\min_{\alpha, \beta} g(\alpha, \beta)$).

- * Least Squares Estimates:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} = \frac{S_{xy}}{S_{xx}}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

- * R Implementation (ISLR Auto data, predict mpg from horsepower):

Listing 3: Simple Linear Regression in R (ISLR Ch 3 Lab, Slides L2 p. 29, L2 p.36)

```

1 # Using Auto dataset from ISLR package
2 library(ISLR)
3 attach(Auto) # or use Auto$horsepower, Auto$mpg
4 lm.fit <- lm(mpg ~ horsepower) # shorthand for lm(mpg ~
   horsepower, data=Auto)
5 summary(lm.fit)
6 # Output from summary(lm.fit) (similar to Slide L2 p.36 for TV~
   sales):
7 # Coefficients:
8 #             Estimate Std. Error t value Pr(>|t|)
9 # (Intercept) 39.935861   0.717499   55.66   <2e-16
10 # horsepower  -0.157845   0.006446  -24.49   <2e-16
11
12 plot(horsepower, mpg)
13 abline(lm.fit, col="red", lwd=3)
14 # To manually calculate coefficients (Slide L2 p.37 concept for
   X, y):
15 # X_design <- cbind(1, horsepower) # Design matrix
16 # y_response <- mpg
17 # beta_hat_manual <- solve(t(X_design) %% X_design) %% t(X_
   design) %% y_response
18 # print(beta_hat_manual) # Matches lm.fit coefficients
19 detach(Auto)

```

2.2.3 Assessing Accuracy of Coefficient Estimates

- The true relationship is $Y = \beta_0 + \beta_1 X + \epsilon$. $\hat{\beta}_0, \hat{\beta}_1$ are estimates.

- **Standard Error (SE)** of estimates: Measures sampling variability. If we refit the model on different random samples, SE tells us how much $\hat{\beta}_j$ would typically vary.

- * $SE(\hat{\beta}_0)^2 = \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right]$
- * $SE(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2}$, where $\sigma^2 = \text{Var}(\epsilon)$.
- * σ^2 is usually unknown and estimated by $RSE^2 = RSS/(n-2)$ for simple linear regression.
- * Distribution of $\hat{\beta}_1$ (Slide L2 p.30, assuming $\epsilon_i \sim N(0, \sigma^2)$ and are independent):

$$\hat{\beta}_1 \sim N\left(\beta_1, \frac{\sigma^2}{\sum (x_i - \bar{x})^2}\right)$$

- **Confidence Intervals (CI)**: A $(1 - \alpha)\%$ CI for β_j is a range that contains the true β_j with $(1 - \alpha)\%$ probability in repeated sampling.

- * For β_1 : $\hat{\beta}_1 \pm t_{\alpha/2, n-2} \cdot SE(\hat{\beta}_1)$. (Slide L2 p.31 uses 1.96 from z -dist for large n).
- * Example from `summary(lm.fit)()`: The `confint(lm.fit)()` function gives CIs.

- **Hypothesis Testing** for β_1 :

- * $H_0 : \beta_1 = 0$ (No linear relationship between X and Y).
- * $H_1 : \beta_1 \neq 0$ (There is a linear relationship).
- * Test statistic: $t = \frac{\hat{\beta}_1 - 0}{SE(\hat{\beta}_1)}$. This t -statistic follows a t -distribution with $n - 2$ degrees of freedom under H_0 .
- * p-value: The probability of observing a $|t|$ -statistic as large or larger than the one computed, assuming H_0 is true. A small p-value (e.g., ≤ 0.05) provides evidence against H_0 .
- * `summary(lm.fit)()` output provides t -values and p-values ('Pr(>|t|)') for each coefficient.

2.2.4 Assessing Overall Model Accuracy

- **Residual Standard Error (RSE)**:

$$RSE = \hat{\sigma} = \sqrt{\frac{RSS}{n - p - 1}}$$

(For simple LR, $p = 1$, so $n - 2$ in denominator. Slide L2 p. 38).

- * An estimate of $\sigma = \text{std.dev.}(\epsilon)$.
- * Represents the average amount that the response will deviate from the true regression line.
- * Measured in units of Y .

- **R-squared (R^2)**:

$$R^2 = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}$$

where $TSS = \sum (y_i - \bar{y})^2$ is Total Sum of Squares. (Slide L2 p. 38)

- * Proportion of variance in Y that is explained by $X(s)$.
- * $0 \leq R^2 \leq 1$. Closer to 1 indicates better fit.

- * For simple linear regression, $R^2 = (\text{Cor}(X, Y))^2$. (ISLR Ch3 Exercise 3.7 provides a proof for this, which involves algebraic manipulation of RSS, TSS and the formula for correlation).
- * R^2 always increases or stays the same when more predictors are added, even if they are irrelevant. This makes it less suitable for comparing models with different numbers of predictors.

2.2.5 Multiple Linear Regression

- **Model:** $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p + \epsilon$.
- **Estimating Coefficients:**
 - * Minimize RSS: $\sum_{i=1}^n (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \dots + \hat{\beta}_p x_{ip}))^2$.
 - * **Matrix Notation** (Slides L2 p. 33-35):
 - Response vector \mathbf{y} ($n \times 1$).
 - Design Matrix \mathbf{X} ($n \times (p+1)$), first column usually all 1s for intercept.
 - Coefficient vector $\boldsymbol{\beta}$ ($(p+1) \times 1$).
 - Model: $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$.
 - RSS: $g(\boldsymbol{\beta}) = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$.
 - OLS Estimator: $\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$.
 - Variance-Covariance Matrix of $\hat{\boldsymbol{\beta}}$: $\text{Var}(\hat{\boldsymbol{\beta}}) = \sigma^2(\mathbf{X}'\mathbf{X})^{-1}$. Diagonal elements are $SE(\hat{\beta}_j)^2$.
- **Hypothesis Testing in Multiple Regression:**
 - * **For individual coefficients β_j :** $H_0 : \beta_j = 0$ (predictor X_j has no effect, holding others constant).
 - t -statistic: $t_j = \frac{\hat{\beta}_j - 0}{SE(\hat{\beta}_j)}$. Follows t_{n-p-1} under H_0 .
 - Provided in `summary(lm.fit)()`.
 - * **Overall F-test (Model Significance):** $H_0 : \beta_1 = \beta_2 = \dots = \beta_p = 0$ (none of the predictors explain Y).
 - $F = \frac{(TSS - RSS)/p}{RSS/(n-p-1)} = \frac{MSR}{MSE}$.
 - Under H_0 , $F \sim F_{p, n-p-1}$.
 - If F is large (small p-value), reject H_0 , meaning at least one predictor is useful.
 - Provided in `summary(lm.fit)()`.
- **R Implementation** (Example from ISLR Ch3 Lab):

Listing 4: Multiple Linear Regression for Boston Housing Data

```

1 library(MASS) # For Boston dataset
2 lm.fit.multi <- lm(medv ~ lstat + age, data=Boston)
3 summary(lm.fit.multi)
4 lm.fit.all <- lm(medv ~ ., data=Boston) # Using all predictors
5 summary(lm.fit.all)

```

2.2.6 Prediction and Confidence/Prediction Intervals

- Prediction for a new set of predictor values \mathbf{x}_0 : $\hat{y}_0 = \mathbf{x}_0' \hat{\boldsymbol{\beta}}$.

- **Confidence Interval for Mean Response** $E(Y|X = \mathbf{x}_0)$:
 - * Quantifies uncertainty about the average value of Y for a specific \mathbf{x}_0 .
 - * $\hat{y}_0 \pm t_{\alpha/2, n-p-1} \cdot SE(\hat{E}(Y|X = \mathbf{x}_0))$.
 - * For simple LR (Slide L2 p.32): $\hat{Y} \pm 1.96\hat{\sigma}\sqrt{\frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum(x_i - \bar{x})^2}}$. The 1.96 is $z_{0.025}$.
 - * In R: `predict(lm.fit, newdata=..., interval="confidence")()`.
- **Prediction Interval for Individual Response** Y_0 at $X = \mathbf{x}_0$:
 - * Quantifies uncertainty about a single future observation Y_0 for a specific \mathbf{x}_0 .
 - * Accounts for both uncertainty in $\hat{\beta}$ and irreducible error ϵ . Always wider than CI.
 - * $\hat{y}_0 \pm t_{\alpha/2, n-p-1} \cdot \sqrt{RSE^2 + SE(\hat{E}(Y|X = \mathbf{x}_0))^2}$.
 - * For simple LR (Slide L2 p.32): $\hat{Y} \pm 1.96\hat{\sigma}\sqrt{1 + \frac{1}{n} + \frac{(x_0 - \bar{x})^2}{\sum(x_i - \bar{x})^2}}$.
 - * In R: `predict(lm.fit, newdata=..., interval="prediction")()`.

Listing 5: Confidence and Prediction Intervals in R

```

1 # Using simple linear model lm.fit <- lm(mpg ~ horsepower, data=Auto
  )
2 new_hp_values <- data.frame(horsepower=c(98, 150, 200))
3 # Confidence intervals for average mpg
4 predict(lm.fit, newdata=new_hp_values, interval="confidence")
5 # Prediction intervals for individual car's mpg
6 predict(lm.fit, newdata=new_hp_values, interval="prediction")

```

2.2.7 Model Extensions and Potential Problems

- **Qualitative Predictors:**
 - * Create $k - 1$ dummy variables for a k -level categorical predictor. One level is baseline.
 - * Coefficients are interpreted relative to the baseline.
 - * R's `lm()` handles factors automatically. `contrasts()` shows encoding.
- **Interaction Terms:** $X_1 \cdot X_2$. Allows effect of X_1 on Y to depend on the level of X_2 .
 - * Syntax in R: `Y ~ X1 + X2 + X1:X2` or `Y ~ X1X2`.
 - * Hierarchical Principle: If an interaction term $X_1 : X_2$ is included, the main effects X_1 and X_2 should also be included, even if their p-values are not significant.
- **Non-linear Relationships:**
 - * **Polynomial Regression:** Include X^2, X^3, \dots as predictors. In R: `lm(Y ~ X + I(X^2))`, or `lm(Y ~ poly(X, degree=2))`. `anova(lm.fit1, lm.fit2())` can compare nested models.
 - * **Potential Problems in Linear Regression:**
 - Non-linearity of Data:* If true relationship is non-linear, linear model is a poor fit. Detect with residual plots (residuals vs. fitted values, or residuals vs. predictors). Solution: transformations (log, sqrt), polynomial terms, or non-linear models.

Correlation of Error Terms (ϵ_i): Standard errors will be underestimated, CIs too narrow, p-values too small. Occurs often with time series data. Detect with Durbin-Watson test, plotting residuals vs. time, or ACF of residuals.

Non-constant Variance of Errors (Heteroscedasticity): Variance of ϵ_i depends on X_i . Funnel shape in residual vs. fitted plot. Invalidates SEs, CIs, hypothesis tests. Solution: transform Y (e.g., $\log Y$, \sqrt{Y}), use weighted least squares.

Outliers: Observations with y_i far from model prediction. Large residuals. Can unduly influence model. Detect with studentized residuals (residuals divided by estimated SE; values $\hat{\epsilon}_i$ —3— are suspect). Solution: remove if data entry error, otherwise be cautious.

High-Leverage Points: Observations with unusual x_i values. Leverage statistic h_{ii} (diagonal of hat matrix $\mathbf{X}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'$). h_{ii} between $1/n$ and 1. Average $h_{ii} = (p+1)/n$. Points with h_{ii} much larger than average have high leverage. Can have large impact on $\hat{\beta}$.

Collinearity/Multicollinearity: Two or more predictors are highly correlated. Difficult to separate individual effects. SEs of $\hat{\beta}_j$ become large, t-stats small. Overall F-test might be significant while individual p-values are not. Detect with correlation matrix of predictors, Variance Inflation Factor (VIF). $VIF_j = 1/(1 - R_{X_j|X_{-j}}^2)$. VIF ≥ 5 or 10 indicates problem. Solution: drop one correlated variable, combine variables (e.g., PCA), use ridge regression.

Listing 6: Checking VIF in R

```
1 # Assuming lm.fit.all <- lm(medv ~ ., data=Boston)
2 library(car) # for vif function
3 vif_values <- vif(lm.fit.all)
4 print(vif_values)
5 # Example: tax and rad often have high VIF in Boston dataset
```

2.2.8 K-Nearest Neighbors (KNN) Regression

- **Concept:** A non-parametric method that predicts Y for a given x_0 by averaging the Y values of its K "closest" neighbors in the training data.
- **Algorithm for prediction at x_0** (Slide L2 p.14):
 1. Identify the K training observations (x_i, y_i) that are closest to x_0 . This set of neighbors is N_0 .
 2. The KNN regression fit for x_0 is $\hat{f}(x_0) = \frac{1}{K} \sum_{x_i \in N_0} y_i$.
- **Distance Metric:**
 - * For a single predictor x : $d(x, x_0) = |x - x_0|$.
 - * For multiple predictors \mathbf{x} : Euclidean distance $\sqrt{\sum_{j=1}^p (x_j - x_{0j})^2}$. It's crucial to scale predictors to similar ranges if they are on different scales, otherwise predictors with larger scales will dominate the distance calculation.
- **Choice of K (The Tuning Parameter):**
 - * $K = 1$ (Slide L2 p.17): Lowest bias, highest variance. The prediction function is very flexible and interpolates the training data. Can be very noisy.
 - * Large K (e.g., $K = 20$ on Slide L2 p.18, or $K = n$): Higher bias, lower variance. The prediction function becomes smoother. If $K = n$, it predicts the global mean \bar{y} for all x_0 .

- * Optimal K : Balances bias and variance to minimize Test MSE. Typically chosen using cross-validation.
 - * Slides L2 p.21-24 illustrate this by plotting Squared Bias, Variance, and Test MSE vs. K for a simulated dataset, showing a U-shaped Test MSE curve with an optimal $K \approx 36$.
- **R Example for KNN Regression** (Slides L2 p.15, and extended concept from ‘ex3.8.R’):

Listing 7: Conceptual KNN Regression in R

```

1 # Function from slide L2 p.15 (for single predictor)
2 knn_function_single_pred <- function(x0, x_train, y_train, K_val=20)
3 {
4   distances <- abs(x_train - x0)
5   ordered_indices <- order(distances)
6   neighbor_indices <- ordered_indices[1:K_val]
7   predicted_y <- mean(y_train[neighbor_indices])
8   return(predicted_y)
9 }
10
11 # Example usage (conceptual, needs a loop or apply for multiple x0)
12 # library(ISLR) # For Auto dataset
13 # attach(Auto)
14 # x_values_sorted <- sort(horsepower)
15 # y_predictions_knn <- sapply(x_values_sorted, knn_function_single_
16   pred,
17   x_train=horsepower, y_train=mpg, K_val
18   =5)
19 # plot(horsepower, mpg)
20 # lines(x_values_sorted, y_predictions_knn, col="blue", lwd=2)
21 # detach(Auto)
22
23 # For actual KNN, use packages like 'FNN' or 'class' (for
24   classification)
25 # library(FNN)
26 # knn.reg(train = as.matrix(horsepower_train), test = as.matrix(
27   horsepower_test),
28   y = mpg_train, k = 5)

```

– **Comparison to Linear Regression:**

- * *Assumptions*: Linear regression assumes a linear functional form. KNN is non-parametric and makes no such assumption.
- * *Performance*:
 - If the true relationship $f(X)$ is close to linear, linear regression usually performs better.
 - If $f(X)$ is highly non-linear, KNN can outperform linear regression, especially if n is large and p (number of predictors) is small.
- * *Curse of Dimensionality*: KNN’s performance degrades rapidly as p increases because the “nearest” neighbors can be very far away in high-dimensional space, making the local average less meaningful.
- * *Interpretability*: Linear regression coefficients are easy to interpret. KNN is less interpretable (a black box).

2.2.9 ISLR Chapter 3 Lab Highlights

- **Simple Linear Regression** (using Boston data, medv vs lstat):

- * Fitting: `lm(medv ~ lstat, data=Boston)()`.
- * Output: `summary()`, `names()`, `coef()`, `confint()`.
- * Prediction: `predict()` with `interval="confidence"` and `interval="prediction"`.
- * Plotting: `plot(lstat, medv)()`, `abline(lm.fit)()`.
- * Diagnostics: `par(mfrow=c(2,2))()`, `plot(lm.fit)()` (Residuals vs Fitted, Normal Q-Q, Scale-Location, Residuals vs Leverage). `hatvalues()`.
- **Multiple Linear Regression:**
 - * Fitting: `lm(medv ~ lstat + age)()`, `lm(medv ~ ., data=Boston)()`.
 - * VIF: `vif(lm.fit)()` from `car` package.
 - * Updating models: `update(lm.fit, . ~.-age)()`.
- **Interaction Terms:** `lm(medv ~ lstat:age)`.
- **Non-linear Transformations:** `lm(medv ~ lstat + I(lstat^2))`, `lm(medv ~ poly(lstat, 5))`, `lm(medv ~ log(rm))`.
 - * Comparing models: `anova(lm.fit.linear, lm.fit.quadratic)()`.
- **Qualitative Predictors** (using `Carseats` data):
 - * Factors are handled automatically. `contrasts(Carseats$ShelveLoc)()`.

2.2.10 Key Insights from ISLR Chapter 3 Exercises

- **ISLR 3.7 (Conceptual):** Proves that for simple linear regression, $R^2 = (\text{Cor}(X, Y))^2$. The proof involves expanding the definitions of R^2 , TSS , RSS , and $\text{Cor}(X, Y)$, and showing their algebraic equivalence using the formulas for $\hat{\beta}_0$ and $\hat{\beta}_1$.
- **ISLR 3.8 (Applied - Auto dataset):**
 - * Fit `lm(mpg ~ horsepower)`.
 - * Interpret relationship: Significant negative relationship.
 - * Predict `mpg` for `horsepower=98`, get confidence and prediction intervals.
 - * Plot data and regression line.
 - * Diagnostic plots: Reveal non-linearity (curved pattern in residuals vs. fitted).
- **ISLR 3.9 (Applied - Auto dataset):**
 - * Scatterplot matrix: `pairs(Auto)()`.
 - * Correlation matrix: `cor(subset(Auto, select=-name))()`.
 - * Multiple regression `lm(mpg ~ . -name, data=Auto)`.
 - * Identify significant predictors (displacement, weight, year, origin).
 - * Diagnostic plots for multiple regression: Similar non-linearity issues.
 - * Explore interaction terms (e.g., `cylindersdisplacement`).
 - * Explore non-linear transformations (e.g., `log(weight)`, `sqrt(horsepower)`, `I(acceleration)`).
 - * A better model might involve transforming the response, e.g., `lm(log(mpg) ~ ., data=Auto)`, which can improve linearity and homoscedasticity of residuals.
- **ISLR 3.10 (Applied - Carseats dataset):**
 - * Fit `lm(Sales ~ Price + Urban + US)`.
 - * Interpret coefficients: Price (negative effect), Urban (not significant), US (positive effect).

- * Model selection: Refit without Urban: `lm(Sales ~ Price + US)`.
- * Compare model fit (RSE, R2).
- * Confidence intervals for coefficients of the selected model.
- * Check for outliers and high leverage points using diagnostic plots.
- **ISLR 3.13, 3.14 (Simulation Exercises):**
 - * Simulate data with known $\beta_0, \beta_1, \sigma^2$.
 - * Fit models, compare $\hat{\beta}$ to true β .
 - * Observe effect of noise (σ^2) on model fit and confidence intervals.
 - * Demonstrate effect of collinearity on coefficient estimates and their significance.
 - * Effect of outliers/leverage points.

2.3 Lecture 3: Classification - Logistic Regression

2.3.1 Introduction to Classification

- **Goal:** Predict a qualitative (categorical) response variable, Y .
- Unlike regression (quantitative Y), classification assigns an observation to a class.
- Y can take values in a set of K classes, e.g., $\{1, 2, \dots, K\}$.
- The probability that Y belongs to class k is p_k , with $\sum_{k=1}^K p_k = 1$.
- Models estimate p_k as a function of predictors X_i : $p_k = P(Y_i = k | X_i = x_i) = f_k(x_i)$.

2.3.2 Measuring Prediction Quality in Classification

- **Error Rate:** Proportion of misclassified observations.
- Training Error Rate:

$$\text{trainER} = \frac{1}{n} \sum_{i=1}^n I(\hat{y}_i \neq y_i)$$

where $I(\cdot)$ is the indicator function (1 if true, 0 if false).

- Test Error Rate:

$$\text{testER} = \frac{1}{m} \sum_{i=n+1}^{n+m} I(\hat{y}_i \neq y_i)$$

Calculated on unseen test data of size m .

2.3.3 Why Not Linear Regression for Classification?

- If Y is binary (0/1), linear regression $\hat{Y} = \hat{\beta}_0 + \hat{\beta}_1 X$ can produce probabilities $\hat{p}(X)$ outside $[0, 1]$.
- The coding of classes (e.g., 0/1 vs. 1/2) affects the linear regression fit, which is undesirable for a classification method.
- Linear probability model (LPM): Using OLS on a 0/1 coded Y . (Slides L3 p. 7-8 illustrate this with `ISLR Default` data, showing predicted probabilities < 0 or > 1).

Listing 8: Linear Probability Model Example (Slides L3 p.8)

```

1 # Using Default dataset from ISLR
2 # y is 0/1 coded default status
3 # linprob <- lm(y ~ balance, data=Default)
4 # summary(linprob) # Shows significant relationship
5 # plot(balance, y)
6 # abline(linprob, col="red") # Line goes below 0 and above 1

```

2.3.4 Logistic Regression Model

- Models the probability $p(X) = P(Y = 1|X)$ using the logistic function:

$$p(X) = \frac{e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}}{1 + e^{\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p}} = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \dots + \beta_p X_p)}}$$

This ensures $0 \leq p(X) \leq 1$. (Slide L3 p.10 shows $p(x)$ for one predictor).

- **Logit Transformation / Log-Odds:**

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p$$

The logit is linear in X . $\frac{p(X)}{1 - p(X)}$ is called the odds.

- **Interpretation of Coefficients** (Slide L3 p.11):

- * A one-unit increase in X_j , holding other predictors constant, changes the log-odds by β_j .
- * Equivalently, it multiplies the odds by e^{β_j} .
- * Example (Slide L3 p.11, `Default` data, $Y = \text{default}$, $X = \text{studentYes}$): `glm(default ~ studentYes, family="binomial", data=Default)` Coefficients: (Intercept) -3.5041, studentYes 0.4049. $e^{0.4049} \approx 1.50$. The odds of default for a student are 1.50 times the odds for a non-student.

2.3.5 Estimating Coefficients: Maximum Likelihood Estimation (MLE)

- Goal: Find $\hat{\beta}_0, \hat{\beta}_1, \dots, \hat{\beta}_p$ that maximize the likelihood of observing the given data.
- **Likelihood Function** for binary $Y_i \in \{0, 1\}$:

$$L(\beta_0, \dots, \beta_p) = \prod_{i: y_i=1} p(x_i) \prod_{i': y_{i'}=0} (1 - p(x_{i'}))$$

where $p(x_i) = P(Y_i = 1|X_i = x_i; \beta_0, \dots, \beta_p)$.

- Often easier to maximize the **log-likelihood function** (Slide L3 p.14-15):

$$\ell(\beta_0, \dots, \beta_p) = \sum_{i=1}^n [y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i))]$$

$$\ell(\beta_0, \beta_1) = \sum_{i=1}^n [y_i(\beta_0 + \beta_1 x_i) - \log(1 + e^{\beta_0 + \beta_1 x_i})]$$

(for single predictor, derived from the formula on Slide L3 p.15)

- Maximization is done numerically (e.g., Iteratively Reweighted Least Squares - IRLS).
- R: `glm()` with `family=binomial` uses MLE.

2.3.6 Making Predictions and Evaluating Accuracy

- Prediction is a two-step process (Slide L3 p.10):
 1. Estimate $P(Y = 1|X = x)$ using the fitted logistic model: $\hat{p}(x)$.
 2. Classify as 1 if $\hat{p}(x) > \text{threshold}$ (often 0.5), else 0. (Slide L3 p.9 example: predict default if $p > 0.5$, implies balance i 2000).

		Predicted Class	
		Negative (0)	Positive (1)
– Confusion Matrix (Slides L3 p.16-17):	Actual Negative (0)	(TN)	(FP)
	Class Positive (1)	(FN)	(TP)

- Common Metrics from Confusion Matrix:
 - * *Overall Accuracy*: $(TN + TP)/(Total)$
 - * *Sensitivity (True Positive Rate, Recall)*: $TP/(TP + FN)$
 - * *Specificity (True Negative Rate)*: $TN/(TN + FP)$
 - * *Precision (Positive Predictive Value)*: $TP/(TP + FP)$
 - * *False Positive Rate*: $FP/(FP + TN) = 1 - \text{Specificity}$
- R Example (Default data, predicting default, Slides L3 p.17-18):

Listing 9: Logistic Regression and Confusion Matrix (Slides L3 p.17-18)

```

1 # Fit model (can be on full data or training data)
2 # logprob <- glm(default ~ student + balance + income, data=Default,
3   family="binomial")
4 # pred_probs <- predict(logprob, type="response")
5 # pred_class <- ifelse(pred_probs > 0.5, "Yes", "No") # Using 0.5
6   threshold
7
8 # conf_matrix_full <- table(Default$default, pred_class)
9 # print(conf_matrix_full)
10
11 # Example with train/test split (Slide L3 p.18)
12 set.seed(123)
13 n <- nrow(Default)
14 train_indices <- sample(1:n, n/2)
15 train_data <- Default[train_indices,]
16 test_data <- Default[-train_indices,]
17
18 logprob_train <- glm(default ~ student + balance + income, data=
19   train_data, family="binomial")
20 pred_probs_test <- predict(logprob_train, newdata=test_data, type="
21   response")
22 pred_class_test <- ifelse(pred_probs_test > 0.5, "Yes", "No")
23 conf_matrix_test <- table(test_data$default, pred_class_test)
24 print(conf_matrix_test)
25 # FALSE TRUE
26 # No 4809 20
27 # Yes 116 55
28 accuracy_test <- sum(diag(conf_matrix_test)) / sum(conf_matrix_test)
29 # accuracy_test is approx 0.9728

```

- **Issue with Imbalanced Classes** (Slide L3 p.19): If one class is much larger (e.g., "No default" is 96.7%), a naive classifier predicting the majority class for all observations can achieve high overall accuracy.
 - * Example: Predicting "Not default" for all in the test set on Slide L3 p.18 (4809+20 = 4829 "No" in test) would give $(4809 + 20)/(4809 + 20 + 116 +$

55) $\approx 4829/5000 \approx 0.9658$ if "No" was the only prediction for the No actuals. More accurately, if we only predict "No", $\text{accuracy} = (\text{TN} + 0) / \text{Total} = (4809 + 116) / (5000)$ if predicting no for "yes" actuals also. The slide implies classifying all 4839 actual "No" correctly, which means a classifier always predicting "No" gets $\text{TN}=4809$, $\text{FP}=0$, $\text{FN}=116+55=171$, $\text{TP}=0$. $\text{Acc} = 4809 / 5000 = 0.9618$. The slide's calculation $4839/5000$ seems to refer to the number of non-defaulters in the test set, implying a naive rule "predict No Default" would be correct for 4809 true "No"s, if we consider total test obs as 5000. If the test set had 4839 actual "No" and 161 actual "Yes", and we predict "No" always, $\text{accuracy} = 4839/5000 = 0.9678$.

- * Point: High overall accuracy is misleading here. Sensitivity (correctly identifying "Yes" defaults) is often more important for rare events.

2.3.7 Multiple Logistic Regression

- Model: $\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X_1 + \dots + \beta_p X_p$.
- Estimation and interpretation are analogous to simple logistic regression.
- **Confounding**: Relationship between a predictor and response can be distorted if other relevant predictors are omitted.

2.3.8 Logistic Regression for ≥ 2 Response Classes (Multinomial Logistic Regression)

- Not explicitly covered in slides, but ISLR discusses it.
- One class is chosen as baseline. Model $K - 1$ log-odds ratios relative to the baseline.
- E.g., for 3 classes (1, 2, 3), baseline class 3:

$$\log\left(\frac{P(Y = 1|X)}{P(Y = 3|X)}\right) = \beta_{01} + \beta_{11}X_1 + \dots$$

$$\log\left(\frac{P(Y = 2|X)}{P(Y = 3|X)}\right) = \beta_{02} + \beta_{12}X_1 + \dots$$

- Probabilities sum to 1.

2.3.9 Exercise: ISLR 4.13a-d (or old 4.10a-d)

(Corresponds to 'ex4.13.R' file, first part)

- (a) Explore **Weekly** data: `summary()`, `pairs()`, `cor()`. Look for patterns, e.g., Volume increases over Year. Lags don't show strong correlations with Today's return or Direction.
- (b) Logistic regression **Direction** `Lag1+Lag2+Lag3+Lag4+Lag5+Volume`:

Listing 10: Logistic Regression on Weekly Data (ISLR 4.13b)

```

1 # library(ISLR)
2 # data(Weekly)
3 # glm.fit <- glm(Direction ~ Lag1 + Lag2 + Lag3 + Lag4 + Lag5 +
  #               Volume,
4 #               data = Weekly, family = binomial)
```

```

5 # summary(glm.fit)
6 # Output: Lag2 appears to be statistically significant (p-value ~
  0.03).

```

- (c) Confusion Matrix and Interpretation:

Listing 11: Confusion Matrix for Weekly Data Logistic Model (ISLR 4.13c)

```

1 # glm.probs <- predict(glm.fit, type="response")
2 # glm.pred <- rep("Down", length(glm.probs))
3 # glm.pred[glm.probs > 0.5] <- "Up"
4 # conf_matrix <- table(Weekly$Direction, glm.pred)
5 # print(conf_matrix)
6 # #           glm.pred
7 # # Direction Down  Up
8 # #       Down   54  430
9 # #       Up    48  557
10 # accuracy <- mean(glm.pred == Weekly$Direction) # (54+557)/1089 =
  0.561
11 # # The model correctly predicts "Up" 557/(48+557) = 92.1% of the
  time it goes Up.
12 # # It correctly predicts "Down" 54/(54+430) = 11.2% of the time it
  goes Down.
13 # # The model is biased towards predicting "Up".

```

- (d) Train/Test Split, fit model with Lag2 only, evaluate on test:

- * Training: Years | 2009. Test: Years 2009-2010.
- * Fit `glm(Direction ~ Lag2, data=train_data, family = binomial)`. Predict on test data, create `test_predictions`.
- * Result in 'ex4.13.R': Accuracy ≈ 0.625 . This is better than chance (50%) and slightly better than always predicting "Up" for the test period (which would be $61/104 \approx 0.5865$).

2.4 Lecture 4: Classification - Discriminant Analysis & KNN

2.4.1 Linear Discriminant Analysis (LDA)

- Alternative to logistic regression, especially when classes are well-separated or n is small and predictors are approx. normal.
- **Bayes' Theorem for Classification** (Slides L4 p.3-4):
 - * Assign observation $X = x$ to class k that maximizes posterior probability $p_k(x) = P(Y = k|X = x)$.
 - * $p_k(x) = \frac{\pi_k f_k(x)}{\sum_{l=1}^K \pi_l f_l(x)}$
 - $\pi_k = P(Y = k)$: Prior probability of class k . Estimated as proportion of class k in training data ($\hat{\pi}_k = n_k/n$). (Slide L4 p.12)
 - $f_k(x) = P(X = x|Y = k)$: Density of X for an observation from class k .
 - * Bayes Classifier: Ideal, but $f_k(x)$ is usually unknown.
- **LDA Assumptions (for one predictor $p = 1$, $K = 2$ classes)** (Slide L4 p.8):
 1. $f_k(x)$ is Gaussian (Normal): $X|Y = k \sim N(\mu_k, \sigma_k^2)$.
 2. LDA specifically assumes common variance: $\sigma_1^2 = \sigma_2^2 = \dots = \sigma_K^2 = \sigma^2$.
- **LDA Discriminant Function ($\delta_k(x)$)** (Slides L4 p.10-11):

- * Maximize $p_k(x)$ is equivalent to maximizing $\log(p_k(x))$, and further equivalent to maximizing $\delta_k(x)$ after removing terms common to all classes.
- * For $p = 1$:

$$\delta_k(x) = x \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k)$$
- * Decision rule: Assign x to class k for which $\delta_k(x)$ is largest.
- * The decision boundary between two classes k and l (where $\delta_k(x) = \delta_l(x)$) is linear in x .
- **Parameter Estimation for LDA ($p = 1$)** (Slide L4 p.12):
 - * $\hat{\pi}_k = n_k/n$ (proportion of training obs in class k).
 - * $\hat{\mu}_k = \frac{1}{n_k} \sum_{i:y_i=k} x_i$ (average of x_i for class k).
 - * $\hat{\sigma}^2 = \frac{1}{n-K} \sum_{k=1}^K \sum_{i:y_i=k} (x_i - \hat{\mu}_k)^2$ (pooled variance). (Slide L4 p.12 has $\hat{\sigma}_k^2$ for QDA, then $\hat{\sigma}^2$ for LDA is essentially a weighted average if $p = 1$, or derived from pooled covariance for $p > 1$). The provided slide shows $\hat{\sigma}_k^2$ which is for QDA. For LDA with $p = 1$, the formula simplifies to a pooled estimate based on class-wise sums of squares. The formula in the slide for $\hat{\sigma}_k^2$ is for QDA, not LDA's common variance. ISLR p.141 gives common $\hat{\sigma}^2$.
- **LDA for $p > 1$ Predictors** (Slide L4 p.15):
 - * Assume $X|Y = k \sim N_p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$ (multivariate Gaussian with common covariance matrix $\boldsymbol{\Sigma}$).
 - * Discriminant function:

$$\delta_k(\mathbf{x}) = \mathbf{x}'\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k - \frac{1}{2}\boldsymbol{\mu}_k'\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \log(\pi_k)$$
 - * Decision boundaries are linear (hyperplanes).
- **R Implementation** (MASS package, Slides L4 p.13-14, 19):

Listing 12: LDA on Default Data (Slides L4 p.13-14 conceptual, ex4.13.R actual)

```

1 # Conceptual R code from slides for LDA (single predictor 'balance'
  # from Default data)
2 # x <- Default$balance
3 # cl <- Default$default # Factor with levels "No", "Yes"
4 # nk <- table(cl)
5 # n <- length(cl)
6 # pi_hat <- nk/n
7 # mu_hat <- as.matrix(by(x, cl, mean))
8 # # Pooled variance s2 (simplified for p=1, assuming K=2 classes)
9 # s2_no <- sum((x[cl=="No"] - mu_hat[1])^2)
10 # s2_yes <- sum((x[cl=="Yes"] - mu_hat[2])^2)
11 # s2_pooled <- (s2_no + s2_yes) / (n - 2) # This is (n-K)
12 #
13 # delta_k <- function(x_val, k_idx, mu_vec, s2_val, pi_vec) {
14 #   # k_idx would be 1 for "No", 2 for "Yes"
15 #   mu_k <- mu_vec[k_idx]
16 #   pi_k <- pi_vec[k_idx]
17 #   return(x_val * mu_k/s2_val - mu_k^2 / (2*s2_val) + log(pi_k))
18 # }
19 # # To predict: calculate delta1 and delta2 for each x, assign to
  # class with larger delta.
20 # delta1_vals <- delta_k(x, 1, mu_hat, s2_pooled, pi_hat)
21 # delta2_vals <- delta_k(x, 2, mu_hat, s2_pooled, pi_hat)
22 # pred_lda_manual <- ifelse(delta2_vals > delta1_vals, "Yes", "No")

```

```

23 # table(cl, pred_lda_manual)
24
25 # Using MASS::lda() (from ex4.13.R for Weekly data)
26 # library(MASS)
27 # train_indices <- (Weekly$Year < 2009)
28 # train_data <- Weekly[train_indices,]
29 # test_data <- Weekly[!train_indices,]
30 # lda.fit <- lda(Direction ~ Lag2, data=train_data)
31 # lda.pred_obj <- predict(lda.fit, newdata=test_data)
32 # lda.class <- lda.pred_obj$class
33 # conf_matrix_lda <- table(test_data$Direction, lda.class)
34 # print(conf_matrix_lda) # Test accuracy was 0.625
35 # head(lda.pred_obj$posterior) # Shows posterior probabilities

```

2.4.2 Quadratic Discriminant Analysis (QDA)

- Similar to LDA, but assumes each class k has its own covariance matrix Σ_k .
- Assumption: $X|Y = k \sim N_p(\mu_k, \Sigma_k)$.
- Discriminant function $\delta_k(x)$ becomes quadratic in x :

$$\delta_k(\mathbf{x}) = -\frac{1}{2} \log |\Sigma_k| - \frac{1}{2}(\mathbf{x} - \mu_k)' \Sigma_k^{-1}(\mathbf{x} - \mu_k) + \log(\pi_k)$$

- Decision boundaries are quadratic.
- **LDA vs. QDA Trade-off:**
 - * LDA: Estimates p parameters for μ_k (for each class), and $p(p+1)/2$ for common Σ . Less flexible.
 - * QDA: Estimates p parameters for μ_k and $p(p+1)/2$ for Σ_k (for each of K classes). More flexible.
 - * QDA has higher variance but lower bias (if true decision boundary is non-linear).
 - * LDA better if n is small or common covariance assumption is reasonable. QDA better if n is large and common covariance is violated.
- R Implementation (MASS package):

Listing 13: QDA on Weekly Data (from ex4.13.R)

```

1 # library(MASS)
2 # qda.fit <- qda(Direction ~ Lag2, data=train_data) # Using same
  train/test as LDA
3 # qda.pred_obj <- predict(qda.fit, newdata=test_data)
4 # qda.class <- qda.pred_obj$class
5 # conf_matrix_qda <- table(test_data$Direction, qda.class)
6 # print(conf_matrix_qda) # Test accuracy was 0.5865 (predicted "Up"
  always)

```

2.4.3 Comparison of Classification Methods (Logistic, LDA, QDA, KNN)

- **Logistic Regression vs. LDA:**
 - * Both produce linear decision boundaries (if $p > 1$).
 - * LDA assumes Gaussian $f_k(x)$ with common Σ . LR makes no such assumption about $f_k(x)$, models $P(Y = k|X)$ directly.

- * If Gaussian assumption holds, LDA is more efficient (stable estimates with smaller n).
- * LR is generally more robust if Gaussian assumption is violated.
- * In practice, often similar performance.
- * LDA parameter estimates can be unstable if classes are well-separated (LR too). (Slide L4 p.16)
- * LDA more stable than LR for small n if X is approx. multinormal. (Slide L4 p.16)
- * LDA is a natural approach for $K > 2$ classes. (Slide L4 p.16)
- **KNN:**
 - * Completely non-parametric. Makes no assumptions about decision boundary shape.
 - * Can outperform parametric methods if boundary is highly non-linear.
 - * Requires large n for good performance, especially if p is large (curse of dimensionality).
 - * Optimal K is crucial.
- **QDA:**
 - * Compromise between non-parametric KNN and rigid LDA/LR.
 - * Assumes Gaussian $f_k(x)$ but allows different Σ_k .
 - * Good if true boundary is moderately non-linear and Gaussian assumption is reasonable.

2.4.4 Evaluating Classification Models (Revisited)

- **Threshold Choice** (Slide L4 p.18):
 - * Default threshold for binary classification is 0.5 for $P(Y = 1|X)$.
 - * Can be adjusted based on costs of misclassification or to balance sensitivity/specificity.
 - * Example: For `Default` data, one might use threshold \neq 0.5 (e.g., 0.2) to identify more potential defaulters, accepting more false positives.
- **ROC Curve (Receiver Operating Characteristics)** (Slides L4 p.19-21):
 - * Plots True Positive Rate (Sensitivity) vs. False Positive Rate (1 - Specificity) for various threshold values.
 - * A good classifier has ROC curve far from the 45-degree line (random guessing) towards top-left corner.
 - * **AUC (Area Under Curve):** Summary measure of ROC performance.
 - $AUC = 1$: Perfect classifier.
 - $AUC = 0.5$: Random guessing.
 - $AUC \neq$ 0.7 generally considered acceptable, \neq 0.8 good, \neq 0.9 excellent.
- R code for ROC (conceptual from Slides L4 p.21, using LDA predictions):

Listing 14: Generating ROC Curve Data (Conceptual, Slides L4 p.21)

```

1 # Assume lda1 fitted on cl ~ x (Default data)
2 # pr <- predict(lda1)$posterior # Posterior probabilities P(Y=k|X)
3 # cl_numeric <- as.numeric(Default$default) - 1 # 0 for No, 1 for
   Yes

```

```

4 #
5 # thrange <- seq(0.01, 0.99, by=0.01) # Thresholds to test
6 # roc_data <- data.frame(FPrate=numeric(length(thrange)),
7 #                         TPrate=numeric(length(thrange)))
8 #
9 # for (i in 1:length(thrange)) {
10 #   th <- thrange[i]
11 #   pred_class <- ifelse(pr[, "Yes"] > th, 1, 0) # Predict "Yes" if
12 #   P(Yes|X) > th
13 #   TP <- sum(pred_class == 1 & cl_numeric == 1)
14 #   FN <- sum(pred_class == 0 & cl_numeric == 1)
15 #   FP <- sum(pred_class == 1 & cl_numeric == 0)
16 #   TN <- sum(pred_class == 0 & cl_numeric == 0)
17 #
18 #   roc_data$TPrate[i] <- TP / (TP + FN) # Sensitivity
19 #   roc_data$FPrate[i] <- FP / (FP + TN) # 1 - Specificity
20 # }
21 # plot(roc_data$FPrate, roc_data$TPrate, type="l", col="red", lwd=2,
22 #       xlab="False Positive Rate", ylab="True Positive Rate", main="
23 #       ROC Curve")
24 # abline(a=0, b=1, lty=2) # Line of no discrimination
25 # # Packages like 'pROC' or 'ROCR' can do this more easily and
26 #   calculate AUC.

```

2.4.5 K-Nearest Neighbors (KNN) Classifier

- To classify a test observation x_0 :
 1. Find the K training points closest to x_0 (neighborhood N_0).
 2. Estimate conditional probability for class j : $P(Y = j|X = x_0) = \frac{1}{K} \sum_{i \in N_0} I(y_i = j)$.
 3. Assign x_0 to the class with the largest probability (majority vote among neighbors).
- Choice of K is critical, often selected by CV.
- Small K : Flexible boundary, low bias, high variance.
- Large K : Smoother, less flexible boundary, high bias, low variance.
- R Implementation (class package):

Listing 15: KNN Classification (from ex4.13.R for Weekly data)

```

1 # library(class)
2 # train_X_knn <- as.matrix(train_data$Lag2)
3 # test_X_knn <- as.matrix(test_data$Lag2)
4 # train_Direction_knn <- train_data$Direction
5 #
6 # set.seed(1) # For reproducibility if there are ties
7 # knn.pred_k1 <- knn(train=train_X_knn, test=test_X_knn, cl=train_
8 #   Direction_knn, k=1)
9 # conf_matrix_knn_k1 <- table(test_data$Direction, knn.pred_k1)
10 # print(conf_matrix_knn_k1) # Test accuracy 0.5
11 #
12 # # Trying different K values (e.g., K=10)
13 # # knn.pred_k10 <- knn(train=train_X_knn, test=test_X_knn, cl=train_
14 #   Direction_knn, k=10)
15 # # accuracy_k10 <- mean(knn.pred_k10 == test_data$Direction)

```

2.4.6 Exercises: ISLR 4.13e-i (or old 4.10e-i)

(Corresponds to 'ex4.13.R' file, second part)

- (e) LDA on **Direction** **Lag2** (train/test split as before):
 - * Fit LDA, predict on test set, calculate accuracy.
 - * *Result from 'ex4.13.R':* Accuracy ≈ 0.625 . Same as logistic regression with only **Lag2**.
- (f) QDA on **Direction** **Lag2**:
 - * Fit QDA, predict, calculate accuracy.
 - * *Result from 'ex4.13.R':* Accuracy ≈ 0.5865 . QDA predicts "Up" for all test observations, performing worse than LDA/logistic.
- (g) KNN with K=1 on **Direction** **Lag2**:
 - * Fit KNN (K=1), predict, calculate accuracy.
 - * *Result from 'ex4.13.R':* Accuracy ≈ 0.50 . Performs poorly.
- (h) Compare Logistic, LDA, QDA, KNN(K=1):
 - * Logistic regression (with **Lag2**) and LDA performed best and identically (accuracy 0.625).
 - * QDA and KNN(K=1) performed worse.
- (i) Experiment with different combinations/transformations/K for KNN:
 - * Logistic with **Lag2:Lag1** interaction: Accuracy ≈ 0.5865 .
 - * LDA with **Lag2:Lag1** interaction: Accuracy ≈ 0.5769 .
 - * QDA with **Lag2 + sqrt(abs(Lag2))**: Accuracy ≈ 0.5769 .
 - * KNN with K=10: Accuracy ≈ 0.5769 .
 - * KNN with K=100: Accuracy ≈ 0.5577 .
 - * *Conclusion:* Original simpler logistic regression and LDA with only **Lag2** were the best performers among these experiments.

2.5 Lecture 5: Resampling Methods - Cross-Validation

2.5.1 Introduction to Resampling Methods

- **Sampling Distribution** : The distribution of a statistic (e.g., sample mean \bar{X}) if we were to repeatedly draw samples from the population.
 - * Example: Estimating average height $\mu = E(X)$ of Norwegian adults. Each sample of $n = 100$ gives one estimate $\hat{\mu} = \bar{x}$. The collection of these \bar{x} values from many hypothetical samples forms the sampling distribution. (Slides L5 p.3-5 show two such sample means: 183.678 and 180.678).
 - * **Central Limit Theorem (CLT)** (Slides L5 p.6, 9): For large n , the sampling distribution of \bar{X} is approximately Normal: $\bar{X} \sim N(\mu, \sigma^2/n)$.
 - * We can estimate this sampling distribution using a single sample by plugging in estimates $\hat{\mu} = \bar{x}$ and $\hat{\sigma} = s$: $\bar{X} \sim N(\bar{x}, s^2/n)$. (Slides L5 p.10-11). This allows for constructing confidence intervals (Slide L5 p.12).
- **Why Resampling?** (Slide L5 p.13):

- * Large sample approximations (like CLT) may not hold for small n or for complex statistics where the asymptotic distribution is hard to derive.
- * Resampling methods mimic sampling from a population by instead repeatedly sampling from our *observed sample data*.
- * Often used to estimate test error $E((Y_i - \hat{f}(X_i))^2)$ or to assess variability of model parameters.

2.5.2 Cross-Validation (CV)

– Core Idea:

1. Split data into a *training set* and a *test (or validation) set*.
2. Fit (train/estimate) the model on the training set.
3. Evaluate model performance (e.g., predict) on the test set.

– Purpose:

- * *Model Assessment*: Estimating the test error of a final chosen model.
- * *Model Selection*: Choosing the appropriate level of flexibility (e.g., degree of polynomial, value of K in KNN).

– Types of CV: Validation Set Approach, Leave-One-Out CV (LOOCV), k-Fold CV.

2.5.3 The Validation Set Approach

– Procedure (Slide L5 p.15):

1. Randomly split the n observations into a training set (e.g., $m \approx n/2$ observations) and a validation/test set ($n - m$ observations).
2. Fit the model using only the training data to get \hat{f} .
3. Evaluate \hat{f} on the validation set by calculating the test MSE (or other error metric):

$$\text{Test MSE}_{\text{valid}} = \frac{1}{n - m} \sum_{i \in \text{valid.set}} (y_i - \hat{f}(x_i))^2$$

– R Example (ISLR Auto data) (Slides L5 p.16-21):

- * Predict mpg from horsepower.

Listing 16: Validation Set Approach in R (Slides L5 p.16, 18, 21)

```

1 library(ISLR)
2 # names(Auto) # mpg, cylinders, displacement, horsepower, ...
3 n <- nrow(Auto)
4 set.seed(1) # For reproducibility of the random split
5 draw <- sample(1:n, size=floor(n/2)) # Indices for training set
6 train_data <- Auto[draw, ]
7 test_data <- Auto[-draw, ]
8
9 # Model 1: Linear regression
10 mod1 <- lm(mpg ~ horsepower, data=train_data)
11 # summary(mod1) # (Slide L5 p.18 shows an example output)
12
13 # Model 2: Polynomial regression (degree 2)
14 mod2 <- lm(mpg ~ horsepower + I(horsepower^2), data=train_data)
15 # summary(mod2) # (Slide L5 p.20 shows an example output)
16

```

```

17 # Predicting on test data and calculating Test MSE
18 pred1_test <- predict(mod1, newdata=test_data)
19 mse1_test <- mean((test_data$mpg - pred1_test)^2)
20 # mse1_test -> 27.36073 (from slide L5 p.21, actual value
    depends on seed)
21
22 pred2_test <- predict(mod2, newdata=test_data)
23 mse2_test <- mean((test_data$mpg - pred2_test)^2)
24 # mse2_test -> 20.29991 (from slide L5 p.21, actual value
    depends on seed)
25 # Polynomial model has lower Test MSE in this specific split.

```

– **Drawbacks of Validation Set Approach** (ISLR Ch 5.1.1, Conceptual Ex 5.3b.i):

1. *High Variability*: The estimated test MSE can be highly variable depending on which observations end up in the training vs. validation set. Different splits can lead to different conclusions about model performance. (The slide L5 p.21 asks "Did you get the exact same numbers?" highlighting this variability).
2. *Overestimation of Test Error*: Only a subset of data ($m < n$) is used for training. Models fit on less data tend to perform worse. So, validation set MSE might overestimate the test error of a model fit to the full dataset.

2.5.4 Leave-One-Out Cross-Validation (LOOCV)

– **Procedure** (Slide L5 p.22):

1. For each observation $i = 1, \dots, n$:
 - * Hold out observation (x_i, y_i) (this is the "test set" of size 1).
 - * Fit the model $\hat{f}^{(-i)}$ using the remaining $n - 1$ observations (training set).
 - * Predict y_i using $\hat{f}^{(-i)}(x_i)$ and calculate the squared error: $MSE_i = (y_i - \hat{f}^{(-i)}(x_i))^2$.
2. The LOOCV estimate of test MSE is the average of these n errors:

$$CV_{(n)} = \text{LOOCV MSE} = \frac{1}{n} \sum_{i=1}^n MSE_i$$

– **R Example (Auto data)** (Slide L5 p.23):

Listing 17: LOOCV for Auto Data (Conceptual from Slide L5 p.23)

```

1 # MSE1_loo <- numeric(n) # For linear model
2 # MSE2_loo <- numeric(n) # For quadratic model
3 #
4 # for(i in 1:n) {
5 #   train_loo <- Auto[-i, ]
6 #   test_loo <- Auto[i, ]
7 #
8 #   mod1_loo <- lm(mpg ~ horsepower, data=train_loo)
9 #   pred1_loo <- predict(mod1_loo, newdata=test_loo)
10 #   MSE1_loo[i] <- (test_loo$mpg - pred1_loo)^2
11 #
12 #   mod2_loo <- lm(mpg ~ horsepower + I(horsepower^2), data=train_
    loo)
13 #   pred2_loo <- predict(mod2_loo, newdata=test_loo)
14 #   MSE2_loo[i] <- (test_loo$mpg - pred2_loo)^2
15 # }
16 #

```

```

17 # mean_MSE1_loo <- mean(MSE1_loo) # Slide says 24.23
18 # mean_MSE2_loo <- mean(MSE2_loo) # Slide says 19.25 (Quadratic is
    better)
19
20 # Using boot::cv.glm() for LOOCV with linear models (more efficient)
21 # library(boot)
22 # glm.fit.linear <- glm(mpg ~ horsepower, data=Auto)
23 # cv.err.linear <- cv.glm(Auto, glm.fit.linear) # Default is LOOCV
24 # print(cv.err.linear$delta[1]) # First element is raw LOOCV MSE
25 #
26 # glm.fit.quad <- glm(mpg ~ poly(horsepower, 2), data=Auto) # poly()
    is preferred
27 # cv.err.quad <- cv.glm(Auto, glm.fit.quad)
28 # print(cv.err.quad$delta[1])

```

– **Advantages of LOOCV:**

- * *Less Bias*: Uses $n - 1$ observations for training in each fold, so $\hat{f}^{(-i)}$ is very similar to \hat{f} fit on all n observations. LOOCV MSE tends to be an almost unbiased estimate of test error for a model trained on n observations.
- * *No Randomness*: Result is always the same, no variability due to random splits.

– **Disadvantages of LOOCV** (ISLR Ch 5.1.2, Conceptual Ex 5.3b.ii):

- * *Computationally Expensive*: Model must be fit n times. Can be very slow for complex models or large n . (For OLS linear regression and polynomial regression, a computational shortcut exists: $CV_{(n)} = \frac{1}{n} \sum (\frac{y_i - \hat{y}_i}{1 - h_{ii}})^2$, where h_{ii} is leverage of i -th obs. So, only need to fit full model once).
- * *High Variance*: The n training sets are highly overlapping (share $n - 2$ observations). The MSE_i values are highly correlated, so their average ($CV_{(n)}$) can have high variance. It can be a poor estimator of the true test MSE.

2.5.5 k-Fold Cross-Validation

– **Procedure:**

1. Randomly divide the n observations into k non-overlapping groups (folds) of approximately equal size (n/k).
2. For each fold $j = 1, \dots, k$:
 - * Hold out fold j as the test set.
 - * Fit the model $\hat{f}^{(-j)}$ using the other $k - 1$ folds as the training set.
 - * Calculate $MSE_j = \frac{1}{n_j} \sum_{i \in \text{fold } j} (y_i - \hat{f}^{(-j)}(x_i))^2$, where n_j is size of fold j .
3. The k -fold CV estimate of test MSE is:

$$CV_{(k)} = \frac{1}{k} \sum_{j=1}^k MSE_j$$

- Common choices for k : 5 or 10. (Slide L5 p.24 notes $k = n$ is LOOCV).

– **Advantages over LOOCV:**

- * *Computationally Cheaper*: Model fit only k times.
- * *Lower Variance*: Training sets for each fold are less overlapping than in LOOCV, leading to less correlated MSE_j values. $CV_{(k)}$ often has lower variance than $CV_{(n)}$.

- **Bias-Variance Trade-off for Choice of k :**
 - * *Bias*: $CV_{(k)}$ uses training sets of size $n(k-1)/k$, which is smaller than $n-1$ (for LOOCV). So $CV_{(k)}$ might have slightly more bias as an estimator of test error for a model fit on full data, compared to LOOCV.
 - * *Variance*: $CV_{(k)}$ generally has lower variance than LOOCV.
 - * $k = 5$ or $k = 10$ often strike a good balance.
- **CV for Classification Problems:** Use error rate instead of MSE.

$$CV_{(k)} = \frac{1}{k} \sum_{j=1}^k \text{Err}_j = \frac{1}{k} \sum_{j=1}^k \left(\frac{1}{n_j} \sum_{i \in \text{fold } j} I(y_i \neq \hat{y}_i) \right)$$

- R Implementation (using `boot::cv.glm()`):

Listing 18: k-Fold CV using `boot::cv.glm`

```

1 # library(boot)
2 # For k-fold CV, specify K argument in cv.glm
3 # Example: 10-fold CV for linear model on Auto data
4 # glm.fit.auto <- glm(mpg ~ horsepower, data=Auto)
5 # cv.err.10fold <- cv.glm(Auto, glm.fit.auto, K=10)
6 # print(cv.err.10fold$delta[1]) # Raw 10-fold CV MSE
7
8 # Example: 10-fold CV for logistic regression on Default data (ISLR
9   Ch5 Ex5)
10 # glm.fit.default <- glm(default ~ income + balance, data=Default,
11   family=binomial)
12 # cv.err.default.10fold <- cv.glm(Default, glm.fit.default, K=10) #
13   Cost function needed for error rate
14 # To get error rate for classification, must define a cost function
15   for cv.glm
16 # Or implement k-fold CV manually:
17 # k <- 10
18 # folds <- sample(cut(seq(1,nrow(Default)),breaks=k,labels=FALSE))
19 # cv_errors <- numeric(k)
20 # for(j in 1:k){
21 #   test_indices <- which(folds==j, arr.ind=TRUE)
22 #   test_data_fold <- Default[test_indices, ]
23 #   train_data_fold <- Default[-test_indices, ]
24 #   fit_fold <- glm(default ~ income + balance, data=train_data_fold,
25     family=binomial)
26 #   probs_fold <- predict(fit_fold, newdata=test_data_fold, type="
27     response")
28 #   preds_fold <- ifelse(probs_fold > 0.5, "Yes", "No")
29 #   cv_errors[j] <- mean(preds_fold != test_data_fold$default)
30 # }
31 # mean_cv_error_10fold <- mean(cv_errors)

```

2.5.6 Pros and Cons of Different CV Approaches

Table 1: Comparison of Cross-Validation Methods (Slide L5 p.25, ISLR Ch 5.1.4)

Property	Validation Set	LOOCV	k-Fold CV
Bias of Test Error Estimate	Higher	Lower (Approx. Unbiased)	Intermediate
Variance of Test Error Estimate	Higher	Higher	Lower
Computational Time	Best (1 fit)	Worst (n fits)	Intermediate (k fits)

- The table summarizes that LOOCV is good for low bias in estimating test error but can have high variance and is computationally expensive. k-Fold CV (e.g., $k = 5$ or 10) often provides a better bias-variance trade-off for the estimate of test error itself and is computationally more feasible. The validation set approach is simple but can be unreliable.

2.5.7 The Right and Wrong Way to do Cross-Validation

- **Crucial Point:** Any variable selection, feature engineering, or parameter tuning step that relies on the outcomes (y_i) must be performed *inside* the CV loop, using only the training data of that specific fold.
- **Wrong Way:** Perform variable selection on the entire dataset first, then use CV to estimate the error of the chosen model. This "leaks" information from the test folds into the model selection process, leading to an overly optimistic (too low) estimate of test error.
- **Right Way:** In each CV fold j :
 1. Perform variable selection (or other tuning) using *only the training part* of fold j .
 2. Fit the model selected in step 1 using *only the training part* of fold j .
 3. Evaluate this model on the *test part* (hold-out portion) of fold j .

2.5.8 Exercise: ISLR 5.5 (Cross-validation for Logistic Regression)

- (a) Fit logistic regression `default ~ income + balance` on full dataset.
- (b) Validation Set Approach:
 - * Split data 50/50.
 - * Fit model on training.
 - * Predict on validation, calculate test error rate (misclassification).
 - * *Result from 'ex5.5.R':* Test error rate $\approx 2.36\%$ to 2.86% depending on split.
- (c) Repeat (b) multiple times: Shows variability of validation set error.
- (d) Validation Set with `student` dummy: Add `student` predictor.
 - * Fit `default ~ income + balance + student` on training.
 - * Test error rate $\approx 2.64\%$. Does not seem to improve much over model without `student` using this single validation split. (More robust comparison would require repeated CV or k-fold CV).
- (Extra Task from 'ex5.5.R': LOOCV on smaller subset)
 - * Illustrates manual LOOCV loop for logistic regression on 500 observations.
 - * For logistic regression, `cv.glm()` with default settings (or `K=nrow(data)`) performs LOOCV but requires a custom cost function to output error rate instead of deviance for binomial models.

Listing 19: Manual LOOCV for Classification Error (Conceptual)

```

1 # n_subset <- 500
2 # subset_data <- Default[sample(1:nrow(Default), n_subset), ]
3 # loocv_errors <- numeric(n_subset)
4 # for(i in 1:n_subset) {

```



```

5 # train_fold <- subset_data[-i, ]
6 # test_fold <- subset_data[i, ]
7 # fit_fold <- glm(default ~ income + balance + student,
8 #                 data=train_fold, family=binomial)
9 # prob_fold <- predict(fit_fold, newdata=test_fold, type="
  response")
10 # pred_fold <- ifelse(prob_fold > 0.5, "Yes", "No")
11 # loocv_errors[i] <- (pred_fold != test_fold$default)
12 # }
13 # mean_loocv_error <- mean(loocv_errors)

```

2.6 Lecture 6: Resampling Methods - The Bootstrap

2.6.1 Introduction to the Bootstrap

- **Purpose:** A powerful and widely applicable tool for quantifying uncertainty associated with a given estimator or statistical learning method.
- Primarily used to estimate the standard error (SE) of an estimator.
- Can also be used to construct confidence intervals.
- Useful when the true sampling distribution of a statistic is unknown or difficult to derive analytically (e.g., for medians, quantiles, complex model parameters).
- **Core Idea:** Mimic the process of obtaining new sample sets from the population by repeatedly sampling *with replacement* from the original observed dataset.
- Each "bootstrap sample" has the same size n as the original dataset.
- Some observations from the original dataset may appear multiple times in a bootstrap sample, while others may not appear at all.

2.6.2 The Bootstrap Procedure for Estimating Standard Error

1. Let the original dataset be $Z = \{z_1, z_2, \dots, z_n\}$.
2. Generate B independent bootstrap samples Z^1, Z^2, \dots, Z^B . Each Z^b is obtained by drawing n observations from Z with replacement.
3. For each bootstrap sample Z^b , compute the statistic of interest, $\hat{\alpha}^b$. This could be a sample mean, median, regression coefficient, etc.
4. The bootstrap estimate of the standard error of $\hat{\alpha}$ (the statistic computed on the original data) is the standard deviation of the B bootstrap estimates:

$$SE_B(\hat{\alpha}) = \sqrt{\frac{1}{B-1} \sum_{b=1}^B \left(\hat{\alpha}^b - \frac{1}{B} \sum_{r=1}^B \hat{\alpha}^r \right)^2}$$

5. A large number of bootstrap samples B (e.g., $B = 1000$ or more) is typically used.

2.6.3 Example: Estimating SE of Investment Allocation

- Task: Minimize variance of portfolio $\alpha X + (1 - \alpha)Y$.
- Optimal $\alpha = \frac{\sigma_Y^2 - \sigma_{XY}}{\sigma_X^2 + \sigma_Y^2 - 2\sigma_{XY}}$.

- Suppose we have $n = 100$ observations of (X, Y) . We can estimate $\hat{\sigma}_X^2, \hat{\sigma}_Y^2, \hat{\sigma}_{XY}$ from this sample, and then $\hat{\alpha}$.
- To find $SE(\hat{\alpha})$ using bootstrap:
 1. Draw B bootstrap samples (X^b, Y^b) of size $n = 100$ from the original 100 observations.
 2. For each bootstrap sample b , calculate $\hat{\alpha}^b$ using the sample variances and covariance from that bootstrap sample.
 3. Compute $SE_B(\hat{\alpha})$ as the standard deviation of these B values of $\hat{\alpha}^b$.
- This provides a measure of how much $\hat{\alpha}$ would vary if we repeatedly sampled 100 observations from the true population.

2.6.4 R Implementation of Bootstrap

- The `boot` package provides the `boot()` function for easy bootstrap implementation.
- **Key arguments for `boot()`** (ISLR Ch 5 Ex 5.6):
- `data`: The original dataset.
- `statistic`: A function that takes two arguments: `data` and `index`. The `index` argument will contain the indices of the observations selected for a particular bootstrap sample. The function should return the statistic(s) of interest computed on `data[index,]`.
- `R`: The number of bootstrap replicates B .
- Example: Bootstrap SE for coefficients of logistic regression on Default data (ISLR Ch 5 Ex 5.6).

Listing 20: Bootstrap for Logistic Regression Coefficients (ISLR Ch 5 Ex 5.6)

```

1 library(ISLR) # For Default dataset
2 library(boot) # For boot() function
3
4 # (a) Fit the logistic regression model once on original data
5 glm.fit.full <- glm(default ~ income + balance, data =
6   Default, family = binomial)
7
8 # summary(glm.fit.full)$coef # Shows original estimates and
9   SEs from asymptotic theory
10
11 # (b) Define the statistic function for bootstrap
12 # This function will be called by boot() for each bootstrap
13   sample.
14 # It needs to return the coefficients of the logistic
15   regression model.
16 boot.fn.coefs <- function(data, index) {
17   # Fit model on the bootstrap sample (data[index,])
18   fit <- glm(default ~ income + balance, data = data[index,
19     ], family = binomial)
20   return(coef(fit)) # Return the vector of estimated
21     coefficients
22 }
23
24 # (c) Run the bootstrap
25 set.seed(1) # For reproducibility
26 boot_results <- boot(data = Default, statistic = boot.fn.
27   coefs, R = 1000) # B=1000 replicates
28
29 # Print the bootstrap results

```

```

22 print(boot_results)
23 # Output includes:
24 # original: Coefficients from the model fit on the original
    data
25 # bias: Difference between mean of bootstrap estimates and
    original estimate
26 # std. error: Bootstrap estimate of SE for each coefficient
27
28 # (d) Comparison: Bootstrap SEs are usually close to SEs from
    summary(glm.fit.full),
29 # but bootstrap is more general and doesn't rely on
    asymptotic theory.
30 # E.g., from Ex 5.6 solution:
31 # Original glm SE for income: 4.99e-06, balance: 2.27e-04
32 # Bootstrap SE for income: ~4.5e-06, balance: ~2.3e-04 (
    approx from R=50 in book example)

```

- Example: Bootstrap SE for the sample median (ISLR Ch 5 Ex 5.9f, Boston data).

Listing 21: Bootstrap SE for Median (ISLR Ch 5 Ex 5.9f)

```

1 library(MASS) # For Boston dataset
2 library(boot)
3 # attach(Boston) # or use Boston$medv
4
5 # (e) Calculate median on original data
6 medv.median.original <- median(Boston$medv) # 21.2
7
8 # (f) Define statistic function and run bootstrap
9 boot.fn.median <- function(data_vector, index) {
10   return(median(data_vector[index]))
11 }
12 set.seed(1)
13 boot_median_results <- boot(data = Boston$medv, statistic =
    boot.fn.median, R = 1000)
14 print(boot_median_results)
15 # Bootstrap Statistics :
16 #   original   bias   std. error
17 # t1      21.2 -0.0098      0.3874 (from ISLR solution)
18 # The bootstrap SE for the median is ~0.387.
19 # detach(Boston)

```

2.6.5 Bootstrap Confidence Intervals

- Standard $(1 - \alpha)\%$ Normal-based CI: $\hat{\alpha} \pm z_{1-\alpha/2} \cdot SE_B(\hat{\alpha})$. Relies on $\hat{\alpha}$ being approx. Normal and $SE_B(\hat{\alpha})$ being a good estimate.
- **Percentile Confidence Interval:** A more direct method.
 1. Obtain B bootstrap estimates $\hat{\alpha}^1, \dots, \hat{\alpha}^B$.
 2. Sort them: $\hat{\alpha}^{(1)} \leq \hat{\alpha}^{(2)} \leq \dots \leq \hat{\alpha}^{(B)}$.
 3. A $(1 - \alpha)\%$ percentile CI is $[\hat{\alpha}^{(\lceil B\alpha/2 \rceil)}, \hat{\alpha}^{(\lceil B(1-\alpha/2) \rceil)}]$.
 4. E.g., for a 95% CI ($\alpha = 0.05$) with $B = 1000$, use the 25th and 975th sorted bootstrap estimates.
- Other types: Bias-Corrected and Accelerated (BCa) intervals (often better but more complex).
- R: `boot.ci()` function from `boot` package can compute various CIs.

Listing 22: Bootstrap Confidence Intervals using boot.ci

```

1 # Assuming boot_results from logistic regression example
   above
2 # conf_intervals <- boot.ci(boot_results, type = c("norm", "
   perc", "bca"), index = 2) # For 'income' coeff
3 # print(conf_intervals)
4 # index=1 for intercept, index=2 for income, index=3 for
   balance
5
6 # For median example:
7 # boot.ci(boot_median_results, type=c("norm", "perc", "bca")
   )

```

2.6.6 When is Bootstrap Useful?

- When it's hard to derive SEs or CIs analytically (e.g., for medians, quantiles, ratios of parameters, complex model outputs).
- For small sample sizes where asymptotic theory for SEs might not hold well.
- To check assumptions of simpler SE formulas.

2.6.7 Limitations of Bootstrap

- **Assumption:** The empirical distribution (from the original sample) is a good approximation of the true population distribution. This may not hold if the sample is very small or not representative.
- **Extreme Quantiles/Parameters:** Bootstrap may perform poorly for estimating properties related to the tails of a distribution if the original sample doesn't capture those tails well.
- **Computational Cost:** Can be intensive if B is large and the statistic computation is complex.
- **Not a substitute for more data:** Bootstrap quantifies uncertainty based on the current sample; it doesn't improve the point estimate itself in the way more data would.
- **Dependence:** Standard bootstrap assumes i.i.d. observations. Modifications exist for time series (e.g., block bootstrap) or dependent data, but are more complex.

2.6.8 Conceptual Exercise Insights

- **(a,b)** Probability a specific observation j is not in a bootstrap sample: $(1 - 1/n)$.
- **(c)** Probability observation j is not in a bootstrap sample of size n : $(1 - 1/n)^n$.
- **(d,e,f,g)** As $n \rightarrow \infty$, $(1 - 1/n)^n \rightarrow e^{-1} \approx 0.368$. So, probability observation j is in the bootstrap sample approaches $1 - e^{-1} \approx 0.632$.
- This means, on average, about 63.2% of original observations are present in any given bootstrap sample (these are the "in-bag" samples). The remaining 36.8

Listing 23: Probability an Observation is in a Bootstrap Sample

```

1 pr_in_bootstrap <- function(n) {
2   return(1 - (1 - 1/n)^n)
3 }
4 n_values <- c(5, 100, 10000)
5 sapply(n_values, pr_in_bootstrap)
6 # For n=5: 0.67232
7 # For n=100: 0.63397
8 # For n=10000: 0.63214
9 # Converges to 1 - exp(-1)
10 # 1 - exp(-1) # approx 0.63212

```

2.6.9 Exercise: ISLR 5.9 (Bootstrap for Boston data)

(Corresponds to ‘ch5-applied.R’ Exercise 9)

- (a) Estimate $\hat{\mu} = \text{mean}(\text{medv})$.
- (b) Estimate $SE(\hat{\mu})$ using formula s/\sqrt{n} .

Listing 24: Mean and SE of Mean for Bostonmedv

```

1 # library(MASS)
2 # data(Boston)
3 # mu_hat_medv <- mean(Boston$medv) # 22.53281
4 # se_mu_hat_formula <- sd(Boston$medv) / sqrt(nrow(Boston))
   # 0.4088611

```

- (c) Estimate $SE(\hat{\mu})$ using bootstrap.

Listing 25: Bootstrap SE of Mean for Bostonmedv

```

1 # library(boot)
2 # boot.fn_mean <- function(data_vector, index) {
3 #   return(mean(data_vector[index]))
4 # }
5 # set.seed(1)
6 # boot_mean_results <- boot(Boston$medv, boot.fn_mean, R
   =1000)
7 # print(boot_mean_results) # Bootstrap SE is ~0.4119 (close
   to formula)

```

- (d) Compare SEs and construct 95% CI for μ .
- Bootstrap SE is similar to formula-based SE.
- CI from `t.test(Boston$medv)()`: e.g., [21.73, 23.34].
- Bootstrap CI (e.g., percentile or BCa from `boot.ci()`) should be similar.
- (e) Estimate $\hat{\mu}_{med} = \text{median}(\text{medv})$. (Result: 21.2)
- (f) Estimate $SE(\hat{\mu}_{med})$ using bootstrap. (Result $SE \approx 0.38$, shown in earlier R example).
- (g) Estimate $\hat{\mu}_{0.1} = 10\text{th percentile of medv}$. (Result: 12.75)
- (h) Estimate $SE(\hat{\mu}_{0.1})$ using bootstrap.

Listing 26: Bootstrap SE for 10th Percentile of *Bostonmedv*

```

1 # boot.fn_q10 <- function(data_vector, index) {
2 #   return(quantile(data_vector[index], probs=0.1))
3 # }
4 # set.seed(1)
5 # boot_q10_results <- boot(Boston$medv, boot.fn_q10, R
6 #   =1000)
7 # print(boot_q10_results) # Bootstrap SE is ~0.5113

```

2.7 Lecture 7: Linear Model Selection Methods

2.7.1 Introduction to Model Selection

- Model: $Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$.
- **Conflicting Goals:**
 - *Prediction Accuracy*: We want a model that predicts well on new, unseen data.
 - *Model Interpretability*: We want a simpler model that is easier to understand and explain.
- **Challenge with many predictors (p large relative to n):**
 - Potentially more information about $Y \implies$ better predictions.
 - Estimating many parameters $\beta_j \implies$ high estimation uncertainty (variance) \implies poor predictions.
 - This leads to a trade-off:
 - *Prediction Bias*: Due to an incorrectly specified model (e.g., omitting important variables).
 - *Prediction Variance*: Due to estimating too many parameters from limited data.
- **Goal of Model Selection**: Find a model that optimally balances this bias-variance trade-off to achieve good prediction accuracy and/or interpretability.

2.7.2 Three Types of Model Selection Techniques

1. **Subset Selection**: Identify a subset of the p predictors that we believe are most relevant to the response. Then fit a model using OLS on this reduced set.
 - Examples: Best Subset Selection, Forward Stepwise, Backward Stepwise.
2. **Shrinkage (Regularization)**: Fit a model involving all p predictors, but the estimated coefficients are constrained or "shrunk" towards zero relative to the OLS estimates. This reduces variance.
 - Examples: Ridge Regression, Lasso. (Covered in Lecture 8)
3. **Dimension Reduction**: Project the p predictors into an M -dimensional subspace, where $M < p$. Then fit a linear regression model using these M projections as predictors.
 - Examples: Principal Components Regression (PCR), Partial Least Squares (PLS).

2.7.3 Best Subset Selection

- **Algorithm** (Slide L6 p.4):
 1. Let \mathcal{M}_0 be the null model (intercept only, no predictors): $Y = \beta_0 + \epsilon$.
 2. For $k = 1, 2, \dots, p$:
 3. Fit all $\binom{p}{k}$ models that contain exactly k predictors.
 4. Pick the best among these $\binom{p}{k}$ models based on some criterion (e.g., highest R^2 or lowest RSS for a fixed k). Call this model \mathcal{M}_k .
 5. Select the single best model from $\mathcal{M}_0, \mathcal{M}_1, \dots, \mathcal{M}_p$ using a criterion that accounts for model complexity/overfitting (e.g., Adjusted R^2 , C_p , BIC, AIC, or cross-validated test MSE). Cannot use raw R^2 or RSS for this final step as they always improve with more predictors.
- **Computational Cost**: Involves fitting 2^p models. If $p = 50$, this is $> 10^{15}$ models, which is computationally infeasible. Generally feasible for $p \leq 30 - 40$.
- **R Example (ISLR Auto data)** (Slides L6 p.5-11, using $p = 6$ predictors):
 - Predictors: cylinders, displacement, horsepower, weight, acceleration, age (derived from year).
 - *Full Model* (Slide L6 p.6): `lm(mpg ~ ., data=Auto_modified).AdjustedR^2 = 0.8063`.
 - *Step 1: Fit \mathcal{M}_0* (Slide L6 p.7): `M0=lm(y ~ 1)`. RSS for \mathcal{M}_0 is TSS.
 - *Step 2: Find best model \mathcal{M}_k for each k* (Slide L6 p.8 for $k = 2$): For $k = 2$, the model with weight and age has the lowest RSS (4568.95) among all 2-predictor models.
 - *Step 3: Select overall best model* (Slide L6 p.9): Using Adjusted R^2 , the model with weight and age (\mathcal{M}_2) has the highest Adj $R^2 = 0.8072$.

Listing 27: Best Subset Selection with `leaps::regsubsets` (Slides L6 p.10-11)

```

1 # require(ISLR)
2 # Auto$age <- 83 - Auto$year # Create age variable
3 # Auto_subset <- Auto[, !(names(Auto) %in% c("name", "origin",
4   "year"))] # Select relevant columns
5
6 # require(leaps)
7 # regfit.full <- regsubsets(mpg ~ ., data=Auto_subset, nvmax
8   = 6) # nvmax = p
9 # reg.summary <- summary(regfit.full)
10 # print(reg.summary)
11 # names(reg.summary) # Shows "which", "rsq", "rss", "adjr2",
12   "cp", "bic"
13
14 # Results from summary (Slide L6 p.10)
15 # cbind(reg.summary$which[,-1], adjR2=round(reg.summary$adjr2
16   ,4))
17 # Shows which variables are in the best model of each size,
18   and its AdjR2.
19 # ##   cylinders displacement horsepower weight acceleration
20   age   adjR2
21 # ## 1      FALSE             FALSE      FALSE      TRUE      FALSE
22   FALSE 0.6918 (weight)
23 # ## 2      FALSE             FALSE      FALSE      TRUE      FALSE
24   TRUE 0.8072 (weight, age)

```

```

17 # ## 3      FALSE      FALSE      FALSE      TRUE      TRUE
    TRUE 0.8071 (weight, acc, age)
18 # ## ...
19
20 # plot(regfit.full, scale="adjr2", col=gray.colors(10)) #
    Visual (Slide L6 p.11)
21 # The plot shows that AdjR2 peaks for the model with 'weight'
    and 'age'.

```

- **Example with 3 predictors** (X_1, X_2, X_3) (Slides L6 p.18-19):
- $\mathcal{M}_0 : Y = \beta_0 + \epsilon$
- \mathcal{M}_1 : Best of $\{X_1\}, \{X_2\}, \{X_3\}$ (based on R^2)
- \mathcal{M}_2 : Best of $\{X_1, X_2\}, \{X_1, X_3\}, \{X_2, X_3\}$ (based on R^2)
- $\mathcal{M}_3 : Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \epsilon$
- Choose among $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ using Adj R^2 , C_p , BIC, or CV test MSE.
- Total models estimated: $2^3 = 8$. (Slide L6 p.20)

2.7.4 Stepwise Selection Methods

- Computationally cheaper alternatives to best subset selection, especially for large p .
- **Forward Stepwise Selection** (Slides L6 p.12):
 1. Start with null model \mathcal{M}_0 (intercept only).
 2. For $k = 0, \dots, p - 1$:
 3. Consider all $p - k$ models that add one additional predictor to \mathcal{M}_k .
 4. Choose the best among these $p - k$ models (e.g., one with lowest RSS or highest R^2). Call this \mathcal{M}_{k+1} .
 5. Select the best model among $\mathcal{M}_0, \dots, \mathcal{M}_p$ using C_p , BIC, Adj R^2 , or CV.
 6. Total models: $1 + \sum_{k=0}^{p-1} (p - k) = 1 + p(p + 1)/2$. For $p = 50$, this is $1 + 1275 = 1276$ models.
- **R Example (Forward Stepwise for Auto data)** (Slides L6 p.13-14):

Listing 28: Forward Stepwise with leaps::regsubsets (Slides L6 p.13)

```

1 # regfit.fwd <- regsubsets(mpg ~ ., data=Auto_subset, nvmax =
    6, method="forward")
2 # summary.fwd <- summary(regfit.fwd)
3 # cbind(summary.fwd$which[,-1], adjR2=round(summary.fwd$adjr2
    ,4))
4 # # Results are identical to Best Subset for this Auto
    example
5 # plot(regfit.fwd, scale="adjr2", col=gray.colors(10))
6 # Number of models estimated: 1 (null) + 6+5+4+3+2+1 = 22.
    Slide says 21, likely not counting M0 in sum.

```

- **Backward Stepwise Selection** (Slides L6 p.15):
 1. Start with full model \mathcal{M}_p (all p predictors).
 2. For $k = p, p - 1, \dots, 1$:
 3. Consider all k models that remove one predictor from \mathcal{M}_k .
 4. Choose the best among these k models (e.g., lowest RSS or highest R^2). Call this \mathcal{M}_{k-1} .

5. Select the best model among $\mathcal{M}_0, \dots, \mathcal{M}_p$ using C_p , BIC, Adj R^2 , or CV.
6. Total models: $1 + \sum_{k=1}^p k = 1 + p(p+1)/2$. Requires $n > p$ for initial full model.

· **R Example (Backward Stepwise for Auto data)** (Slides L6 p.16-17):

Listing 29: Backward Stepwise with leaps::regsubsets (Slides L6 p.16)

```

1 # regfit.bwd <- regsubsets(mpg ~ ., data=Auto_subset, nvmax =
   # 6, method="backward")
2 # summary.bwd <- summary(regfit.bwd)
3 # cbind(summary.bwd$which[,-1], adjR2=round(summary.bwd$adjr2
   # ,4))
4 # # Results are identical to Best Subset and Forward for this
   # Auto example
5 # plot(regfit.bwd, scale="adjr2", col=gray.colors(10))

```

- **Hybrid Approaches:** Combine forward and backward steps (e.g., add predictors then remove some if they become non-significant).
- Stepwise methods are not guaranteed to find the true best model out of all 2^p possibilities but are often good approximations and computationally efficient.

2.7.5 Choosing the Optimal Model (Criteria)

- After generating a sequence of models $\mathcal{M}_0, \dots, \mathcal{M}_p$ (either from best subset or stepwise), we need to select one.
- Training error (RSS, R^2) is not suitable as it always improves with more variables.
- We need to estimate test error. Two approaches:
 1. **Adjusting Training Error for Model Size:** Add a penalty for number of predictors (d).
 2. **Mallow's C_p** (Slide L6 p.21): For OLS with estimated $\hat{\sigma}^2$ from full model.

$$C_p = \frac{1}{n}(RSS_d + 2d\hat{\sigma}^2)$$

Choose model with smallest C_p . If $C_p \approx d$, model has low bias.

3. **Akaike Information Criterion (AIC)** (Slide L6 p.21): Proportional to

$$AIC \propto \frac{1}{n\hat{\sigma}^2}(RSS_d + 2d\hat{\sigma}^2)$$

(Derived from maximizing log-likelihood minus a penalty $2d$). Choose model with smallest AIC. For linear models with Gaussian errors, C_p and AIC are proportional.

4. **Bayesian Information Criterion (BIC)** (Slide L6 p.21):

$$BIC = \frac{1}{n}(RSS_d + \log(n)d\hat{\sigma}^2)$$

(For linear models with Gaussian errors, assuming $\hat{\sigma}^2$ is error variance). BIC penalizes model size more heavily than AIC/ C_p for $n > 7$. Tends to select smaller models. Choose model with smallest BIC.

5. **Adjusted R^2** (Slide L6 p.21):

$$R_{adj}^2 = 1 - \frac{RSS_d/(n - d - 1)}{TSS/(n - 1)}$$

Penalizes for adding useless predictors. Choose model with largest Adj R^2 .

6. **Direct Estimation of Test Error using Cross-Validation:**

7. For each model size $k = 0, \dots, p$, get the best k -variable model \mathcal{M}_k .
8. Compute its CV test error (e.g., 5-fold or 10-fold CV).
9. Select k that minimizes CV test error. This is often considered the most direct and reliable approach if computationally feasible.
10. "One-standard-error rule": Choose the simplest model whose CV error is within one standard error of the minimum CV error.

2.7.6 Exercise: ISLR 6.8

(Refers to 'ex6.8.R')

- (a) Generate data: $X \sim N(0, 1)$, $\epsilon \sim N(0, 1)$, $Y = \beta_0 + \beta_1 X + \dots + \beta_p X^p + \epsilon$. For this exercise, data is simulated for $p = 100$ predictors, $n = 1000$ obs, where some β_j are zero. $Y = X\beta + \epsilon$.
- (b) Split into train (100 obs) and test (900 obs).
- (c) Best subset selection on training data, plot training MSE vs. subset size. Training MSE will decrease monotonically.
- (d) Plot test MSE vs. subset size. Will show a U-shape, identifying optimal number of predictors for test performance.
- (e) Find best model size based on test MSE.
- (f) Compare coefficients of best model to true β s.
- (g) Plot error $\sqrt{\sum(\hat{\beta}_j - \beta_j)^2}$ vs. subset size. This measures how well coefficients are estimated. The minimum may not align with minimum test MSE.
- *Key R functions for ISLR 6.8: `rnorm()`, `matrix()`, `sample()`, `leaps::regsubsets()`, `coef()`, `predict.regsubsets()` (custom function usually needed for 'regsubsets' as it doesn't have a standard 'predict' method for new data directly).*

Listing 30: Conceptual predict function for regsubsets (ISLR Ch6 Lab)

```
1 # predict.regsubsets <- function(object, newdata, id, ...) {
2 #   form <- as.formula(object$call[[2]]) # Get formula from
   regsubsets object
3 #   mat <- model.matrix(form, newdata)   # Create model
   matrix for new data
4 #   coefi <- coef(object, id=id)         # Get coefficients
   for model of size 'id'
5 #   xvars <- names(coefi)
6 #   pred <- mat[, xvars] %*% coefi      # Matrix
   multiplication
7 #   return(pred)
8 # }
```

2.8 Lecture 8: Regularization (Shrinkage) Methods and PCR

2.8.1 Introduction to Shrinkage Methods

- Recall: Subset selection methods select a subset of predictors and fit OLS.
- **Shrinkage Methods:** Use all p predictors, but constrain or "shrink" the estimated coefficients $\hat{\beta}_j$ towards zero (or each other).
- **Goal:** Reduce variance at the cost of a small increase in bias, potentially leading to better prediction accuracy (lower test MSE).
- **OLS objective:** Minimize $RSS(\boldsymbol{\beta}) = \sum_{i=1}^n (y_i - \mathbf{x}_i' \boldsymbol{\beta})^2$.
- **Shrinkage methods add a penalty term** to the RSS:
- Ridge Regression: Minimize $RSS(\boldsymbol{\beta}) + \lambda \sum_{j=1}^p \beta_j^2$.
- LASSO: Minimize $RSS(\boldsymbol{\beta}) + \lambda \sum_{j=1}^p |\beta_j|$.
- The penalty term discourages large coefficient values.

2.8.2 Ridge Regression

- **Objective Function** (Slide L7 p.3):

$$\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p \beta_j^2 \right\}$$

The term $\lambda \sum_{j=1}^p \beta_j^2$ is the L_2 penalty or shrinkage penalty.

- **Tuning Parameter** $\lambda \geq 0$ (Slide L7 p.3):
- Controls the amount of shrinkage.
- $\lambda = 0$: Penalty has no effect. Ridge estimates = OLS estimates.
- $\lambda \rightarrow \infty$: Shrinkage penalty dominates. Coefficients $\hat{\beta}_j^{\text{ridge}} \rightarrow 0$ (for $j = 1, \dots, p$). Model approaches intercept-only model.
- β_0 (intercept) is typically not penalized.
- **Equivalent Formulation** (Constrained optimization, Slide L7 p.4):

$$\min_{\boldsymbol{\beta}} \left\{ \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^p \beta_j^2 \leq s$$

There is a one-to-one correspondence between λ and s .

- **Standardization of Predictors** (Slides L7 p.4-5):
- OLS estimates are scale equivariant (if X_j is multiplied by c , $\hat{\beta}_j^{OLS}$ is divided by c).
- Ridge regression estimates are *not* scale equivariant because the penalty $\lambda \sum \beta_j^2$ treats all β_j equally, regardless of the scale of X_j .
- **Crucial:** Standardize predictors to have mean 0 and standard deviation 1 before fitting Ridge:

$$\tilde{x}_{ij} = \frac{x_{ij} - \bar{x}_j}{s_j}$$

(Slide L7 p.5 shows $\tilde{x}_{ij} = x_{ij} / \sqrt{\frac{1}{n} \sum (x_{ij} - \bar{x})^2}$, which is scaling to have RMS of 1, effectively standardizing if mean is 0. Standard practice is to scale to unit variance.)

- **Effect on Bias and Variance** (Slide L7 p.3):
- Ridge regression reduces variance compared to OLS, especially when predictors are collinear.
- It introduces some bias (coefficients are shrunk from OLS values).
- Can lead to lower Test MSE if the reduction in variance outweighs the increase in squared bias.
- **R Implementation (glmnet package)** (Slides L7 p.9-12):
- `glmnet()` function. Set `alpha=0` for Ridge.
- Requires predictors `X` as a matrix and response `y` as a vector.
- `cv.glmnet()` performs cross-validation to select optimal λ .

Listing 31: Ridge Regression Example (Simulated Data, Slides L7 p.6-12)

```

1 # Simulated data from slides L7 p.6
2 # set.seed(1234567)
3 # n <- 100
4 # p_vars <- 10
5 # X_sim <- matrix(0, n, p_vars)
6 # for(j in 1:p_vars) X_sim[,j] <- rnorm(n, sd=j) # Predictors
   with increasing variance
7 # beta_sim <- 2:(p_vars+1)
8 # e_sim <- rnorm(n, sd=200)
9 # y_sim <- 1 + X_sim %% beta_sim + e_sim
10
11 # OLS fit (Slide L7 p.8)
12 # ols.fit <- lm(y_sim ~ X_sim)
13 # summary(ols.fit) # Shows some coeffs not significant, high
   RSE
14
15 library(glmnet)
16 # Ridge with small lambda (Slide L7 p.9)
17 # ridge.fit_small_lambda <- glmnet(X_sim, y_sim, alpha=0,
   lambda=0.01)
18 # cbind(coef(ols.fit), coef(ridge.fit_small_lambda)) # Coeffs
   very similar
19
20 # Ridge with large lambda (Slide L7 p.10)
21 # ridge.fit_large_lambda <- glmnet(X_sim, y_sim, alpha=0,
   lambda=1000000)
22 # cbind(coef(ols.fit), coef(ridge.fit_large_lambda)) # Ridge
   coeffs shrunk close to 0
23
24 # Choosing lambda by L00CV (Slide L7 p.11)
25 # cv.ridge <- cv.glmnet(X_sim, y_sim, alpha=0, nfolds=n) #
   nfolds=n for L00CV
26 # lambda_min_ridge <- cv.ridge$lambda.min # e.g., 28.27952
   from slide
27 # ridgemin.fit <- glmnet(X_sim, y_sim, alpha=0, lambda=lambda
   _min_ridge)
28 # cbind(coef(ols.fit), coef(ridgemin.fit)) # Shows shrunk
   coefficients
29
30 # Test predictions (Slide L7 p.12)
31 # # Assume xy_df created, split into train_df, test_df,
   Xtrain, ytrain, Xtest, ytest
32 # olstrain.fit <- lm(y ~ ., data=train_df) # Using original
   formula style with data.frame
33 # predols_test <- predict(olstrain.fit, newdata=test_df)

```

```

34 # ols_MSE_test <- mean((ytest - predols_test)^2) # e.g.,
    53559.83
35
36 # ridgetrain.fit <- glmnet(Xtrain, ytrain, alpha=0, lambda=
    lambda_min_ridge)
37 # predridge_test <- predict(ridgetrain.fit, newx=Xtest)
38 # ridge_MSE_test <- mean((ytest - predridge_test)^2) # e.g.,
    52444.16 (better than OLS)

```

2.8.3 The Lasso (Least Absolute Shrinkage and Selection Operator)

- **Objective Function** (Slide L7 p.13):

$$\min_{\beta} \left\{ \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 + \lambda \sum_{j=1}^p |\beta_j| \right\}$$

The term $\lambda \sum_{j=1}^p |\beta_j|$ is the L_1 penalty.

- **Key Property:** Unlike Ridge, the L_1 penalty can force some coefficient estimates to be *exactly zero* if λ is large enough.
- This means Lasso performs **variable selection**.
- **Tuning Parameter** $\lambda \geq 0$ (Slide L7 p.13):
- $\lambda = 0$: Lasso estimates = OLS estimates.
- $\lambda \rightarrow \infty$: All coefficients $\hat{\beta}_j^{\text{lasso}} \rightarrow 0$.
- As λ increases, more coefficients are set to zero.
- **Equivalent Formulation** (Constrained optimization):

$$\min_{\beta} \left\{ \sum_{i=1}^n \left(y_i - \beta_0 - \sum_{j=1}^p \beta_j x_{ij} \right)^2 \right\} \quad \text{subject to} \quad \sum_{j=1}^p |\beta_j| \leq s$$

- Like Ridge, predictors should be standardized before applying Lasso.
- **R Implementation (glmnet package)** (Slides L7 p.14-15):
- Use `glmnet()` with `alpha=1`.
- `cv.glmnet()` for choosing λ .

Listing 32: Lasso Example (Simulated Data, Slides L7 p.14-15)

```

1 # Using X_sim, y_sim from Ridge example
2 # cv.lasso <- cv.glmnet(X_sim, y_sim, alpha=1, nfolds=n) #
    nfolds=n for LOOCV
3 # lambda_min_lasso <- cv.lasso$lambda.min # e.g., 13.12619
    from slide
4 # lassomin.fit <- glmnet(X_sim, y_sim, alpha=1, lambda=lambda
    _min_lasso)
5 # cbind(coef(ols.fit), coef(ridgemin.fit), coef(lassomin.fit)
    )
6 # # Lasso sets X1, X3, X4 to zero (coefficients are '.') in
    slide example
7
8 # Test predictions (Slide L7 p.15)
9 # # Using Xtrain, ytrain, Xtest, ytest from Ridge example
10 # lassotrain.fit <- glmnet(Xtrain, ytrain, alpha=1, lambda=
    lambda_min_lasso)
11 # predlasso_test <- predict(lassotrain.fit, newx=Xtest)
12 # lasso_MSE_test <- mean((ytest - predlasso_test)^2) # e.g.,
    51258.12 (best of OLS, Ridge, Lasso)

```

2.8.4 Comparing Lasso and Ridge

- **Variable Selection:** Lasso can produce sparse models (some $\beta_j = 0$), Ridge cannot (all β_j are non-zero unless $\lambda = \infty$).
- **Performance:**
 - If many predictors have small/moderate effects, Ridge might perform better (keeps all variables).
 - If a smaller number of predictors have substantial effects and others are negligible, Lasso might perform better (by setting negligible ones to zero).
- **Geometric Interpretation** (ISLR Fig 6.7):
 - Ridge constraint $\sum \beta_j^2 \leq s$ is a circle (2D) or hypersphere. OLS solution contours (ellipses) are less likely to hit an axis exactly.
 - Lasso constraint $\sum |\beta_j| \leq s$ is a diamond (2D) or rhomboid. OLS solution contours are more likely to hit a corner, making a coefficient zero.
- **Elastic Net:** A compromise between Ridge and Lasso, penalty $\alpha \sum |\beta_j| + (1 - \alpha) \sum \beta_j^2$. (`glmnet()` can fit this by setting $0 < \alpha < 1$).

2.8.5 Selecting the Tuning Parameter λ

- Cross-validation (typically k-fold, e.g., 10-fold) is the standard method.
- For a grid of λ values:
 1. For each λ , perform k-fold CV.
 2. Calculate average test MSE (or other error metric) across folds for that λ .
 3. Choose λ that results in the lowest average CV test error (`lambda.min` from `cv.glmnet()`).
 4. Often, `lambda.1se` is also considered: largest λ such that error is within one standard error of the minimum. This gives a more parsimonious model with similar performance.

2.8.6 Dimension Reduction Methods

- Transform predictors X_1, \dots, X_p into a new set of $M < p$ predictors Z_1, \dots, Z_M . Then fit OLS using Z_m .
- $Z_m = \sum_{j=1}^p \phi_{jm} X_j$ (linear combinations). (Slide L7 p.16)
- Two main methods: Principal Components Regression (PCR) and Partial Least Squares (PLS).

2.8.7 Principal Components Analysis (PCA) and Regression (PCR)

- **Principal Component Analysis (PCA)** (Slides L7 p.17-18):
 - A technique for dimension reduction that finds directions (principal components, PCs) in the data that capture the most variance.
 - Z_1 (first PC): Linear combination of X_j s that has maximal variance.
 - Z_2 (second PC): Linear combination of X_j s, uncorrelated with Z_1 , that has maximal variance among all such combinations.
 - And so on, up to p PCs.
 - The ϕ_{jm} (loadings) define the components.

- Predictors should be standardized before PCA if on different scales.
- **Principal Components Regression (PCR):**
 1. Perform PCA on the p predictors to get Z_1, \dots, Z_p .
 2. Select the first $M < p$ principal components. M is a tuning parameter chosen by CV.
 3. Fit OLS regression of Y on Z_1, \dots, Z_M .
- PCA is an unsupervised method (does not use Y to find components). PCR becomes supervised when Y is regressed on Z_m .
- PCR can reduce variance by using fewer components, especially if early PCs capture most of the signal related to Y .
- If most variation in predictors can be seen in a few PCs, we reduce overfitting. (Slide L7 p.18)
- **R Implementation (pls package)** (Slides L7 p.19-21):

Listing 33: PCR Example (Auto Data, Slides L7 p.19, 21)

```

1 library(pls)
2 library(ISLR)
3 # Auto2 <- Auto[,!(names(Auto) %in% c("name","origin","year"))
  # From slide L6 p.6
4 # Auto2$age <- 83 - Auto$year # From slide L6 p.6
5 # set.seed(1) # Assuming a seed for train/test split
6 # n_auto <- nrow(Auto2)
7 # train_idx_auto <- sample(1:n_auto, floor(n_auto/2))
8 # train_auto <- Auto2[train_idx_auto,]
9 # test_auto <- Auto2[-train_idx_auto,]
10
11 # Fit PCR on training data, choose M using CV (built-in)
12 # pcr.fit <- pcr(mpg ~ ., data=train_auto, scale=TRUE,
  validation="CV")
13 # summary(pcr.fit) # Shows % variance explained by X and mpg
  for different M
14 # validationplot(pcr.fit, val.type="MSEP") # Helps choose M
15
16 # Example output from summary(pcr.fit) on full Auto2 (Slide
  L7 p.19):
17 # TRAINING: % variance explained
18 #      1 comps 2 comps 3 comps 4 comps 5 comps 6 comps
19 # X      99.76  99.96 100.00 100.00 100.00 100.00
20 # mpg     69.35  70.09  70.75  80.79  80.88  80.93
21 # First PC explains 99.76% of variance in X's, and 69.35% of
  variance in mpg.
22
23 # Training/test evaluation (Slide L7 p.21 - assuming M=4
  chosen from CV)
24 # regall.fit <- lm(mpg ~ ., data=train_auto)
25 # pcr.fit_M4 <- pcr(mpg ~ ., ncomp=4, data=train_auto, scale=
  TRUE)
26 #
27 # predall_test <- predict(regall.fit, newdata=test_auto)
28 # MSEall_test <- mean((test_auto$mpg - predall_test)^2) # e.g
  ., 11.47
29 #
30 # predpcr_test <- predict(pcr.fit_M4, newdata=test_auto)
31 # MSEpcr_test <- mean((test_auto$mpg - predpcr_test)^2) # e.g
  ., 15.60
32 # In this specific split on slide L7, OLS performed better
  than PCR with M=4.

```

2.8.8 Partial Least Squares (PLS)

- Similar to PCR, but finds components Z_m in a supervised way.
- $Z_1 = \sum \phi_{j1} X_j$ where ϕ_{j1} are coefficients of simple linear regression of Y on each X_j . Z_1 is the linear combination most correlated with Y .
- Z_2 is found by first regressing Y on Z_1 and taking residuals. Then Z_2 is the linear combination of X_j s (after orthogonalizing them w.r.t Z_1) that best explains these residuals.
- Can sometimes outperform PCR, especially if predictors are highly collinear or if the response is strongly related to directions with lower variance in X .
- Number of components M chosen by CV.
- R: `plsr()` in `pls` package.

2.8.9 Exercise: ISLR 6.11

(Refers to 'ex6.11.R')

- Goal: Compare Best Subset, Lasso, Ridge, PCR for predicting per capita crime rate (`crim`).
- Setup: Split `Boston` data into training/test. Prepare `X` matrix and `y` vector.
- (a) Best Subset Selection:
 - Use k-fold CV (e.g., 10-fold) to select optimal number of predictors.
 - Need a 'predict' method for 'regsubsets' (often custom written).
 - Fit best model of chosen size on full training set, evaluate on test set.
 - From 'ex6.11.R': 10-fold CV suggests 9 predictors. Test RMSE ≈ 6.59 .
- (a) Lasso:
 - Use `cv.glmnet()` on training data to find optimal λ .
 - Fit Lasso model with optimal λ on training data.
 - Predict on test data, calculate test RMSE.
 - From 'ex6.11.R': Test RMSE (using λ_{1se}) ≈ 7.405 .
- (a) Ridge Regression:
 - Use `cv.glmnet()` (`alpha=0`) on training data for λ .
 - Fit Ridge model with optimal λ on training data.
 - Predict on test data, calculate test RMSE.
 - From 'ex6.11.R': Test RMSE (using λ_{1se}) ≈ 7.457 .
- (a) PCR:
 - Use `pcr()` with `validation="CV"` on training data to choose number of components M .
 - Fit PCR model with chosen M on training data.
 - Predict on test data, calculate test RMSE.
 - From 'ex6.11.R': CV suggests $M = 13$ components (full model). Test RMSE ≈ 6.546 .
- (b) Conclusion: Compare test RMSEs. Best subset and PCR (with many components) performed best in this example.
- (c) Are models parsimonious?: Discuss which methods yield simpler models (Lasso, Best Subset if few variables chosen).

2.9 Lecture 9: Non-linear Models - Polynomials, Splines, Local Regression

2.9.1 Introduction to Non-linear Models

- Linear models assume a linear relationship between predictors and response, which is often too simplistic.
- This lecture explores methods to model non-linear relationships:
- Polynomial Regression
- Step Functions (Piecewise Constant Regression)
- Regression Splines (Piecewise Polynomials)
- Local Regression
- Smoothing Splines (covered briefly, more detail in GAMs)
- Generalized Additive Models (GAMs) provide a framework to use these for multiple predictors (covered in Lecture 10).

2.9.2 Polynomial Regression

- Extends linear model by adding polynomial terms of predictors:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \cdots + \beta_d x_i^d + \epsilon_i$$

- This is still a linear model in terms of coefficients β_j , so it can be fit using OLS.
- Choice of degree d is critical. Too low $d \implies$ underfitting. Too high $d \implies$ overfitting (wiggly fit).
- d often chosen by cross-validation or hypothesis tests (e.g., ANOVA to compare nested models).
- R: `lm(y ~ poly(x, degree=d))()` or `lm(y ~ x + I(x^2) + ...)()`. Using `poly()` creates orthogonal polynomials, which can be numerically more stable.

2.9.3 Step Functions (Piecewise Constant Regression)

- Divide the range of X into $K + 1$ disjoint regions using K cutpoints (knots) $c_1 < c_2 < \cdots < c_K$.
- Fit a constant (mean of Y) in each region:

$$y_i = \beta_0 + \beta_1 I(x_i < c_1) + \beta_2 I(c_1 \leq x_i < c_2) + \cdots + \beta_{K+1} I(x_i \geq c_K) + \epsilon_i$$

(Slide L8 p.3 shows a slightly different formulation with 5 coefficients for 4 knots, creating 5 regions).

- This is achieved by creating dummy variables for each interval.
- Choice of number and location of knots is important. Often chosen by CV or domain knowledge.
- R Example (Simulated data, Slides L8 p.3-7):

Listing 34: Piecewise Constant Regression (Step Function) in R

```
1 # Simulating data (Slides L8 p.3)
2 # set.seed(1); n=100; x=runif(n)
3 # c_knots=seq(0.10,0.90,length=4) # c1, c2, c3, c4
```

```

4 # b_coeffs=c(1,2,3,-1,1) # Beta for intercept and each
   interval
5 # y_step = b_coeffs[1] +
6 #         b_coeffs[2](x < c_knots[1]) +
7 #         b_coeffs[3]((x >= c_knots[1]) & x < c_knots[2]) +
8 #         b_coeffs[4]((x >= c_knots[2]) & x < c_knots[3]) +
   # Error in slide formula for b[4]
9 #         b_coeffs[5]((x >= c_knots[3]) & x < c_knots[4]) +
   # Error in slide formula for b[5]
10 #         0.1*norm(n) # Should be 5 intervals for 5 beta
   coeffs after intercept if b[1] is overall mean
11 #         # Or one more interval for x >= c_knots
   [4] if b[1] is first interval
12 # A more direct way in R is using cut()
13 # fit_step <- lm(y ~ cut(x, breaks=c(min(x)-.1, c_knots, max(
   x)+.1)))
14 # summary(fit_step)
15 # The coefficients will be differences from the intercept (
   mean of first interval).
16
17 # Example with Wage data (Slides L8 p.8)
18 # library(ISLR)
19 # attach(Wage)
20 # table(cut(age, 4)) # Divides age into 4 equally spaced
   intervals
21 # fit_wage_step <- lm(wage ~ cut(age, 4), data=Wage)
22 # coef(summary(fit_wage_step))
23 # # Shows average wage for first age group, and differences
   for others.
24 # detach(Wage)

```

- Drawback: Fit is not continuous, can miss trends at boundaries.

2.9.4 Basis Functions

- General idea: Transform X using a set of known basis functions $b_1(X), b_2(X), \dots, b_K(X)$
- Then fit linear model: $y_i = \beta_0 + \beta_1 b_1(x_i) + \dots + \beta_K b_K(x_i) + \epsilon_i$.
- Examples:
 - Polynomial regression: $b_j(x) = x^j$.
 - Step functions: $b_j(x) = I(c_j \leq x < c_{j+1})$.
 - Periodic functions: $b_j(x) = \sin(\alpha_j x)$ or $\cos(\alpha_j x)$.

2.9.5 Regression Splines (Piecewise Polynomials)

- Fit separate low-degree polynomials in different regions defined by knots ξ_1, \dots, ξ_K .
- **Constraints for Smoothness** (Slide L8 p.11): To avoid discontinuities, constraints are imposed at the knots.
- Function must be continuous at knots.
- First derivative continuous (smooth curve).
- Second derivative continuous (smoother curve).
- A degree- d spline with K knots has $d + 1 + K$ degrees of freedom (parameters).
- **Cubic Spline** (degree 3) is common: Continuous, with continuous 1st and 2nd derivatives. Smooth and flexible.

- **Cubic Spline Basis Representation** (Slide L8 p.12): A cubic spline with K knots ξ_1, \dots, ξ_K can be modeled using a basis:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \sum_{k=1}^K \beta_{k+3} h(x_i, \xi_k) + \epsilon_i$$

where $h(x, \xi) = (x - \xi)_+^3 = \begin{cases} (x - \xi)^3 & \text{if } x > \xi \\ 0 & \text{otherwise} \end{cases}$. This model has $K + 4$ parameters (degrees of freedom = $K + 4$).

- **Choosing Knots and Number of Knots (or Degrees of Freedom):**
 - More knots \implies more flexible, can overfit.
 - Knots often placed at uniform quantiles of X or based on domain knowledge.
 - Number of knots (or equivalently, degrees of freedom ‘df’) chosen by cross-validation.
- **Natural Splines** (ISLR Ch 7.4.3): Cubic splines constrained to be linear beyond boundary knots. Reduces variance at boundaries. Uses K df for K knots.
- **R Implementation (splines package)** (Slide L8 p.13, using `bs()`):

Listing 35: Regression Spline with Wage Data (Slide L8 p.13)

```

1 # library(ISLR)
2 # library(splines) # For bs() function
3 # attach(Wage)
4 # # Fit a cubic spline with knots at ages 25, 40, 60
5 # fit.spline <- lm(wage ~ bs(age, knots=c(25,40,60)), data=
  Wage)
6 # summary(fit.spline) # Shows coefficients for basis
  functions
7 #
8 # # Plotting the fit
9 # agelims <- range(age)
10 # age.grid <- seq(from=agelims[1], to=agelims[2])
11 # pred.spline <- predict(fit.spline, newdata=list(age=age.
  grid))
12 # plot(age, wage, col="lightgrey")
13 # lines(age.grid, pred.spline, col="black", lwd=2)
14 # # Add vertical lines for knots
15 # abline(v=c(25,40,60), lty=2, col="red")
16 # detach(Wage)
17 # # Alternatively, specify degrees of freedom (df) instead of
  knots
18 # # fit.spline_df <- lm(wage ~ bs(age, df=6), data=Wage) # df
  =6 for 3 knots cubic spline (df=K+3+1-1=K+3 usually, or K
  +degree)
19 # # bs() uses df = K + degree (for cubic, df=K+3). Intercept
  is separate.
20 # # So, 3 knots with bs() is df=3+3=6.

```

2.9.6 Smoothing Splines

- Goal: Find a function $g(x)$ that fits the data well ($RSS = \sum (y_i - g(x_i))^2$ is small) but is also smooth.

- **Objective Function** (Slide L9 p.3):

$$\min_g \left\{ \sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int (g''(t))^2 dt \right\}$$

- First term: Measures closeness to data (RSS).
- Second term: Roughness penalty. $\int (g''(t))^2 dt$ is large if $g(x)$ is "wiggly".
- $\lambda \geq 0$ is a tuning parameter:
- $\lambda = 0$: $g(x)$ interpolates data (can be very rough, zero RSS if all x_i unique). (Slide L9 p.2 figure)
- $\lambda \rightarrow \infty$: $g''(t)$ must be zero, meaning $g(x)$ is a linear function (OLS line).
- **Solution**: The function $g(x)$ that minimizes this is a *natural cubic spline* with knots at every unique value of x_i . (Slide L9 p.3)
- Seems complex (many knots), but λ controls effective degrees of freedom (df_λ).
- λ is chosen by (generalized) cross-validation, often LOOCV.
- R: `smooth.spline(x, y, cv=TRUE)()` (Slide L9 p.5). `cv=TRUE` chooses λ by LOOCV.

Listing 36: Smoothing Spline with Wage Data (Slide L9 p.5)

```

1 # library(ISLR)
2 # attach(Wage)
3 # fit.smooth <- smooth.spline(age, wage, cv=TRUE) # Chooses
  lambda by LOOCV
4 # print(fit.smooth)
5 # # fit.smooth$df gives effective degrees of freedom, e.g.,
  6.79
6 #
7 # plot(age, wage, col="lightgrey")
8 # lines(fit.smooth, col="red", lwd=2) # Red line is CV-chosen
  smoothing spline
9 # # Can also specify df to get a specific lambda
10 # lines(smooth.spline(age, wage, df=16), col="blue", lwd=2) #
  More wiggly (higher df)
11 # detach(Wage)

```

2.9.7 Local Regression (LOESS/LOWESS)

- Fits separate simple models (e.g., linear or quadratic polynomials) in local neighborhoods of x_0 .
- **Algorithm at a target point x_0 :**
 1. Gather $s \cdot n$ training points closest to x_0 (the neighborhood). s is the span (e.g., 0.2 to 0.5).
 2. Assign weights to points in this neighborhood, with points closer to x_0 getting higher weights (e.g., tricube weight function). Points outside neighborhood get weight 0.
 3. Fit a weighted least squares regression of y on x using only points in the neighborhood with these weights.
 4. The fitted value \hat{y}_0 at x_0 is the prediction.
- This is repeated for all x_0 where a prediction is desired.

- Span s is the main tuning parameter, controls smoothness. Small $s \implies$ local, wiggly fit. Large $s \implies$ global, smoother fit.
- R: `loess(y ~ x, span=..., data=...)()`.

2.9.8 Kernel Estimators (Nadaraya-Watson)

- Model $y = f(x) + \epsilon$. (Slides L9 p.6)
- **Nadaraya-Watson Estimator**: A type of local averaging.

$$\hat{f}(x_0) = \sum_{i=1}^n w_i(x_0) y_i$$

where weights $w_i(x_0) = \frac{K(\frac{x_i - x_0}{h})}{\sum_{j=1}^n K(\frac{x_j - x_0}{h})}$ sum to 1.

- $K(\cdot)$ is a **kernel function**, a non-negative function that integrates to one and is symmetric about zero (e.g., Gaussian kernel, Epanechnikov kernel, boxcar kernel).
- Gaussian kernel (using standard normal density ϕ): $K(u) = \phi(u)$. (Slide L9 p.6 shows weights with ϕ).
- The plot on Slide L9 p.7 shows how $K((x - x_0)/h)$ gives higher weights to x_i close to x_0 .
- h is the **bandwidth**, a tuning parameter controlling smoothness. Small $h \implies$ local, wiggly. Large $h \implies$ global, smooth. Chosen by CV.
- R Example (Slides L8 p.19, implementing N-W with Gaussian kernel):

Listing 37: Nadaraya-Watson with Gaussian Kernel (Slides L8 p.19)

```

1 # K_gauss <- function(x0_val, x_vec, h_val=0.2) {
2 #   u <- (x_vec - x0_val) / h_val
3 #   weights_numerator <- dnorm(u) # Using standard normal
  density
4 #   return(weights_numerator / sum(weights_numerator))
5 # }
6 #
7 # nw_estimator <- function(x0_val, x_train, y_train, h_val
  =0.2) {
8 #   weights <- K_gauss(x0_val, x_train, h_val)
9 #   return(sum(weights * y_train))
10 # }
11 #
12 # # Example data from slides L8 p.16
13 # # set.seed(1); x_sim_nw <- rnorm(100, sd=20); y_sim_nw <- x
  _sim_nw2 + rnorm(100, sd=50)
14 # # plot(x_sim_nw, y_sim_nw)
15 # # ypred_knn_nw <- sapply(as.matrix(x_sim_nw), knn, x=x_sim_
  nw, y=y_sim_nw) # Using knn from slide L8 p.17
16 # # lines(sort(x_sim_nw), ypred_knn_nw[order(x_sim_nw)], col
  ="blue")
17 # # ypred_nw_h5 <- sapply(as.matrix(x_sim_nw), nw_estimator,
18 # #                       x_train=x_sim_nw, y_train=y_sim_nw,
  h_val=5)
19 # # lines(sort(x_sim_nw), ypred_nw_h5[order(x_sim_nw)], col="
  red")

```

- **Local Linear Regression** (Slides L8 p.28): Can be seen as an extension of Nadaraya-Watson. Instead of local constant (average), fit a local linear model using weighted least squares, with weights from a kernel.

$$\min_{\beta_0, \beta_1} \sum_{i=1}^n K\left(\frac{x_i - x_0}{h}\right) (y_i - \beta_0 - \beta_1(x_i - x_0))^2$$

Then $\hat{f}(x_0) = \hat{\beta}_0$. (Slide formula has $\beta_1 x_i$, often $(x_i - x_0)$ is used for stability). LOESS is a form of local polynomial regression.

2.9.9 Exercises: ISLR 7.6, 7.9, (7.10, 7.11 for GAMs)

- **ISLR 7.6 (Wage data)**: (Corresponds to ‘ex7.6.R’)
- (a) Polynomial regression for **wage** **age**: Use CV to choose degree. ANOVA for comparison. Plot results. *Result from ‘ex7.6.R’*: CV suggests degree 3 or 4 (varies with seed/method). ANOVA often suggests degree 3 or 4 as significant improvements over lower degrees.
- (b) Step function for **wage** **age**: Use CV to choose number of cuts. Plot. *Result from ‘ex7.6.R’*: CV often suggests around 7-9 cuts.
- **ISLR 7.9 (Boston data)**: (Corresponds to ‘ex7.9.R’)
- (a) Polynomial regression for **nox** **dis**: Plot polynomial fits of various degrees.
- (b) Report training RSS for different degrees. (Decreases monotonically).
- (c) Use CV to select best degree. Plot CV error. (Often degree 3 or 4 is chosen).
- (d) Regression spline with **bs()** for **nox** **dis**: Specify knots (e.g., at quartiles of ‘dis’). Report fit. Plot.
- (e) Regression spline with varying df: Fit for df from 3 to 16. Report training RSS. (Decreases).
- (f) Use CV to select best df for spline. Plot CV error.

2.10 Lecture 10: Generalized Additive Models (GAMs)

2.10.1 Introduction to GAMs

- Extends linear models to allow non-linear functions for *each* predictor, while maintaining an additive structure.
- Model form:

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

or, for expected value: $E(Y|X_1, \dots, X_p) = \beta_0 + \sum_{j=1}^p f_j(X_j)$.

- Each $f_j(x_{ij})$ can be a smooth non-linear function (e.g., spline, local polynomial) or a linear term.
- **Additive**: Allows examining the effect of each X_j on Y individually, holding other predictors constant, by plotting $\hat{f}_j(x_j)$.
- Avoids the "curse of dimensionality" faced by fitting a general non-linear function $f(x_1, \dots, x_p)$ directly in high dimensions.

2.10.2 Fitting GAMs: Backfitting Algorithm

- An iterative procedure to fit GAMs.
- **Example with two predictors** ($y = f_1(x_1) + f_2(x_2) + \epsilon$) (Slide L9 p.9-10):
 1. Initialize: e.g., $\hat{\beta}_0 = \bar{y}$, $\hat{f}_2(x_{i2}) = 0$ for all i .
 2. Iterate until convergence:
 3. Estimate \hat{f}_1 by fitting a model to $y_i - \hat{\beta}_0 - \hat{f}_2(x_{i2})$ using x_{i1} (e.g., smoothing spline on x_1).
 4. Estimate \hat{f}_2 by fitting a model to $y_i - \hat{\beta}_0 - \hat{f}_1(x_{i1})$ using x_{i2} (e.g., smoothing spline on x_2).
 5. (Often, residuals are centered).
 6. This means iteratively fitting each f_j on "partial residuals" $y_i - \hat{\beta}_0 - \sum_{k \neq j} \hat{f}_k(x_{ik})$.
- R: The `gam` package (or `mgcv`) handles this.

2.10.3 GAMs for Regression

- R Example (ISLR Wage data, using `gam` package):

Listing 38: GAM for Regression with Wage Data (Slide L9 p.11-12)

```
1 # library(ISLR)
2 # library(gam) # Older gam package. mgcv is more modern.
3 # attach(Wage)
4
5 # GAM with smoothing splines for year and age (df specified)
6 # gam.fit1 <- gam(wage ~ s(year, df=3) + s(age, df=3), data=
7 #   Wage) # Slide uses df=3 for both
8 # plot.Gam(gam.fit1, terms="s(age, 3)", col="red", se=TRUE) #
9 #   Slide L9 p.11, example with one term
10 # par(mfrow=c(1,3)) # To plot multiple terms (Slide L9 p.12)
11
12 # GAM including a linear term for education and smoothing
13 #   splines for year & age
14 gam.fit2 <- gam(wage ~ s(year, df=4) + s(age, df=5) +
15 #   education, data=Wage)
16 # Note: df might be different in slide examples, often chosen
17 #   by CV or set.
18 # plot.Gam(gam.fit2, se=TRUE, col="red") # Plots all terms
19 # par(mfrow=c(1,3)); plot(gam.fit2, se=TRUE, col="red") #
20 #   Example from ISLR book for Fig 7.12
21 # (Slide L9 p.12 shows plots for s(year), s(age), education.
22 #   s() implies smoothing spline)
23 # The plots show the estimated non-linear effect of year and
24 #   age, and linear effect of education.
25 # detach(Wage)
```

- For factor predictors like 'education', `gam()` fits a separate constant for each level (similar to linear regression with dummy variables).

2.10.4 GAMs for Classification

- Extends logistic regression by allowing non-linear functions for predictors.
- Model for binary response $Y \in \{0, 1\}$ (Slide L9 p.13):

$$\log \left(\frac{P(Y = 1|X)}{1 - P(Y = 1|X)} \right) = \beta_0 + f_1(x_1) + \cdots + f_p(x_p)$$

- Each f_j can be a smooth function.
- Fitted using backfitting with weighted least squares for each step (as logistic regression is fit with IRLS).
- R Example (Wage data, predict if `wage > 250`, Slide L9 p.14):

Listing 39: GAM for Classification (Slide L9 p.14)

```

1 # library(ISLR)
2 # library(gam)
3 # attach(Wage)
4 # gam.log.fit <- gam(I(wage > 250) ~ s(year, df=4) + s(age,
5 #                               df=5) + education,
6 #                               data=Wage, family=binomial)
7 # # par(mfrow=c(1,3))
8 # # plot.Gam(gam.log.fit, se=TRUE, col="red")
9 # # Shows effect of each predictor on the log-odds of wage >
  250.
# detach(Wage)

```

2.10.5 Pros and Cons of GAMs

- **Pros:**
 - Automatically model non-linear relationships without manual specification of transformations.
 - Additive nature allows examining effect of each predictor individually.
 - More flexible than linear models, but more interpretable than fully non-parametric methods like KNN in high dimensions.
 - Can improve predictive accuracy if true relationships are non-linear.
- **Cons:**
 - Primarily additive, so important interactions can be missed unless explicitly included (e.g., ‘`s(x1,x2)`’ or ‘`te(x1,x2)`’ in `mgcv`).
 - With many predictors, interpretation of many individual non-linear plots can still be complex.

2.10.6 Exercises: ISLR 7.10, 7.11

- **ISLR 7.10 (College data):** (Corresponds to ‘ex7.10.R’)
- (a) Forward stepwise selection for `Outstate` .. Identify best model by BIC/Cp/AdjR2. (E.g., 6 variables: Private, Room.Board, Terminal, perc.alumni, Expend, Grad.Rate).
- (b) Fit GAM with selected variables, using smoothing splines (`s()`) for continuous predictors. Plot terms.
- (c) Evaluate GAM on test set. Compare MSE with linear model using same predictors.
- (d) Check for non-linear evidence using Anova for GAM terms (‘`summary(gam.fit)`’ in `gam` provides this for non-parametric effects).
- *Result from ‘ex7.10.R’:* GAM Test MSE can be slightly better than OLS. Anova often shows evidence of non-linearity for some predictors like ‘Expend’.
- **ISLR 7.11 (Backfitting Algorithm):** (Corresponds to ‘ex7.11.R’)
- (a) Simulate data: $Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \epsilon$.

- (b, c, d, e) Implement backfitting algorithm manually: Initialize $\hat{\beta}_1$. Iterate:
 1. $a_i = y_i - \hat{\beta}_1 x_{i1}$. Regress a_i on X_2 to get $\hat{\beta}_{2,\text{iter}}$.
 2. $a_i = y_i - \hat{\beta}_{2,\text{iter}} x_{i2}$. Regress a_i on X_1 to get $\hat{\beta}_{1,\text{iter}+1}$. Repeat until convergence. Plot $\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2$ vs. iteration.
- (f) Compare with OLS coefficients from `lm(Y ~ X1 + X2)()`. They should match.
- (g) How many iterations? Typically very few for linear case.
- *'ex7.11.R' shows code for this manual backfitting.*

2.11 Lecture 11: Tree-Based Methods

2.11.1 Introduction to Decision Trees

- **Core Idea:** Segment the predictor space into a number of simple, non-overlapping regions. For every observation that falls into a region R_m , make the same prediction.
- **Regression Trees:** Prediction is the mean of response values for training observations in R_m .
- **Classification Trees:** Prediction is the most common class (mode) for training observations in R_m .
- Example (Slides L10 p.2-3, ISLR Hitters data, predict `logSalary` from `Years` and `Hits`):
 - First split: `Years < 4.5`.
 - If `Years < 4.5`, predict mean `logSalary` for this group (e.g., 5.11).
 - If `Years >= 4.5`, then split further by `Hits < 117.5`.
 - If `Hits < 117.5`, predict mean (e.g., 6.00).
 - If `Hits >= 117.5`, predict mean (e.g., 6.74).
 - These splits define regions R_1, R_2, R_3 .

2.11.2 Building Regression Trees

- **Stratifying Predictor Space:**
 1. Divide predictor space (X_1, \dots, X_p) into J non-overlapping rectangular regions R_1, \dots, R_J .
 2. For any observation in region R_j , predict $\hat{y}_{R_j} = \text{mean}(y_i | x_i \in R_j)$.
 3. Goal: Find regions that minimize RSS:

$$RSS = \sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

- **Recursive Binary Splitting** (Greedy Algorithm, Slides L10 p.7-8, 14):
 - Computationally infeasible to consider all possible partitions.
 - Start with all data in one region.
 - Iteratively split existing regions. At each step, choose the predictor X_j and cutpoint s that lead to the greatest possible reduction in RSS when splitting a region into $\{X | X_j < s\}$ and $\{X | X_j \geq s\}$.
 - This is a top-down, greedy approach (best split at current step, may not lead to global optimum).

- The two new regions are then split further using the same principle.
- Continue until a stopping criterion is met (e.g., minimum number of observations per region/node, say 5 or 10 (Slide L10 p.14 `mincut=10`)).
- R Example (Manual splitting logic for `Hitters` data, Slides L10 p.9-12):
- Demonstrates finding the best split for `Hits` by minimizing RSS over possible cutpoints s .
- Then for `Years`.
- Compares RSS reduction from best `Hits` split vs. best `Years` split. Chooses `Years < 4.5` as the first split because it gives lower total RSS.
- R Implementation with `tree` package (Slides L10 p.15-16):

Listing 40: Fitting a Regression Tree with `tree` (Slides L10 p.15)

```

1 # library(ISLR)
2 # data(Hitters)
3 # Hitters <- Hitters[complete.cases(Hitters$Salary),] #
  Remove NAs for Salary
4 # Hitters$logSalary <- log(Hitters$Salary)
5
6 # library(tree)
7 # Fit tree to predict logSalary using Years and Hits
8 # tree.fit1 <- tree(logSalary ~ Years + Hits, data=Hitters,
9 #                   control = tree.control(nobs=nrow(Hitters),
10 #                                         mincut=10)) # mincut is like minleaf
11 # summary(tree.fit1)
12 # plot(tree.fit1)
13 # text(tree.fit1, pretty=0, digits=3) # digits for precision
14 # of node labels
15
16 # Fit tree with all variables (Slide L10 p.16, more complex
17 # tree)
18 # tree.fit.allvars <- tree(logSalary ~ . - Salary, data=
19 #   Hitters,
20 #   control = tree.control(nobs=nrow(
21 #     Hitters), mincut=10))
22 # plot(tree.fit.allvars)
23 # text(tree.fit.allvars, pretty=0)

```

2.11.3 Tree Pruning

- **Problem:** Large trees grown by recursive binary splitting often overfit the training data, leading to poor test performance.
- **Strategy:**
 1. Grow a large initial tree T_0 (e.g., by having a small ‘mincut’ or ‘min-split’).
 2. Prune this tree back to get a sequence of smaller subtrees.
- **Cost Complexity Pruning (Weakest Link Pruning)** (Slide L10 p.17):
- For each possible subtree $T \subseteq T_0$, consider a penalized RSS:

$$\sum_{m=1}^{|T|} \sum_{i: x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

where $|T|$ is the number of terminal nodes in subtree T .

- $\alpha \geq 0$ is a non-negative tuning parameter.
- $\alpha = 0$: T_0 is chosen (no penalty for complexity).
- As α increases, the penalty for more terminal nodes increases, favoring smaller trees.
- Goal: For a given α , find the subtree T_α that minimizes this criterion.
- **Selecting Optimal α (and thus best subtree):**
- Use k-fold cross-validation.
- For each α in a range, get T_α .
- Compute CV error for each T_α .
- Choose α (and corresponding T_α) that minimizes CV error.
- R Implementation (**tree** package, Slides L10 p.18-19):

Listing 41: Pruning a Regression Tree (Slides L10 p.18-19)

```

1 # Assuming tree.fit1 from before (logSalary ~ Years + Hits)
2 # cv.results <- cv.tree(tree.fit1, FUN=prune.tree) # FUN=
  prune.tree is for regression
3 # # For classification, FUN=prune.misclass can be used.
4 # names(cv.results) # "size", "dev", "k", "method"
5 # # 'size' is number of terminal nodes.
6 # # 'dev' is cross-validated deviance (RSS for regression, or
  deviance for classification).
7 # # 'k' is effectively alpha (cost-complexity parameter).
8 #
9 # plot(cv.results$size, cv.results$dev, type="b",
10 #      xlab="Number of Terminal Nodes (Tree Size)", ylab="CV
  Deviance (RSS)")
11 # # Identify the size that minimizes CV deviance.
12 # best.size <- cv.results$size[which.min(cv.results$dev)] # e
  .g., 4 from slide
13 #
14 # # Prune the original tree to this best size
15 # pruned.tree <- prune.tree(tree.fit1, best=best.size)
16 # plot(pruned.tree)
17 # text(pruned.tree, pretty=0)

```

2.11.4 Classification Trees

- Similar to regression trees, but Y is qualitative.
- **Prediction in a region R_m :** Predict the most common class among training observations in R_m .
- **Splitting Criteria** (Instead of RSS, Slide L10 p.21):
- Let \hat{p}_{mk} be the proportion of training observations in region R_m that belong to class k .
- **Classification Error Rate:** $E_m = 1 - \max_k(\hat{p}_{mk})$. Proportion of misclassified obs in region R_m if we predict majority class.
- Not sensitive enough for tree-growing (can lead to same error for different splits).
- Often used for pruning.
- **Gini Index:** $G_m = \sum_{k=1}^K \hat{p}_{mk}(1 - \hat{p}_{mk})$.
- Measure of total variance across K classes, or node purity.
- Small Gini \implies node contains mostly observations from one class.

- **Cross-Entropy (Deviance):** $D_m = - \sum_{k=1}^K \hat{p}_{mk} \log(\hat{p}_{mk})$.
- Similar to Gini index. Small value \implies node is pure.
- Gini or Cross-Entropy are typically used for growing the tree because they are more sensitive to changes in node probabilities than classification error rate.
- Recursive binary splitting proceeds by choosing splits that achieve the largest reduction in Gini index or cross-entropy.
- Pruning is done similarly to regression trees, using CV with error rate, Gini, or deviance.
- R Example (ISLR Carseats data, Slides L10 p.22-25):

Listing 42: Classification Tree for Carseats Data (Slides L10 p.22-25)

```

1 # library(ISLR)
2 # data(Carseats)
3 # Carseats$High <- as.factor(ifelse(Carseats$Sales <= 8, "No
  ", "Yes"))
4 #
5 # library(tree)
6 # tree.carseats <- tree(High ~ . - Sales, data=Carseats) #
  Exclude original Sales
7 # summary(tree.carseats)
8 # plot(tree.carseats)
9 # text(tree.carseats, pretty=0) # Shows classes and
  proportions at nodes
10 #
11 # # Pruning for classification tree
12 # cv.carseats <- cv.tree(tree.carseats, FUN=prune.misclass) #
  Use misclass error for pruning
13 # plot(cv.carseats$size, cv.carseats$k, type="b", # Note:
  Slide plots size vs deviance (k=-alpha)
14 #       xlab="Tree Size", ylab="Alpha (Cost-Complexity)")
15 # # Or plot deviance: plot(cv.carseats$size, cv.carseats$dev,
  type="b")
16 #
17 # # From slide L10 p.24, deviance plot suggests size 4 might
  be optimal
18 # best.size.class <- 4 # Example from slide
19 # pruned.carseats <- prune.misclass(tree.carseats, best=best.
  size.class)
20 # plot(pruned.carseats)
21 # text(pruned.carseats, pretty=0)

```

2.11.5 Pros and Cons of Decision Trees

- **Pros (+):**
 - Simple to understand and interpret.
 - Easy to visualize (graphical representation).
 - Can handle categorical predictors without dummy variables (natively in some packages like `tree`).
 - Can capture non-linear relationships and interactions.
 - Considered to mimic human decision-making.
- **Cons (-):**
 - Predictive accuracy is often not as good as other methods.

- High variance: Small changes in training data can lead to very different tree structures.
- Greedy approach of recursive binary splitting is not guaranteed to find the globally optimal tree.

2.11.6 Exercise: ISLR 8.8a-c

(Corresponds to 'ex8.8.R' file, first part)

- (a) Split data into training and test sets.
- (b) Fit a regression tree to training data. Plot. Interpret. Compute test MSE.

Listing 43: ISLR 8.8b - Basic Regression Tree

```

1 # library(ISLR); library(tree)
2 # data(Carseats)
3 # set.seed(123) # Or seed from problem
4 # train_idx <- sample(1:nrow(Carseats), nrow(Carseats)/2)
5 # train_data <- Carseats[train_idx,]
6 # test_data <- Carseats[-train_idx,]
7 #
8 # tree.sales <- tree(Sales ~ ., data=train_data)
9 # summary(tree.sales) # Variables used, number of nodes,
   deviance
10 # plot(tree.sales); text(tree.sales, pretty=0)
11 # # Interpretation: e.g., ShelfLoc=Good and Price < 109.5 ->
   high sales
12 #
13 # pred.sales.test <- predict(tree.sales, newdata=test_data)
14 # test_mse_tree <- mean((test_data$Sales - pred.sales.test)
   ^2)
15 # # test_mse_tree around 4-5 usually

```

- (c) Use cross-validation to determine optimal tree complexity (pruning). Does pruning improve test MSE?

Listing 44: ISLR 8.8c - Pruning Regression Tree

```

1 # cv.sales <- cv.tree(tree.sales, FUN=prune.tree)
2 # plot(cv.sales$size, cv.sales$dev, type="b", xlab="Tree Size
   ", ylab="CV Deviance")
3 # # Find optimal size (e.g., cv.sales$size[which.min(cv.sales
   $dev)])
4 # best_size_cv <- cv.sales$size[which.min(cv.sales$dev)] # e.
   g., 15 in ex8.8.R
5 #
6 # pruned.sales <- prune.tree(tree.sales, best=best_size_cv)
7 # plot(pruned.sales); text(pruned.sales, pretty=0)
8 #
9 # pred.pruned.test <- predict(pruned.sales, newdata=test_data
   )
10 # test_mse_pruned <- mean((test_data$Sales - pred.pruned.test
   )^2)
11 # # Compare test_mse_pruned with test_mse_tree.
12 # # In ex8.8.R, pruning didn't help or slightly worsened MSE
   for that particular seed.

```

2.12 Lecture 12: Bagging, Random Forests, Boosting

2.12.1 Bagging (Bootstrap Aggregation)

- **Motivation** (Slide L11 p.3): Decision trees suffer from high variance. Averaging a set of observations reduces variance: $\text{Var}(\bar{X}) = \sigma^2/n < \sigma^2 = \text{Var}(X)$.
- **Idea** (Slide L11 p.4): Reduce prediction error variance by fitting many trees on different training sets and averaging their predictions. Since we only have one training set, we use bootstrap.
- **Bagging Algorithm for Regression Trees** (Slides L11 p.5, 12-13):
 1. Generate B bootstrap samples from the original training data (sample with replacement).
 2. For each bootstrap sample $b = 1, \dots, B$:
 3. Grow a regression tree $\hat{f}^b(x)$ on this bootstrap sample. Trees are typically grown deep (unpruned).
 4. To predict for a new point x_0 : Average the predictions from all B trees:

$$\hat{f}_{\text{bag}}(x_0) = \frac{1}{B} \sum_{b=1}^B \hat{f}^b(x_0)$$

- For classification, the prediction is the majority vote among the B trees. (Slide L11 p.13)
- Bagging for KNN (Slides L11 p.5, 8-10): Illustrates the concept. KNN is relatively stable, so bagging offers less improvement compared to unstable methods like trees. Test MSE improved slightly from 0.0607 to 0.0564 in the example.
- **Out-of-Bag (OOB) Error Estimation** (ISLR Ch 8.2.2, Slide L11 p.20):
 - Computationally efficient way to estimate test error without explicit CV.
 - Each bootstrap sample uses 2/3 of original observations. The remaining 1/3 are "out-of-bag" (OOB) for that tree.
 - For each observation i :
 - Predict y_i using only the trees for which observation i was OOB.
 - Average these predictions (or take majority vote).
 - Calculate overall OOB MSE or error rate. Provides a valid estimate of test error.
- R Example (Bagging for `Carseats` Sales, Slides L11 p.14-15):

Listing 45: Bagging with randomForest Package (Slide L11 p.15)

```
1 # library(ISLR); library(randomForest)
2 # data(Carseats)
3 # set.seed(1) # Assuming a seed from lecture example for
  Carseats train/test
4 # train_idx_cs <- sample(1:nrow(Carseats), nrow(Carseats)/2)
5 # train_cs <- Carseats[train_idx_cs,]
6 # test_cs <- Carseats[-train_idx_cs,]
7
8 # For bagging, set mtry = number of predictors (e.g., 10 if
  Sales is excluded)
9 # ncol(train_cs)-1 if Sales is the first column and no other
  exclusions.
```

```

10 # For Carseats, Sales is col 1. Other 10 vars are predictors.
11 # bagging.carseats <- randomForest(Sales ~ ., data=train_cs,
    mtry=10, importance=TRUE)
12 # # Default ntree=500
13 # pred.bag <- predict(bagging.carseats, newdata=test_cs)
14 # mse.bag <- mean((test_cs$Sales - pred.bag)^2)
15 # # Slide L11 p.15 shows mse.bag = 3.113909
16 # # Compare to single tree mse1 = 4.881557 (Slide L11 p.14) -
    Bagging improves.

```

2.12.2 Random Forests

- **Improvement over Bagging:** Bagged trees can be highly correlated if there are a few very strong predictors that are always selected at top splits. Averaging correlated quantities doesn't reduce variance as much.
- **Random Forest Algorithm** (Slide L11 p.17):
 1. Generate B bootstrap samples.
 2. For each bootstrap sample, grow a tree:
 3. At *each split* in the tree, randomly select a subset of m predictors out of the total p predictors.
 4. Choose the best split among these m predictors only.
 5. Average predictions (regression) or take majority vote (classification).
- **Choice of m :**
- Typically $m \approx \sqrt{p}$ for classification.
- Typically $m \approx p/3$ for regression.
- If $m = p$, then Random Forest = Bagging.
- By restricting choice of predictors at each split, Random Forests decorrelate the trees, leading to greater variance reduction when averaged.
- R Example (Carseats Sales, Slides L11 p.18):

Listing 46: Random Forest with randomForest Package (Slide L11 p.18)

```

1 # Assuming train_cs, test_cs from Bagging example
2 # For Carseats, p=10 predictors. mtry=p/3 approx 3.
3 # rf.carseats <- randomForest(Sales ~ ., data=train_cs, mtry
    =3, importance=TRUE)
4 # pred.rf <- predict(rf.carseats, newdata=test_cs)
5 # mse.rf <- mean((test_cs$Sales - pred.rf)^2)
6 # # Slide L11 p.18 shows mserf = 3.568976. This is worse than
    bagging in this specific example.
7 # # Often RF is better or similar. Results can vary with seed
    and mtry choice.

```

2.12.3 Variable Importance Measures (for Bagging/RF)

- Trees make it hard to see overall variable importance when many trees are averaged.
- **Mean Decrease in Impurity (e.g., RSS or Gini)** (Slide L11 p.22):
- For each tree, record total reduction in RSS (regression) or Gini index (classification) due to splits on each predictor.
- Average this reduction over all B trees for each predictor.

- Larger value \implies more important.
- R: `importance(rf.object)()` gives this (IncNodePurity column), `varImpPlot(rf.ob`
- **Permutation Importance (Mean Decrease in Accuracy/MSE):**
- For each tree, calculate OOB error.
- Then, for each predictor X_j :
- Randomly permute the values of X_j in the OOB data for that tree.
- Recalculate OOB error with permuted X_j .
- The average increase in OOB error across all trees due to permuting X_j is its importance.
- R: `importance(rf.object, type=1)()` or `importance=TRUE` in `randomForest()` and check `%IncMSE` (regression) or `MeanDecreaseAccuracy` (classification).
- R Example (Carseats Bagging, Slides L11 p.23):

Listing 47: Variable Importance Plot (Slide L11 p.23)

```

1 # Assuming bagging.carseats from before
2 # varImpPlot(bagging.carseats)
3 # # Shows ShelfLoc and Price as most important by
  IncNodePurity (Gini/RSS decrease)

```

2.12.4 Boosting

- **Core Idea:** Sequentially fit trees. Each new tree is fit to the *residuals* of the previous model, thus focusing on observations the model currently mispredicts.
- This is different from Bagging/RF where trees are grown independently.
- **Boosting Algorithm for Regression Trees** (Slide L11 p.25):
 1. Initialize $\hat{f}(x) = 0$ and residuals $r_i = y_i$ for all i .
 2. For $b = 1, \dots, B$ (number of trees):
 - (a) Fit a tree \hat{f}^b with d splits (often small, e.g., $d = 1$ for stumps) to the training data (X, r) (i.e., predict current residuals).
 - (b) Update the overall prediction: $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x)$.
 - (c) Update residuals: $r_i \leftarrow r_i - \lambda \hat{f}^b(x_i)$.
 3. The final boosted model is $\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x)$. (Slide L11 p.25 has λ multiplying sum, which is equivalent if λ is constant).
- **Tuning Parameters for Boosting:**
- **Number of Trees B :** Unlike Bagging/RF, boosting can overfit if B is too large. Chosen by CV.
- **Shrinkage Parameter λ** (learning rate): Small positive number (e.g., 0.01, 0.1). Controls rate at which boosting learns. Smaller λ requires larger B .
- **Interaction Depth d** (or tree depth): Number of splits in each tree. $d = 1$ gives an additive model. $d > 1$ allows for interactions.
- R Example (Simulated data, Slides L11 p.26-31, Hitters data, Slides L11 p.32-37):

Listing 48: Boosting with gbm Package (Hitters data, Slides L11 p.36)

```

1 # library(ISLR); library(gbm)
2 # data(Hitters)
3 # Hitters <- Hitters[complete.cases(Hitters$Salary),]

```



```

4 # Hitters$logSalary <- log(Hitters$Salary)
5 # set.seed(1234) # From slide
6 # itrain_hit <- sample(1:nrow(Hitters), floor(nrow(Hitters)/
7   2))
8 # traindata_hit <- Hitters[itrain_hit,]
9 # testdata_hit <- Hitters[-itrain_hit,]
10 # ytest_hit <- testdata_hit$logSalary
11
12 # Define formula (slide uses a subset of variables)
13 # formula1_hit <- logSalary ~ CRuns+CHits+HmRun+CWalks+CRBI+
14   Hits+RBI
15
16 # Fit boosted model
17 # boost.hitters <- gbm(formula1_hit, data=traindata_hit,
18   distribution="gaussian",
19   n.trees=1000, interaction.depth=4,
20   shrinkage=0.01)
21 # summary(boost.hitters) # Shows variable importance
22
23 # Predict and calculate Test MSE
24 # pred.gbm <- predict(boost.hitters, newdata=testdata_hit, n.
25   trees=1000)
26 # mse.gbm <- mean((ytest_hit - pred.gbm)^2)
27 # # mse.gbm = 0.3591336 from slide L11 p.36
28 # # mse.simple_tree = 0.3851602 from slide L11 p.35 -
29   Boosting improves.

```

2.12.5 Exercises: ISLR 8.8d-e, 8.11

- **ISLR 8.8d-e (Carseats data):** (Corresponds to ‘ex8.8.R’ file, latter part)
- (d) Bagging: Fit bagged trees for **Sales**. Compute test MSE. Variable importance. *From ‘ex8.8.R’:* Bagging with `mtry=10` (all predictors) gives Test MSE $\approx 2.5 - 2.6$. Price, ShelfLoc, Age are important.
- (e) Random Forest: Fit RF for **Sales**. Vary `mtry`. Compute test MSE. Variable importance. *From ‘ex8.8.R’:* RF with `mtry=5` (approx $p/2$) gives Test MSE $\approx 2.8 - 2.9$. Sometimes bagging performed slightly better for this dataset/seed.
- **ISLR 8.11 (Caravan data):** (Corresponds to ‘ch8-applied.R’ Exercise 11)
- Predict **Purchase**. Highly imbalanced data.
- (b) Boosting: Fit boosted trees. Variable importance.
- (c) Evaluate Boosting: Predict on test set. Use threshold (e.g., 0.2). Compare with logistic regression. Focus on fraction of correctly predicted "Yes" among those predicted "Yes".
- *From ‘ch8-applied.R’ Ex 11:* Boosting (threshold 0.2) correctly identifies 20

2.13 Lecture 13: Support Vector Machines (SVMs)

2.13.1 Introduction to Support Vector Machines

- SVMs are a powerful class of supervised learning algorithms used for classification and regression (though primarily classification is covered).
- We focus on binary classification (two classes).

- Progression of concepts:
 1. **Maximal Margin Classifier**: Assumes classes are perfectly separable by a linear boundary.
 2. **Support Vector Classifier (SVC)**: Extends maximal margin classifier to handle non-separable classes by allowing some misclassifications (soft margin). Still linear boundaries.
 3. **Support Vector Machine (SVM)**: Extends SVC to accommodate non-linear boundaries using kernels.

2.13.2 Hyperplanes

- A hyperplane in a p -dimensional space is a flat affine subspace of dimension $p - 1$.
- Equation: $\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p = 0$.
- If $p = 2$: Line, $X_2 = -\frac{\beta_0}{\beta_2} - \frac{\beta_1}{\beta_2} X_1$.
- If $p = 3$: Plane.
- A hyperplane divides the p -dimensional space into two half-spaces.

2.13.3 Classification by Separating Hyperplanes

- Data: n training observations (\mathbf{x}_i, y_i) , where $\mathbf{x}_i \in \mathbb{R}^p$ and $y_i \in \{-1, 1\}$.
- Decision rule for a test observation \mathbf{x} :
- Classify as $+1$ if $f(\mathbf{x}) = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p > 0$.
- Classify as -1 if $f(\mathbf{x}) < 0$.
- (Slide L12 p.6-7 illustrates this for $p = 2$).
- For a separating hyperplane, all training observations satisfy $y_i f(\mathbf{x}_i) > 0$.
- The magnitude $|f(\mathbf{x})|$ indicates confidence in the classification (larger magnitude \implies further from hyperplane).

2.13.4 The Maximal Margin Classifier (MMC)

- If classes are linearly separable, there can be infinitely many separating hyperplanes.
- **MMC**: The separating hyperplane that is farthest from the closest training observations from either class. (Slide L12 p.8).
- **Margin (M)**: The perpendicular distance from the hyperplane to the nearest training observation. MMC maximizes M .
- **Support Vectors**: Training observations that lie exactly on the margin (equidistant from the hyperplane). They "support" the hyperplane; if moved, the hyperplane would change.
- **Optimization Problem** (Slide L12 p.9):

$$\max_{\beta_0, \dots, \beta_p, M} M$$

Subject to:

$$\sum_{j=1}^p \beta_j^2 = 1 \quad (\text{Normalization constraint}) \quad (1)$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M \quad \forall i = 1, \dots, n \quad (2)$$

The term $y_i f(\mathbf{x}_i)$ is the functional margin. With $\sum \beta_j^2 = 1$, M becomes the geometric margin.

- Slide L12 p.10 shows the maximal margin hyperplane (black) and the margin boundaries (red/blue).
- **Limitation:** Only exists if classes are perfectly linearly separable. Sensitive to outliers. (Slide L12 p.11 shows a non-separable case).

2.13.5 Support Vector Classifier (SVC) / Soft Margin Classifier

- Handles non-linearly separable data or cases where a wider margin with a few errors is preferred.
- Allows some observations to be on the wrong side of the margin, or even on the wrong side of the hyperplane.
- **Optimization Problem** (Slide L12 p.12):

$$\max_{\beta_0, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n, M} M$$

Subject to:

$$\sum_{j=1}^p \beta_j^2 = 1 \quad (3)$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \quad \forall i \quad (4)$$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^n \epsilon_i \leq C \quad (5)$$

- **Slack Variables** ϵ_i (Slide L12 p.13):
- $\epsilon_i = 0$: Observation i is on the correct side of its margin.
- $0 < \epsilon_i \leq 1$: Observation i is on the correct side of the hyperplane but on the wrong side of its margin (margin violation).
- $\epsilon_i > 1$: Observation i is on the wrong side of the hyperplane (misclassified).
- **Tuning Parameter** C (Cost/Budget for violations, Slide L12 p.14):
- Controls the trade-off between margin width and number/severity of margin violations.
- Small C : Wide margin, more violations allowed (high bias, low variance). Similar to larger s in $\sum \beta_j^2 \leq s$.
- Large C : Narrow margin, fewer violations allowed (low bias, high variance). Approaches MMC if data is separable.
- C is chosen by cross-validation.
- Only observations on the margin or violating the margin (support vectors) affect the solution. (Slide L12 p.13)
- Still produces a linear decision boundary.

2.13.6 Support Vector Machines (SVMs) - Non-linear Boundaries

- Extends SVC to handle non-linear decision boundaries by using **kernels**.
- **The Kernel Trick:** Instead of explicitly mapping data to a higher-dimensional space to find a linear separator, kernels compute inner products in that higher-dimensional space directly using original data.

- The decision function becomes (Slide L12 p.19, simplified from dual form):

$$f(\mathbf{x}) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$

where \mathcal{S} is the set of support vectors, α_i are parameters, and $K(\cdot, \cdot)$ is a kernel function.

- **Common Kernels** (Slide L12 p.19):
- **Linear Kernel:** $K(\mathbf{x}_i, \mathbf{x}_{i'}) = \mathbf{x}_i' \mathbf{x}_{i'}$. Recovers the SVC.
- **Polynomial Kernel** of degree d : $K(\mathbf{x}_i, \mathbf{x}_{i'}) = (1 + \gamma \mathbf{x}_i' \mathbf{x}_{i'} + r)^d$. (Slide has $1 + \sum x_{ij} x_{i'j}$). Tuning parameters: d, γ , cost C .
- **Radial Basis Function (RBF) Kernel / Gaussian Kernel:**

$$K(\mathbf{x}_i, \mathbf{x}_{i'}) = \exp \left(-\gamma \sum_{j=1}^p (x_{ij} - x_{i'j})^2 \right) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_{i'}\|^2)$$

Tuning parameters: $\gamma > 0$, cost C .

- If γ is very large, fit is very local (wiggly, high variance).
- If γ is very small, fit is very smooth (like linear, high bias).

2.13.7 R Implementation of SVMs

- Main function: `svm()` from `e1071`.
- Key arguments:
- `formula` or `x`, `y` input.
- `data`: Training data.
- `kernel`: "linear", "polynomial", "radial" (default), "sigmoid".
- `cost`: Parameter C (SVC/SVM).
- `degree`: For polynomial kernel.
- `gamma`: For polynomial and radial kernels.
- `scale = TRUE/FALSE`: Whether to standardize predictors. (Default TRUE). (Slide L12 p.17 shows FALSE for manual data).
- **Tuning Parameters** using `tune()` from `e1071`:

Listing 49: Tuning SVM with `e1071::tune` (ISLR Ch9 Lab, Ex 9.7)

```

1 # library(e1071)
2 # set.seed(1) # For reproducibility
3 # # Example data from ISLR Ch9 Lab / Slides L12 p.15
4 # x_sim_svm <- matrix(rnorm(202), ncol=2)
5 # y_sim_svm <- c(rep(-1,10), rep(1,10))
6 # x_sim_svm[y_sim_svm==1,] <- x_sim_svm[y_sim_svm==1,] + 1 #
  Create separable classes
7 # dat_sim_svm <- data.frame(x=x_sim_svm, y=as.factor(y_sim_
  svm))
8
9 # # Tune linear SVM for cost
10 # tune.out.linear <- tune(svm, y ~ ., data=dat_sim_svm,
  kernel="linear",
11 #                               ranges=list(cost=c(0.001, 0.01,
  0.1, 1, 5, 10, 100)))
12 # summary(tune.out.linear) # Shows best cost and CV error
13 # bestmod.linear <- tune.out.linear$best.model

```

```

14 # summary(bestmod.linear) # Shows parameters of best model,
    number of SVs
15
16 # # Plotting the SVM decision boundary (Slides L12 p.17)
17 # # svmfit.linear <- svm(y ~ ., data=dat_sim_svm, kernel="
    linear", cost=10, scale=FALSE)
18 # # plot(svmfit.linear, dat_sim_svm) # For 2D data
19
20 # # Example with radial kernel (Slides L12 p.21)
21 # # svmfit.radial <- svm(y ~ ., data=dat_sim_svm, kernel="
    radial", cost=10, gamma=1, scale=FALSE)
22 # # plot(svmfit.radial, dat_sim_svm)

```

- **Support Vectors:** Can be accessed via `svmfit$index`.

2.13.8 SVMs with More than Two Classes

- **One-Versus-One (OVO):** Construct $\binom{K}{2}$ SVMs, each comparing a pair of classes. Classify a new observation by majority vote among these classifiers.
- **One-Versus-All (OVA):** Fit K SVMs, each separating one class from the rest. Classify to the class whose SVM gives largest $|f(\mathbf{x})|$ (or highest probability if calibrated).
- `e1071` typically uses OVO for multi-class SVM.

2.13.9 Relationship to Logistic Regression

- SVC optimization problem has similarities to logistic regression, particularly when L_2 penalty is used with logistic regression.
- Loss function for SVC (hinge loss) vs. logistic loss. Hinge loss is zero for points correctly classified beyond the margin.
- In practice, performance can be similar, especially with linear boundaries. SVMs can be more robust to outliers when classes are well-separated due to focus on support vectors.

2.13.10 Exercise: ISLR 9.7

(Corresponds to ‘ex9.7.R’)

- (a) Create binary variable `high`: 1 if `mpg > median(mpg)`, 0 otherwise.
- (b) SVC (linear kernel): Use `tune()` to find best `cost`. From ‘ex9.7.R’: Best cost is often 1.
- (c) SVM with polynomial and radial kernels: Use `tune()` for `cost`, `degree` (poly), `gamma` (radial). From ‘ex9.7.R’:
 - Polynomial: E.g., `cost=10`, `degree=2`.
 - Radial: E.g., `cost=10`, `gamma=0.01`.
- (d) Plot results and compare predictions (e.g., training confusion matrices).
- Radial kernel often gives good separation on training data, but generalization to test data is key.
- Linear SVM with optimal cost usually performs reasonably well.

Listing 50: ISLR 9.7 - Tuning SVM (Example for Radial)

```

1 # library(ISLR); library(e1071)

```

```

2 # Auto$high <- as.factor(ifelse(Auto$mpg > median(Auto$mpg),
  1, 0))
3 # Auto_svm <- Auto[, !(names(Auto) %in% c("mpg", "name"))] #
  Exclude original mpg and name
4
5 # set.seed(1)
6 # tune.radial <- tune(svm, high ~ ., data=Auto_svm, kernel="
  radial",
7 #                       ranges=list(cost=c(0.1, 1, 10, 100),
  gamma=c(0.01, 0.1, 1, 5)))
8 # summary(tune.radial)
9 # best.svm.radial <- tune.radial$best.model
10 # plot(best.svm.radial, Auto_svm, horsepower ~ weight) #
  Example plot

```

2.14 Lecture 14: Unsupervised Learning

2.14.1 Introduction to Unsupervised Learning

- **Key Characteristic:** We only observe predictors \mathbf{X} , no response variable y .
- **Goals:**
 - Exploratory / descriptive data analysis.
 - Detect unusual/remarkable patterns or subgroups in observations.
 - Generate hypotheses about the data structure.
 - Can be seen as "Advanced descriptive analysis."
- **Main Techniques Covered:**
 - Principal Component Analysis (PCA) for dimension reduction and visualization.
 - Clustering (K-Means, Hierarchical) for finding subgroups.

2.14.2 Principal Component Analysis (PCA)

- **Motivation** (Slides L13 p.3-4):
 - Visualizing high-dimensional data ($p > 2$) is difficult. A pairs plot for $p = 10$ yields $\binom{10}{2} = 45$ scatterplots.
 - PCA aims to summarize p variables into a smaller set of $M < p$ new variables (principal components) that capture as much of the original variability as possible.
- **What are Principal Components?**
 - Principal components (PCs) are linear combinations of the original predictors:

$$Z_m = \phi_{1m}X_1 + \phi_{2m}X_2 + \cdots + \phi_{pm}X_p = \sum_{j=1}^p \phi_{jm}X_j$$

- The coefficients ϕ_{jm} are called **loadings**. The loading vector for PC m is $\phi_m = (\phi_{1m}, \dots, \phi_{pm})'$.
- Loadings define the direction in feature space along which the data varies the most (for Z_1), or varies most subject to being uncorrelated with previous PCs (for Z_2, Z_3, \dots).
- **First Principal Component (Z_1)** (Slide L13 p.5):

- Z_1 is the normalized linear combination of X_j 's that has the largest variance:

$$\text{maximize } \text{Var}(Z_1) = \text{Var} \left(\sum_{j=1}^p \phi_{j1} X_j \right)$$

subject to the normalization constraint $\sum_{j=1}^p \phi_{j1}^2 = 1$.

- The loadings ϕ_{j1} define the first principal component direction.
- **Second Principal Component (Z_2)** (Slide L13 p.6):
- Z_2 is the normalized linear combination of X_j 's that has the largest variance, subject to being uncorrelated with Z_1 ($\text{Cov}(Z_1, Z_2) = 0$).

$$\text{maximize } \text{Var}(Z_2) \quad \text{subject to} \quad \sum_{j=1}^p \phi_{j2}^2 = 1 \quad \text{and} \quad \text{Cov}(Z_1, Z_2) = 0$$

- Subsequent PCs are defined similarly (maximize variance, uncorrelated with all previous PCs). Up to $\min(n-1, p)$ PCs can be constructed.
- **Geometric Interpretation:** PCs represent new axes in the feature space. Z_1 is the direction of greatest spread, Z_2 is the next direction of greatest spread orthogonal to Z_1 , etc.
- **Scaling of Variables:** It is crucial to scale variables to have standard deviation one (and often mean zero) before performing PCA if they are measured on different scales. Otherwise, variables with larger variance will dominate the PCs. (Slide L13 p.13 uses `scale=TRUE` with `prcomp()`).
- **Proportion of Variance Explained (PVE):**
- The variance explained by PC m is $\text{Var}(Z_m)$. Total variance is $\sum_{j=1}^p \text{Var}(X_j)$ (if scaled, this is p).
- $PVE_m = \frac{\text{Var}(Z_m)}{\sum_{j=1}^p \text{Var}(X_j)}$.
- Cumulative PVE helps decide how many PCs to retain.
- **Scree Plot:** Plot of PVE for each PC (or eigenvalues $\text{Var}(Z_m)$). Look for an "elbow" to decide number of components to keep.
- **R Implementation** (`prcomp()` or `princomp()`):
- `prcomp()` is generally preferred (uses SVD).
- `rotation` element contains loadings ϕ_{jm} .
- `x` element contains the principal component scores z_{im} .
- `sdev` contains standard deviations of PCs (square for variances).
- R Example (Simulated $p = 3$ data, Slides L13 p.7-10):

Listing 51: PCA on Simulated 3D Data (Slides L13 p.9-10)

```

1 # Assume 'x' is the n x 3 matrix of simulated data
2 # pca_sim <- princomp(x) # Or prcomp(x, scale.=TRUE, center.=
  TRUE)
3 # summary(pca_sim)
4 # Output from slide L13 p.9:
5 # Importance of components:
6 #
7 # Standard deviation      1.5940343 0.6333490 0.031345120
8 # Proportion of Variance 0.8633688 0.1362973 0.000333842
9 # Cumulative Proportion  0.8633688 0.9996662 1.000000000
10 # Almost all variance (99.97%) captured by first two PCs.
11
12 # plot(pca_sim$scores[,1:2]) # Plot observations in PC1-PC2
  space (Slide L13 p.10)

```

- R Example (ISLR USArrests data, Slides L13 p.11-16):

Listing 52: PCA on USArrests Data (Slides L13 p.13-16)

```

1 # library(ISLR)
2 # data(USArrests)
3 # # summary(USArrests) # Variables have different scales/
  # means
4 # # apply(USArrests, 2, mean)
5 # # apply(USArrests, 2, var)
6
7 # pca.usarrests <- prcomp(USArrests, scale=TRUE, center=TRUE)
8 # summary(pca.usarrests)
9 # # Importance of components:
10 # #
11 # # Standard deviation      PC1      PC2      PC3      PC4
12 # # Proportion of Variance  0.6201  0.2474  0.08914  0.04336
13 # # Cumulative Proportion   0.6201  0.8675  0.95664  1.00000
14 # # First two PCs explain ~87% of variance.
15
16 # pca.usarrests$rotation # Loadings (Slide L13 p.15)
17 # # PC1: High negative loadings for Murder, Assault, Rape (
  # crime factor), negative for UrbanPop.
18 # # States with low PC1 scores have high crime rates.
19 # # PC2: High positive loading for UrbanPop, small negative
  # for Assault. (urbanization factor)
20 # # States with high PC2 scores are highly urbanized.
21
22 # # Biplot (Slide L13 p.14 shows state names, Slide L13 p.16
  # shows arrows for variables)
23 # biplot(pca.usarrests, scale=0) # scale=0 ensures arrows
  # represent loadings

```

2.14.3 Clustering Methods

- Goal: Find homogeneous subgroups (clusters) among observations. Observations within a cluster should be similar; observations in different clusters should be dissimilar.
- "Similarity" is context-dependent, often based on Euclidean distance between observations in feature space.
- Examples (Slide L13 p.17): Classifying plants based on features, identifying consumer subgroups based on shopping patterns.

2.14.4 K-Means Clustering

- **Objective:**
- Partition n observations into a pre-specified number of K clusters C_1, \dots, C_K .
- Clusters must be non-overlapping ($C_k \cap C_{k'} = \emptyset$) and cover all observations ($\cup C_k = \{1, \dots, n\}$).
- Minimize within-cluster variation (WCV). A common measure is sum of squared Euclidean distances between observations in a cluster and the cluster centroid, or sum of pairwise squared Euclidean distances within a cluster.

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

where $W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^p (x_{ij} - x_{i'j})^2$. This is equivalent to minimizing $2 \sum_{k=1}^K \sum_{i \in C_k} \sum_{j=1}^p (x_{ij} - \bar{x}_{kj})^2$, where \bar{x}_{kj} is the mean of feature j for cluster k (centroid). (ISLR Ch10 Exercise 10.1).

- **Algorithm** (Greedy, iterative, Slide L13 p.20):
 1. Randomly assign each observation to one of K clusters.
 2. Iterate until cluster assignments stop changing:
 - (a) For each cluster k , compute its centroid μ_k (vector of feature means for observations in C_k).
 - (b) Reassign each observation to the cluster whose centroid is closest (e.g., by Euclidean distance).
- **Sensitivity to Initialization:** Final clustering can depend on initial random assignment. Run algorithm multiple times with different random starts (argument `nstart` in R's `kmeans()`) and choose the solution with smallest total WCV.
- **Choosing K:** No single best method.
- Elbow method: Plot total WCV (or within-cluster sum of squares, `tot.withinss`) vs. K . Look for an "elbow" point where adding more clusters gives diminishing returns.
- Silhouette analysis.
- Domain knowledge or practical considerations.
- R Example (Simulated 2D data, 3 clusters, Slides L13 p.21-27):

Listing 53: K-Means Clustering in R (Slides L13 p.22, 26-27)

```

1  # Assume 'x_cluster_sim' is an n x 2 matrix of simulated data
   # with 3 true clusters
2  # K_val <- 3
3
4  # Manual iteration steps (conceptual from slides L13 p.22-25)
5  # 1. Random initial assignment:
6  #   cluster_colors <- c("black", "red", "blue")
7  #   initial_assignment <- sample(1:K_val, size=nrow(x_
   cluster_sim), replace=TRUE)
8  #   plot(x_cluster_sim, col=cluster_colors[initial_
   assignment])
9  #   centroids <- matrix(0, K_val, ncol(x_cluster_sim))
10 #   for(k_idx in 1:K_val) {
11 #     centroids[k_idx,] <- colMeans(x_cluster_sim[initial_
   assignment==k_idx,])
12 #     points(centroids[k_idx,1], centroids[k_idx,2], pch=17,
   col=cluster_colors[k_idx], cex=2)
13 #   }
14 # 2. Reassign based on distance to centroids, recompute
   centroids ... (iterate)
15
16 # Using kmeans() function (Slide L13 p.26-27)
17 # set.seed(123) # For reproducibility of kmeans if nstart=1
18 # km.out <- kmeans(x_cluster_sim, centers=K_val, nstart=20) #
   nstart=20 runs 20 random initializations
19 # names(km.out) # "cluster", "centers", "totss", "withinss",
   "tot.withinss", "betweenss", "size"
20 #
21 # plot(x_cluster_sim, col=cluster_colors[km.out$cluster],
   main="K-Means Clustering Result (K=3)")

```

```
22 # points(km.out$centers, col=cluster_colors, pch=17, cex=2,
    lwd=2)
```

2.14.5 Hierarchical Clustering

- Does not require pre-specifying K .
- Produces a tree-based representation (dendrogram) of the observations.
- **Agglomerative (Bottom-Up) Approach** (Algorithm on Slide L13 p.32):
 1. Start with each of n observations as its own cluster.
 2. Compute all $n(n-1)/2$ pairwise dissimilarities (e.g., Euclidean distances).
 3. For $i = n, n-1, \dots, 2$:
 - (a) Fuse the two closest (most similar) clusters. The distance at which they fuse becomes the height in the dendrogram.
 - (b) Compute new pairwise inter-cluster dissimilarities among the $i-1$ remaining clusters.
- **Linkage Methods** (How to define dissimilarity between clusters, Slide L13 p.33):
 - *Complete*: Maximal inter-cluster dissimilarity (largest distance between an obs in cluster A and an obs in cluster B).
 - *Single*: Minimal inter-cluster dissimilarity (smallest distance). Can result in "chaining."
 - *Average*: Mean inter-cluster dissimilarity.
 - *Centroid*: Dissimilarity between centroids of two clusters. Can lead to inversions in dendrogram (later fusions at lower height).
 - *Ward's method*: Minimizes increase in total within-cluster variance upon merging.
- **Dendrogram** (Slide L13 p.30, USArrests example):
 - Leaves are individual observations.
 - Fusions are represented by horizontal lines; height indicates dissimilarity at fusion.
 - Cut dendrogram at a certain height to obtain K clusters. (Slide L13 p.31)
 - Choice of dissimilarity measure (Euclidean, correlation-based, etc.) and linkage method can significantly affect results.
- R Example (USArrests data, Slide L13 p.29-30):

Listing 54: Hierarchical Clustering in R (Slides L13 p.30)

```
1 # data(USArrests)
2 # # Hierarchical clustering with complete linkage and
  # Euclidean distance
3 # hc.complete <- hclust(dist(USArrests), method="complete")
4 # plot(hc.complete, main="Complete Linkage", xlab="", sub="",
  # cex=0.9)
5 #
6 # # Cutting the tree to get specific number of clusters
7 # cutree(hc.complete, k=3) # Get 3 clusters
8 #
9 # # Scaling data is often recommended before calculating
  # distances
```

```

10 # hc.complete.scaled <- hclust(dist(scale(USArrests)), method
    ="complete")
11 # plot(hc.complete.scaled, main="Complete Linkage (Scaled
    Data)")

```

2.14.6 Practical Issues in Clustering

- **Scaling Variables:** Important if variables are on different scales, as distance measures will be dominated by variables with larger magnitudes/variances. Standardize to mean 0, std dev 1.
- **Choice of K (for K-Means) or Cut Height (for Hierarchical):** Often subjective or guided by external criteria/domain knowledge.
- **Interpreting Clusters:** Examine feature means/distributions within each cluster.
- Clustering is exploratory; results should be interpreted with caution and domain expertise.

2.14.7 Exercises: ISLR 12.8, 12.9 (

- **ISLR 12.8 (old 10.8, PCA on USArrests)** (Corresponds to ‘ch10-lab1.R’, ‘ch10ex8.R’):
 - (a) Calculate PVE for each PC.
 - (b) Relate PVE to sum of squared PC scores and sum of squared scaled original data. (PVE for PC m is $\frac{\sum_{i=1}^n z_{im}^2}{\sum_{j=1}^p \sum_{i=1}^n \tilde{x}_{ij}^2}$, where z_{im} are scores for PC m , and \tilde{x}_{ij} are scaled original data.)
- **ISLR 12.9 (old 10.9, Hierarchical Clustering on USArrests)** (Corresponds to ‘ch10-lab2.R’, ‘ch10ex9.R’):
 - (a) Hierarchical clustering with complete linkage, Euclidean distance. Plot dendrogram.
 - (b) Cut tree to get 3 clusters. Identify states in each.
 - (c) Repeat with scaled data. Plot.
 - (d) Compare results. Scaling can change cluster assignments significantly. Discuss if scaling is appropriate (generally yes if variables have different units/scales).

3 Previous Exams

3.1 Exam Spring 2021

Overall Datasets: Smarket (ISLR), dataCar(insuranceData)

Task 1: Analysis of Smarket Data

3.1.1 1a: Bootstrap histogram for volatility

Question Use the bootstrap to create a histogram of the sampling distribution for the volatility (i.e., the standard deviation of the returns) of the S&P stock index. Observations of the returns are in the variable **Today**. Start by setting the seed to 1 (`set.seed(1)`). Use 1000 bootstrap replicates.

Base R Solution

```
1 library(ISLR)
2 library(boot)
3 data(Smarket)
4
5 set.seed(1)
6 calculate_sd_Smarket <- function(data_vector, index) {
7   return(sd(data_vector[index]))
8 }
9 boot_volatility_Smarket <- boot(data = Smarket$Today, statistic
  = calculate_sd_Smarket, R = 1000)
10 hist(boot_volatility_Smarket$t, main = "Bootstrap Dist. of
  Volatility (Smarket$Today)",
11       xlab = "Estimated Volatility (Std Dev)", col = "lightblue",
        breaks = "Scott")
```

Tidyverse Solution (Data Prep + Base R boot)

```
1 library(ISLR); library(boot); library(dplyr); library(ggplot2)
2 data(Smarket)
3 today_returns_Smarket <- Smarket %>% pull(Today)
4 set.seed(1)
5 # calculate_sd_Smarket function is the same
6 boot_volatility_tidy_Smarket <- boot(data = today_returns_
  Smarket, statistic = calculate_sd_Smarket, R = 1000)
7 ggplot(data.frame(volatility = boot_volatility_tidy_Smarket$t),
  aes(x = volatility)) +
8   geom_histogram(aes(y=..density..), fill = "lightblue", color =
  "black", bins=30) +
9   geom_density(alpha=.2, fill="#FF6666") +
10   labs(title="Bootstrap Distribution of Volatility (Smarket$
  Today)",
11         x="Estimated Volatility", y="Density")
```

3.1.2 1b: 95% CI for volatility (normal assumption)

Question Compute a 95% confidence interval based on the bootstrap, assuming that the volatility is normally distributed.

Base R Solution (using boot)

```
1 # Assuming boot_volatility_Smarket from 1a
2 ci_normal_Smarket <- boot.ci(boot_volatility_Smarket, type = "
  norm")
3 print(ci_normal_Smarket)
4 # Interpretation: Based on the normal approximation, we are 95%
  confident the true volatility is between [lower] and [upper].
```

3.1.3 1c: 95% CI for volatility (percentile method)

Question Compute a 95% confidence interval based on the bootstrap, not making the normality assumption but use percentiles from the bootstrapped volatilities.

Base R Solution (using boot)

```
1 # Assuming boot_volatility_Smarket from 1a
2 ci_percentile_Smarket <- boot.ci(boot_volatility_Smarket, type =
  "perc")
```

```

3 print(ci_percentile_Smarket)
4 # Interpretation: Using percentiles, we are 95% confident the
  true volatility is between [lower] and [upper].

```

3.1.4 1d: Regression of Squared Returns

Question *Squared returns, Today^2 , can be used as a proxy for volatility. Squared returns are often autocorrelated, indicating that volatility can be predicted. Run a regression of the square of **Today** on the square of **Lag1** and interpret the estimate of the regression parameter for the square of **Lag1**.*

Base R Solution

```

1 model_sq_returns_Smarket <- lm(I(Today^2) ~ I(Lag1^2), data =
  Smarket)
2 summary_model_sq_Smarket <- summary(model_sq_returns_Smarket)
3 print(summary_model_sq_Smarket)
4 # Interpretation of I(Lag1^2) coefficient:
5 # A one-unit increase in Lag1^2 (yesterday's squared return) is
  associated
6 # with an estimated [coefficient_value, e.g., 0.165] change in
  Today^2 (today's squared return).
7 # This relationship is statistically significant (p-value = [p_
  val]).

```

3.1.5 1e: Bootstrap SE of regression coefficient

Question *Compute the bootstrapped standard error of the regression coefficient in front of the square of **Lag1** in the model in d) and compare it with the standard error from the regression output.*

Base R Solution

```

1 coef_sq_lag1_Smarket_bs <- function(data, index) {
2   subset_data <- data[index, ]
3   fit <- lm(I(Today^2) ~ I(Lag1^2), data = subset_data)
4   return(coef(fit)[2])
5 }
6 set.seed(1) % \textit{(Or seed from marking scheme for this
  specific subtask if different, e.g., if a variant was used.)}
  %
7 boot_coef_Smarket <- boot(data = Smarket, statistic = coef_sq_
  lag1_Smarket_bs, R = 1000)
8 bootstrap_se_coef_Smarket <- sd(boot_coef_Smarket$t)
9 cat("Bootstrap SE for I(Lag1^2) coef (Smarket):", bootstrap_se_
  coef_Smarket, "\n")
10
11 lm_se_coef_Smarket <- summary(model_sq_returns_Smarket)$
  coefficients["I(Lag1^2)", "Std. Error"]
12 cat("LM Output SE for I(Lag1^2) coef (Smarket):", lm_se_coef_
  Smarket, "\n")
13 # Comparison: The bootstrap SE ([value]) is [larger/smaller/
  similar] than the lm output SE ([value]).
14 # Marking scheme: Bootstrap SE (0.0488) was almost twice lm SE
  (0.0279).

```

Task 2: Analysis of dataCar Data - Predicting claimcst0

3.1.6 2a: Data Filtering and Variable Removal

Question Start by first selecting the observations where there is an occurrence of claim, `clm!=0`. Also, remove the variable `X_OBSTAT_`. Work with this new data set for the rest of Task 2.

Base R Solution

```
1 library(insuranceData)
2 data(dataCar)
3 xdata_task2_car <- dataCar[dataCar$clm != 0, ]
4 xdata_task2_car <- xdata_task2_car[, names(xdata_task2_car) != "
  X_OBSTAT_"]
5 # print(paste("Dimensions after filtering:", paste(dim(xdata_
  task2_car), collapse="x")))
```

Tidyverse Solution

```
1 library(insuranceData); library(dplyr)
2 data(dataCar)
3 xdata_task2_car_tidy <- dataCar %>%
4   filter(clm != 0) %>%
5   select(-any_of("X_OBSTAT_"))
6 # glimpse(xdata_task2_car_tidy)
```

3.1.7 2b: Descriptive statistics and variable formatting

Question Use descriptive statistics, the description of the data in the help-function and common sense to remove variables that you think will not be helpful. Motivate your choices thoroughly. Also, make sure that all variables are on the right format.

Base R / Tidyverse Solution (Conceptual - based on marking scheme)

```
1 # Assuming xdata_task2_car from 2a
2 # Marking scheme solution: Remove clm (col 3 in their xdata
  after X_OBSTAT_ removal)
3 # Original columns: veh_value, exposure, clm, numclaims,
  claimcst0, veh_body, veh_age, gender, area, agecat
4 # After X_OBSTAT_ removal and clm filtering, clm column is now
  constant 1.
5 # If X_OBSTAT_ was, say, col 11 as in marking scheme:
6 # data(dataCar); xdata_task2_car <- dataCar[dataCar$clm !=0,];
  xdata_task2_car <- xdata_task2_car[,-11]
7 # Original column indices matter for marking scheme solution.
8 # Let's assume xdata_task2_car is as per previous step (X_OBSTAT
  _ removed).
9 # 'clm' column is now all 1s.
10
11 # Removing 'clm' as it's constant after filtering:
12 xdata_task2_car_processed <- xdata_task2_car[, names(xdata_task2
  _car) != "clm"]
13
14 # Recoding 'veh_age' and 'agecat' to factors as they represent
  categories
15 xdata_task2_car_processed$veh_age <- as.factor(xdata_task2_car_
  processed$veh_age)
16 xdata_task2_car_processed$agecat <- as.factor(xdata_task2_car_
  processed$agecat)
```

```

17 # 'numclaims' is kept as numeric. 'exposure' also kept.
18
19 # Motivation for removals/changes:
20 # - 'clm': After filtering for clm!=0, this variable is constant
    and uninformative for predicting claimcst0 among those who
    claimed.
21 # - 'veh_age', 'agecat': These are described as age categories (
    e.g., "youngest"), making factor representation appropriate.
22 # - 'X_OBSTAT_': Likely an ID, not predictive.
23 # - Other variables (e.g., 'exposure', 'numclaims') are kept as
    potential predictors for claim amount.
24 # print(sapply(xdata_task2_car_processed, class))

```

3.1.8 2c: Cross-validation for linear models

Question Use a cross validation approach to evaluate predictions for *claimcst0* using: 1) Full LM, 2) All simple LMs, 3) Intercept-only LM. Motivate CV method. Seed 123 (Marking guide used 543, this exam used 123 in task description, so using 123).

Base R Solution (5-fold CV)

```

1 # Assuming xdata_task2_car_processed from 2b
2 set.seed(123)
3 K_cv <- 5
4 n_car_proc <- nrow(xdata_task2_car_processed)
5 folds_cv_car <- sample(cut(seq(1, n_car_proc), breaks=K_cv,
    labels=FALSE))
6
7 response_car <- "claimcst0"
8 predictors_all_car <- names(xdata_task2_car_processed)[names(
    xdata_task2_car_processed) != response_car]
9
10 # Matrix to store MSEs
11 mse_results_car <- matrix(NA, nrow=K_cv, ncol=2 + length(
    predictors_all_car))
12 colnames(mse_results_car) <- c("FullLM", "InterceptOnly",
    predictors_all_car)
13
14 for (k_idx_cv in 1:K_cv) {
15   test_idx_cv <- which(folds_cv_car == k_idx_cv)
16   train_df_cv <- xdata_task2_car_processed[-test_idx_cv, ]
17   test_df_cv <- xdata_task2_car_processed[test_idx_cv, ]
18
19   # Full LM
20   lm_full_cv_car <- lm(paste(response_car, "~ ."), data=train_df_
    _cv)
21   pred_full_cv_car <- predict(lm_full_cv_car, newdata=test_df_cv
    )
22   mse_results_car[k_idx_cv, "FullLM"] <- mean((test_df_cv[[
    response_car]] - pred_full_cv_car)^2, na.rm=TRUE)
23
24   # Intercept-only LM
25   lm_int_cv_car <- lm(paste(response_car, "~ 1"), data=train_df_
    cv)
26   pred_int_cv_car <- predict(lm_int_cv_car, newdata=test_df_cv)
27   mse_results_car[k_idx_cv, "InterceptOnly"] <- mean((test_df_cv
    [[response_car]] - pred_int_cv_car)^2, na.rm=TRUE)
28
29   # Simple LMs

```



```

30 for (pred_name_car in predictors_all_car) {
31   # Ensure factor levels are consistent or handle potential
      errors if a level is missing in a fold
32   # This might require more careful data handling in a real
      scenario for factors
33   if (is.factor(train_df_cv[[pred_name_car]]) && length(levels(
      droplevels(train_df_cv[[pred_name_car]]))) < 2) next
34
35   formula_simple_cv_car <- as.formula(paste(response_car, "~",
      pred_name_car))
36   tryCatch({ # In case of errors with factors in small folds
37     lm_simple_cv_car <- lm(formula_simple_cv_car, data=train
      _df_cv)
38     pred_simple_cv_car <- predict(lm_simple_cv_car, newdata=
      test_df_cv)
39     mse_results_car[k_idx_cv, pred_name_car] <- mean((test_
      df_cv[[response_car]] - pred_simple_cv_car)^2, na.rm=
      TRUE)
40   }, error = function(e) { print(paste("Error with", pred_name
      _car, "in fold", k_idx_cv))})
41 }
42 }
43 cv_avg_mses_car <- colMeans(mse_results_car, na.rm=TRUE)
44 print("Average CV Test MSEs for different models:")
45 print(sort(cv_avg_mses_car))
46 # Motivation for K=5: Balances bias (not too small training
      folds) and variance (multiple estimates)
47 # of the test error estimate, and is computationally feasible
      for fitting many models.

```

3.1.9 2d: Generalized Additive Model (GAM)

Question Use a GAM with smoothing splines for the numerical variables. Include categorical variables in the model as well. Use `s()`-function in `gam`-library, with argument `df=4`, without motivation. Plot the result and comment on the relationship between predictors and dependent variable.

Base R Solution (using gam)

```

1 library(gam)
2 # Assuming xdata_task2_car_processed from 2b
3 num_preds_car_gam <- names(xdata_task2_car_processed)[apply(
      xdata_task2_car_processed, is.numeric) &
4                                     names(xdata_task2_
      car_processed)
      != "claimcst0"]
5 cat_preds_car_gam <- names(xdata_task2_car_processed)[apply(
      xdata_task2_car_processed, is.factor)]
6
7 gam_terms_str_car <- c()
8 for (pred in num_preds_car_gam) { gam_terms_str_car <- c(gam_
      terms_str_car, paste0("s(", pred, ", df=4)")) }
9 for (pred in cat_preds_car_gam) { gam_terms_str_car <- c(gam_
      terms_str_car, pred) }
10 gam_formula_full_car_str <- paste("claimcst0 ~", paste(gam_terms
      _str_car, collapse=" + "))
11 gam_formula_full_car <- as.formula(gam_formula_full_car_str)
12
13 # Fit GAM on the processed dataset (xdata_task2_car_processed)
      for plotting

```



```

14 gam_model_car_full <- gam(gam_formula_full_car, data=xdata_task2
   _car_processed)
15 # summary(gam_model_car_full)
16
17 # Plot partial effects (adjust mfrow based on number of terms)
18 # num_plots <- length(num_preds_car_gam) + length(cat_preds_car_
   gam)
19 # par(mfrow=c(ceiling(num_plots/3), 3))
20 # plot(gam_model_car_full, se=TRUE, col="blue", ask=TRUE)
21 # Comment: For each s() term: is it linear, U-shaped, increasing
   /decreasing?
22 # Example from marking guide: numclaims linear, veh_value non-
   linear for small values, exposure non-linear.

```

3.1.10 2e: Specify appropriate GAM and compute testMSE

Question Use your conclusions in 2d to specify an appropriate GAM-model. Compute the testMSE for this GAM-model.

Base R Solution (Integrating with CV loop from 2c)

```

1 # Based on marking scheme plots/conclusions: s(veh_value), s(
   exposure), linear numclaims, and all factors.
2 # Assuming 'numclaims' is numeric, others like 'veh_body', 'veh_
   age', 'gender', 'area', 'agecat' are factors.
3 gam_formula_refined_car_str <- "claimcst0 ~ s(veh_value, df=4) +
   s(exposure, df=4) + numclaims + veh_body + veh_age + gender
   + area + agecat"
4 gam_formula_refined_car <- as.formula(gam_formula_refined_car_
   str)
5
6 # To compute CV testMSE, add this to the CV loop from 2c:
7 # (Inside the loop for k_idx_cv in 1:K_cv)
8 # Ensure factors have same levels in train_df_cv and test_df_cv,
   or handle carefully.
9 # tryCatch({
10 #   gam_refined_cv_car <- gam(gam_formula_refined_car, data=
   train_df_cv)
11 #   pred_gam_refined_cv_car <- predict(gam_refined_cv_car,
   newdata=test_df_cv)
12 #   # Add a new column to mse_matrix_car if not already
   defined:
13 #   # mse_matrix_car[k_idx_cv, "RefinedGAM"] <- mean((test_df_
   cv[[response_car]] - pred_gam_refined_cv_car)^2, na.rm=TRUE)
14 # }, error = function(e) { print(paste("Error with GAM in fold
   ", k_idx_cv, ":", e$message))})
15 # } %% End of loop %
16 # # After loop:
17 # # cv_avg_msес_car <- colMeans(mse_matrix_car, na.rm=TRUE)
18 # # print(cv_avg_msес_car["RefinedGAM"])
19 # # Marking guide: Full GAM had testMSE ~12,243,092. Full linear
   reg ~12,277,335. GAM slightly better.

```

Task 3: Analysis of dataCar Data (Original) - Predicting clm *Note: For this task, use the original dataCar, not the version from Task 2.*

3.1.11 3a: Data prep, descriptive stats for clm

Question Start by removing variables *X.OBSTAT*, *numclaims* and *claimcst0*. Compute and interpret appropriate cross-tables of *clm* and the categorical variables. Also compute and interpret well-chosen descriptive statistics (such as central tendencies and measures of variation) for the numerical variables for each of the two categories of *clm*. Are some of the categories of the categorical variables more interesting than others? Based on this, do you see any promising predictors for *clm*? Mention a drawback with such pairwise comparisons.

Base R Solution

```
1 data(dataCar) # Reload original
2 xdata_task3_orig_car <- dataCar[, !(names(dataCar) %in% c("X_
  OBSTAT_", "numclaims", "claimcst0"))]
3 xdata_task3_orig_car$clm <- as.factor(xdata_task3_orig_car$clm)
  # Target variable
4
5 # Ensure other categoricals are factors
6 xdata_task3_orig_car$veh_body <- as.factor(xdata_task3_orig_car$
  veh_body)
7 xdata_task3_orig_car$veh_age <- as.factor(xdata_task3_orig_car$
  veh_age)
8 xdata_task3_orig_car$gender <- as.factor(xdata_task3_orig_car$
  gender)
9 xdata_task3_orig_car$area <- as.factor(xdata_task3_orig_car$area
  )
10 xdata_task3_orig_car$agecat <- as.factor(xdata_task3_orig_car$
  agecat)
11
12 # cat_predictors_task3 <- names(xdata_task3_orig_car)[sapply(
  xdata_task3_orig_car, is.factor) & names(xdata_task3_orig_car
  )!="clm"]
13 # num_predictors_task3 <- names(xdata_task3_orig_car)[sapply(
  xdata_task3_orig_car, is.numeric)]
14
15 # for (pred in cat_predictors_task3) {
16 #   cat("\nCross-table for clm vs", pred, ":\n")
17 #   tbl <- table(Claim=xdata_task3_orig_car$clm, Predictor=xdata
  _task3_orig_car[[pred]])
18 #   print(tbl)
19 #   print(round(prop.table(tbl, margin = 2)100,1)) # Column
  percentages (P(Claim | Predictor Level))
20 # }
21 # # Interpretation: Look for levels of categorical predictors
  with notably higher/lower claim rates.
22 # # E.g., "BUS and MCARA vehicle types show higher claim rates
  (19% and 11% vs. overall 6.8%)."
23
24 # for (pred_num in num_predictors_task3) {
25 #   cat("\nSummary of", pred_num, "by clm status:\n")
26 #   print(by(xdata_task3_orig_car[[pred_num]], xdata_task3_orig_
  car$clm, summary))
27 #   # boxplot(as.formula(paste(pred_num, "~ clm")), data=xdata_
  task3_orig_car, main=paste(pred_num, "by Claim"))
28 # }
29 # # Interpretation: "Policies with claims have, on average,
  higher veh_value and exposure."
30 # # Promising predictors: veh_body, agecat, veh_value, exposure
  (from marking guide).
```

```

31 # # Drawback of pairwise: Does not account for confounding or
    interaction effects between predictors.

```

3.1.12 3b: Logistic regression for clm

Question Based on your hypotheses from 3a, use logistic regression based on all or a subset of the predictors to predict *clm*. Interpret the estimated coefficients.

Base R Solution

```

1 # Assuming xdata_task3_orig_car from 3a.
2 # Predictors from marking guide: veh_body, agecat, veh_value,
  exposure
3 logit_model_clm_car_task3 <- glm(clm ~ veh_body + agecat + veh_
  value + exposure,
4                                   data=xdata_task3_orig_car,
                                   family=binomial)
5 summary(logit_model_clm_car_task3)
6 # Interpretation (odds ratios):
7 # coefs <- coef(logit_model_clm_car_task3)
8 # odds_ratios <- exp(coefs)
9 # print(odds_ratios)
10 # Example: For agecat2, odds of claim are exp(coef_agecat2) (e.g
   ., 0.81) times odds for agecat1 (baseline),
11 # holding other variables constant.
12 # For veh_value (continuous), a one-unit increase (10000 dollars
   ) multiplies odds of claim by exp(0.06) ~ 1.06.

```

3.1.13 3c: Validation set for logistic regression, thresholding

Question Modify model if you believe it to be sensible. Use the validation set approach with a 50/50 training/test split to evaluate the predictions from the logistic regression. Choose a threshold for when an estimated probability should lead to a prediction of a claim (imagine cost $FN \neq FP$). Compute a confusion matrix and argue for your choice of threshold. Set the seed to 1234 (Marking guide seed was different, here 1234).

Base R Solution

```

1 # Using xdata_task3_orig_car from 3a
2 set.seed(1234)
3 n_task3 <- nrow(xdata_task3_orig_car)
4 train_idx_task3 <- sample(1:n_task3, floor(n_task3/2))
5 train_data_task3 <- xdata_task3_orig_car[train_idx_task3,]
6 test_data_task3 <- xdata_task3_orig_car[-train_idx_task3,]
7
8 logit_train_task3 <- glm(clm ~ veh_body + agecat + veh_value +
  exposure,
9                                   data=train_data_task3, family=binomial)
10 probs_test_task3 <- predict(logit_train_task3, newdata=test_data
   _task3, type="response")
11
12 # Threshold choice (e.g., 0.07 as in marking scheme to catch
   more actual claims)
13 # Motivation: Overall claim rate is low (~6.8%). A standard 0.5
   threshold will predict
14 # very few claims. To reduce False Negatives (costly if missing
   an actual claim),

```

```

15 # lower the threshold. This increases True Positives but also
    False Positives.
16 chosen_threshold_task3 <- 0.07
17 pred_class_task3_logistic <- ifelse(probs_test_task3 > chosen_
    threshold_task3,
18                                     levels(train_data_task3$clm)
                                     [2], # "1" or "Yes"
19                                     levels(train_data_task3$clm)
                                     [1]) # "0" or "No"
20 pred_class_task3_logistic <- factor(pred_class_task3_logistic,
    levels=levels(train_data_task3$clm))
21
22 conf_matrix_task3_logistic <- table(Actual = test_data_task3$clm
    , Predicted = pred_class_task3_logistic)
23 print(conf_matrix_task3_logistic)
24 # print(prop.table(conf_matrix_task3_logistic, margin=1)) # Row
    percentages (Sensitivity, Specificity)
25 # Interpretation: "With a threshold of 0.07, we correctly
    identify [Sensitivity]% of actual claims,
26 # while incorrectly classifying [1-Specificity]% of non-claims
    as claims. This trade-off was
27 # chosen to prioritize detecting claims."

```

3.1.14 3d: Boosted trees for clm

Question Use boosted trees to predict *clm* and evaluate the predictions with the same approach as in 3c.

Base R Solution (using gbm)

```

1 library(gbm)
2 # Using train_data_task3, test_data_task3, chosen_threshold_
    task3 from 3c
3 # GBM requires numeric 0/1 response for bernoulli
4 train_data_task3$clm_numeric <- as.numeric(train_data_task3$clm)
    - 1
5 test_data_task3$clm_numeric <- as.numeric(test_data_task3$clm)
    - 1
6
7 formula_gbm_task3 <- clm_numeric ~ veh_body + agecat + veh_value
    + exposure
8
9 set.seed(1234) # For reproducibility
10 boost_model_task3 <- gbm(formula_gbm_task3,
11                           data=train_data_task3,
12                           distribution="bernoulli",
13                           n.trees=100, # Marking scheme suggests
                                100 trees
14                           interaction.depth=4, # Example, tune
                                this
15                           shrinkage=0.01)      # Example, tune
                                this
16
17 probs_boost_task3_test <- predict(boost_model_task3, newdata=
    test_data_task3,
18                                     n.trees=100, type="response")
19 pred_class_boost_task3 <- ifelse(probs_boost_task3_test > chosen
    _threshold_task3, 1, 0)
20

```

```

21 conf_matrix_boost_task3 <- table(Actual = test_data_task3$clm_
    numeric, Predicted = pred_class_boost_task3)
22 print(conf_matrix_boost_task3)
23 # print(prop.table(conf_matrix_boost_task3, margin=1))
24 # Compare with logistic regression from 3c. Marking scheme: GBM
    improved TP rate (0.67 vs 0.59).

```

3.2 Exam Spring 2022

Overall Datasets: OJ (ISLR), Computers (Ecdat)

Task 1: Analysis of OJ Data - Predicting Purchase

(Assume *LoyalCH* unavailable except for 1f)

3.2.1 1a: Data preparation and splitting

Question Remove the variable *LoyalCH*. If necessary, recode categorical variables as factors and remove variables that cannot be used in the analysis. Base your reasoning to do this on the help function and by looking at the data. Split the data in a 50/50 split to create a training and a test dataset. Use the seed 8655, when you split the data, `set.seed(8655)`.

Base R Solution (basert på løsningsforslag)

```

1 library(ISLR)
2 data(OJ)
3
4 # Remove LoyalCH and StoreID (StoreID is same as STORE after
    factor conversion)
5 oj_data_prep <- OJ[, !(names(OJ) %in% c("LoyalCH", "StoreID"))]
6
7 # Identify variables to convert to factor (<= 5 unique values in
    solution)
8 # card_func <- function(x) length(table(x))
9 # len_oj <- sapply(oj_data_prep, card_func)
10 # print(len_oj)
11 # Purchase:2, SpecialCH:2, SpecialMM:2, Store7:2, STORE:5 should
    become factors
12 # WeekofPurchase:52, PriceCH:10, PriceMM:8, DiscCH:12, DiscMM
    :12, etc. are numeric
13
14 for (col_name in names(oj_data_prep)) {
15   if (length(unique(oj_data_prep[[col_name]])) <= 5) {
16     oj_data_prep[[col_name]] <- as.factor(oj_data_prep[[col_name]]
17   )
18 }
19 # Verify: sapply(oj_data_prep, class)
20
21 # Split data
22 set.seed(8655) # As specified in this exam version
23 n_oj <- nrow(oj_data_prep)
24 train_idx_oj <- sample(1:n_oj, floor(n_oj/2))
25 train_oj <- oj_data_prep[train_idx_oj, ]
26 test_oj <- oj_data_prep[-train_idx_oj, ]
27
28 # print(paste("Train OJ dimensions:", paste(dim(train_oj),
    collapse="x")))

```

```

29 # print(paste("Test OJ dimensions:", paste(dim(test_oj),
      collapse="x")))
30 # Motivation: LoyalCH removed as per instruction. StoreID
      removed as STORE factor captures same info.
31 # Variables with few unique values and no inherent order
      converted to factors.

```

Tidyverse Solution

```

1 library(ISLR); library(dplyr); library(purrr)
2 data(OJ)
3
4 oj_data_prep_tidy <- OJ %>%
5   select(-LoyalCH, -StoreID) %>%
6   mutate(across(where(~length(unique(.)) <= 5), as.factor))
7
8 set.seed(8655)
9 train_idx_oj_tidy <- sample(1:nrow(oj_data_prep_tidy), floor(
      nrow(oj_data_prep_tidy)/2))
10 train_oj_tidy <- oj_data_prep_tidy[train_idx_oj_tidy, ]
11 test_oj_tidy <- oj_data_prep_tidy[-train_idx_oj_tidy, ]
12 # glimpse(train_oj_tidy)

```

3.2.2 1b: Descriptive statistics for promising predictors

Question Use descriptive statistics to detect promising predictors for *Purchase*. Present the most interesting results as tables and graphs and comment on them.

Base R / Tidyverse Solution (Conceptual - Løsningsforslag fokuserer på plott og 'by')

```

1 # Assuming train_oj from 1a
2 # Visual exploration (Base R)
3 # par(mfrow=c(1,2)) # Example for specific plots
4 # plot(Purchase ~ STORE, data=train_oj, main="Purchase by Store
      ")
5 # boxplot(PriceDiff ~ Purchase, data=train_oj, main="PriceDiff
      by Purchase")
6 # graphics.off()
7
8 # Numerical summaries (Base R)
9 # cat("Mean PctDiscCH by Purchase:\n")
10 # print(by(train_oj$PctDiscCH, train_oj$Purchase, mean, na.rm=
      TRUE))
11 # cat("Mean PctDiscMM by Purchase:\n")
12 # print(by(train_oj$PctDiscMM, train_oj$Purchase, mean, na.rm=
      TRUE))
13 # cat("Table of SpecialMM by Purchase:\n")
14 # print(table(train_oj$SpecialMM, train_oj$Purchase))
15
16 # Tidyverse for summaries
17 # library(dplyr)
18 # train_oj %>% group_by(Purchase) %>%
19 #   summarise(across(c(PriceDiff, PctDiscCH, PctDiscMM),
20 #     list(mean=mean, median=median), na.rm=TRUE)
21 #   )
22 # train_oj %>% count(STORE, Purchase) %>% group_by(STORE) %>%
23 #   mutate(prop = n/sum(n))
24
25 # Comments (basert på løsningsforslag):

```

```

24 # - STORE: Variation in brand purchased by store.
25 # - PriceDiff: Higher (MM more expensive) when CH bought.
26 # - PctDiscCH/PctDiscMM: Discounts seem to influence choice.
27 # - SpecialMM: Appears differently distributed between CH/MM
    purchases.

```

3.2.3 1c: Logistic regression

Question Fit a logistic regression of *Purchase* on all the variables you found promising in 1b, evaluate the predictions using test accuracy and a confusion matrix. If you, in the process, detect that you would modify your model, you should do so and explain why. Use the threshold 0.5 when you go from estimated probability to a prediction.

Base R Solution (Løsningsforslag bruker PriceDiff + STORE)

```

1 # Assuming train_oj, test_oj from 1a
2 # Løsningsforslag ender opp med PriceDiff og STORE etter ha
    vurdert PctDisc-variablene som ikke-signifikante.
3 # Vi starter med de 'lovende' og fjerner evt. ikke-signifikante.
4 # Initial promising: STORE, PriceDiff, PctDiscCH, PctDiscMM,
    SpecialMM
5 # logit_oj_initial <- glm(Purchase ~ STORE + PriceDiff +
    PctDiscCH + PctDiscMM + SpecialMM,
6 #                         data=train_oj, family=binomial)
7 # summary(logit_oj_initial)
8 # % If PctDiscCH, PctDiscMM, SpecialMM are not significant,
    remove them. %
9
10 # Final model from solution:
11 logit_oj_final <- glm(Purchase ~ PriceDiff + STORE, data=train_
    oj, family=binomial)
12 # print(summary(logit_oj_final))
13
14 # Predictions on test set
15 probs_logit_oj_test <- predict(logit_oj_final, newdata=test_oj,
    type="response")
16 pred_logit_oj_test <- ifelse(probs_logit_oj_test > 0.5, "MM", "
    CH") # OJ levels: CH, MM
17 pred_logit_oj_test <- factor(pred_logit_oj_test, levels = levels
    (test_oj$Purchase))
18
19 # Confusion matrix and accuracy
20 conf_matrix_logit_oj <- table(Actual=test_oj$Purchase, Predicted
    =pred_logit_oj_test)
21 print(conf_matrix_logit_oj)
22 accuracy_logit_oj <- sum(diag(conf_matrix_logit_oj)) / sum(conf_
    matrix_logit_oj)
23 cat("Test Accuracy (Logistic Regression):", accuracy_logit_oj, "
    \n")
24 # prop_table_logit_oj <- prop.table(conf_matrix_logit_oj, margin
    =1)
25 # print(round(prop_table_logit_oj,3))
26 # Interpretation: The model predicts [value]% of CH purchases
    correctly and [value]% of MM purchases.

```


3.2.4 1d: Classification tree

Question Fit a classification tree, using all variables (except the one(s) you removed in 1a), and plot it. Give an example of how to interpret a branch of the tree. Also, use the same evaluation measures as in 1c and evaluate the predictions. Use the threshold 0.5.

Base R Solution (using tree)

```
1 library(tree)
2 # Assuming train_oj, test_oj from 1a (with all original
  predictors kept, except LoyalCH, StoreID)
3 tree_oj <- tree(Purchase ~ ., data=train_oj)
4 # summary(tree_oj)
5 # plot(tree_oj); text(tree_oj, pretty=0)
6
7 # Interpretation example (basert p l sningsforslagets
  trefigur):
8 # "If STORE is 'ae' (0 or 4 in factor levels) AND PriceDiff <
  -0.35 AND WeekofPurchase < 258.5,
9 # then the prediction is MM." (Values vil variere basert p
  seed).
10
11 # Predictions on test set
12 # predict() for tree with type="class" gives direct class
  predictions.
13 # For 0.5 threshold on probabilities, get probabilities first:
14 probs_tree_oj_test <- predict(tree_oj, newdata=test_oj, type="
  vector") # Gives matrix of probs
15 # Assuming Purchase is factor with levels CH, MM. probs_tree_oj_
  test[,2] is P(MM)
16 pred_tree_oj_test <- ifelse(probs_tree_oj_test[,2] > 0.5, "MM",
  "CH")
17 pred_tree_oj_test <- factor(pred_tree_oj_test, levels = levels(
  test_oj$Purchase))
18
19 conf_matrix_tree_oj <- table(Actual=test_oj$Purchase, Predicted=
  pred_tree_oj_test)
20 print(conf_matrix_tree_oj)
21 accuracy_tree_oj <- sum(diag(conf_matrix_tree_oj)) / sum(conf_
  matrix_tree_oj)
22 cat("Test Accuracy (Classification Tree):", accuracy_tree_oj, "\
  n")
23 # prop_table_tree_oj <- prop.table(conf_matrix_tree_oj, margin
  =1)
24 # print(round(prop_table_tree_oj,3))
```

3.2.5 1e: Cross-validation for optimal threshold for tree

Question In 1c and 1d you have used the threshold 0.5. Use cross-validation to find the threshold that maximizes the accuracy for the tree. Evaluate predictions from the tree using the best threshold. Are the predictions improved? Use the seed 4598 when you split the data for CV (this implies an inner CV split on the original training data).

Base R Solution (Validation set for threshold tuning on original training data)

```
1 # Using original train_oj and test_oj from 1a.
```



```

2 # To find optimal threshold, we need a validation set, split
  from original train_oj.
3 set.seed(4598) # Seed for splitting train_oj further
4 n_train_oj <- nrow(train_oj)
5 val_idx_oj <- sample(1:n_train_oj, floor(n_train_oj/2))
6 train2_oj <- train_oj[val_idx_oj, ]
7 valid_oj <- train_oj[-val_idx_oj, ]
8
9 # Fit tree on train2_oj
10 tree_for_thresh <- tree(Purchase ~ ., data=train2_oj)
11 probs_valid_oj <- predict(tree_for_thresh, newdata=valid_oj,
  type="vector")[,2] # P(MM)
12
13 thresholds <- seq(0.1, 0.9, by=0.05)
14 accuracies_thresh <- numeric(length(thresholds))
15
16 for (i in 1:length(thresholds)) {
17   th <- thresholds[i]
18   pred_class_thresh <- ifelse(probs_valid_oj > th, "MM", "CH")
19   pred_class_thresh <- factor(pred_class_thresh, levels = levels
    (valid_oj$Purchase))
20   accuracies_thresh[i] <- mean(pred_class_thresh == valid_oj$
    Purchase)
21 }
22
23 # plot(thresholds, accuracies_thresh, type="b", xlab="Threshold
  ", ylab="Validation Accuracy")
24 best_threshold_idx <- which.max(accuracies_thresh)
25 optimal_threshold_oj <- thresholds[best_threshold_idx]
26 cat("Optimal threshold found:", optimal_threshold_oj, "\n")
27
28 # Evaluate original tree_oj (fit on full train_oj) on test_oj
  using this optimal_threshold_oj
29 probs_tree_fulltrain_test <- predict(tree_oj, newdata=test_oj,
  type="vector")[,2] # P(MM)
30 pred_tree_optimal_thresh <- ifelse(probs_tree_fulltrain_test >
  optimal_threshold_oj, "MM", "CH")
31 pred_tree_optimal_thresh <- factor(pred_tree_optimal_thresh,
  levels=levels(test_oj$Purchase))
32
33 conf_matrix_tree_optimal <- table(Actual=test_oj$Purchase,
  Predicted=pred_tree_optimal_thresh)
34 print(conf_matrix_tree_optimal)
35 accuracy_tree_optimal <- sum(diag(conf_matrix_tree_optimal)) /
  sum(conf_matrix_tree_optimal)
36 cat("Test Accuracy (Tree with Optimal Threshold):", accuracy_
  tree_optimal, "\n")
37 # Compare accuracy_tree_optimal with accuracy_tree_oj (from 1d
  using 0.5 threshold)
38 # Marking guide's plot actually uses the same tree (tree1) for
  validation, which is fine.
39 # Their plot shows threshold around 0.4-0.6 being optimal.
  Accuracy doesn't improve much.

```

3.2.6 1f: Analysis with LoyalCH

Question In a) you removed the variable *LoyalCH*. Imagine now that this variable is actually available at the time of prediction. By augmenting your analysis with this variable both for the logistic regression and the classification

tree, conclude if this variable contribute to predicting *Purchase*.

Solution Outline Repeat steps 1a-1d, but *KEEP LoyalCH* in the dataset. Compare test accuracies and model summaries.

```
1 # 1. Data Prep: Start with original OJ, remove only StoreID.
  Convert factors. Split.
2 #   oj_with_loyalch <- OJ[, names(OJ) != "StoreID"]
3 #   ... convert factors, split into train_oj_L, test_oj_L ...
4
5 # 2. Logistic Regression with LoyalCH:
6 #   logit_oj_L <- glm(Purchase ~ PriceDiff + STORE + LoyalCH,
7 #                     data=train_oj_L, family=binomial)
8 #   % (Or include all promising predictors + LoyalCH) %
9 #   summary(logit_oj_L) % Check significance of LoyalCH %
10 #   % Calculate test accuracy and confusion matrix on test_oj_L
11 #   %
12 #   % Compare accuracy with result from 1c. %
13
14 # 3. Classification Tree with LoyalCH:
15 #   tree_oj_L <- tree(Purchase ~ ., data=train_oj_L)
16 #   summary(tree_oj_L) % See if LoyalCH is used in splits %
17 #   plot(tree_oj_L); text(tree_oj_L, pretty=0)
18 #   % Calculate test accuracy and confusion matrix on test_oj_L
19 #   %
20 #   % Compare accuracy with result from 1d. %
21
22 # Conclusion: If LoyalCH is significant in logistic regression
23 # and/or used prominently
24 # in the tree, and if test accuracies improve notably, then
25 # LoyalCH contributes.
26 # (LoyalCH is expected to be a very strong predictor).
```

Task 2: Analysis of Computers Data - Predicting price

3.2.7 2a: Data preparation and splitting

Question Just as in 1a; if necessary, recode categorical variables as factors and remove variables that cannot be used in the analysis. Base your reasoning to do this on the help function and by looking at the data. Split the data in a 50/50 split to create a training and a test dataset. Use the seed 4598, when you split the data, *set.seed(4598)*.

Base R Solution (basert på løsningsforslag)

```
1 library(Ecdat)
2 data(Computers)
3
4 # Recode categoricals (<= 5 unique values, from marking scheme)
5 # Inspect: sapply(Computers, function(x) length(unique(x)))
6 # screen (3), cd (2), multi (2), premium (2) should become
7 # factors
8 cols_to_factor_comp <- c("screen", "cd", "multi", "premium")
9 for (col in cols_to_factor_comp) {
10   Computers[[col]] <- as.factor(Computers[[col]])
11 }
12 # sapply(Computers, class) # Verify
```

```

12
13 # Split data
14 set.seed(4598)
15 n_comp <- nrow(Computers)
16 train_idx_comp <- sample(1:n_comp, floor(n_comp/2))
17 train_comp <- Computers[train_idx_comp, ]
18 test_comp <- Computers[-train_idx_comp, ]
19 # print(paste("Train Computers dimensions:", paste(dim(train_
    comp), collapse="x")))

```

Tidyverse Solution

```

1 library(Ecdat); library(dplyr)
2 data(Computers)
3 Computers_prep_tidy <- Computers %>%
4   mutate(across(all_of(c("screen", "cd", "multi", "premium")),
5     as.factor))
6
7 set.seed(4598)
8 train_idx_comp_tidy <- sample(1:nrow(Computers_prep_tidy), floor
9   (nrow(Computers_prep_tidy)/2))
10 train_comp_tidy <- Computers_prep_tidy[train_idx_comp_tidy, ]
11 test_comp_tidy <- Computers_prep_tidy[-train_idx_comp_tidy, ]
12 # glimpse(train_comp_tidy)

```

3.2.8 2b: Descriptive statistics for promising predictors

Question *Use descriptive statistics to find promising predictors. Comment on your observations.*

Base R / Tidyverse Solution (Conceptual - Løsningsforslag bruker

```

    plot(price ~., data=train))
1 # Assuming train_comp from 2a
2 # Visual exploration (Base R)
3 # par(mfrow=c(3,3)) # Adjust layout as needed
4 # plot(price ~ ., data=train_comp) # As per marking guide
5 # graphics.off()
6 # Comment: "Speed, hd, ram show positive trends with price.
7   Screen17 has higher price than screen15.
8   # cd=yes, multi=yes, premium=yes associated with higher price.
9   Ads and trend also show relationships."
10
11 # Tidyverse for specific plots
12 # library(ggplot2)
13 # ggplot(train_comp, aes(x=speed, y=price)) + geom_point() +
14   geom_smooth()
15 # ggplot(train_comp, aes(x=ram, y=price)) + geom_point() + geom_
16   smooth()
17 # ggplot(train_comp, aes(x=screen, y=price)) + geom_boxplot()

```

3.2.9 2c: Linear regression for price

Question *Run a linear regression on all explanatory variables. Interpret some of the coefficients. Evaluate the prediction performance on an appropriate measure.*

Base R Solution

```
1 # Assuming train_comp, test_comp from 2a
2 lm_comp_full <- lm(price ~ ., data=train_comp)
3 summary_lm_comp <- summary(lm_comp_full)
4 print(summary_lm_comp)
5 # Interpretation: e.g., "A 1MHz increase in speed is associated
  # with an expected price
6 # increase of [coef_speed] dollars, holding other factors
  # constant (p=[p_val])."
7 # "Having a CD-ROM (cdyes) is associated with an expected price
  # increase of
8 # [coef_cdyes] compared to no CD-ROM, holding others constant (p
  # =[p_val])."
9
10 # Evaluate performance: Test MSE
11 pred_lm_comp_test <- predict(lm_comp_full, newdata=test_comp)
12 mse_lm_comp_test <- mean((test_comp$price - pred_lm_comp_test)
  ^2, na.rm=TRUE)
13 cat("Test MSE (Full LM for Computers):", mse_lm_comp_test, "\n")
14 # Marking guide result: MSE1 = 72200
```

3.2.10 2d: Linear regression of log(price)

Question Investigate if the predictions of *price* would be better if we use a linear regression of *log(price)* on the other variables.

Base R Solution

```
1 # Create log(price) variable, ensuring it's available in train
  # and test
2 # train_comp_log <- train_comp
3 # test_comp_log <- test_comp
4 # train_comp_log$log_price <- log(train_comp_log$price)
5 # test_comp_log$log_price <- log(test_comp_log$price) # For
  # actual values if needed, but predict() on log scale
6
7 # Fit model on training data using log_price, excluding original
  # price
8 # lm_comp_log <- lm(log(price) ~ . - price, data=train_comp) #
  # Simpler way
9 lm_comp_log <- lm(log(price) ~ speed + hd + ram + screen + cd +
  multi + premium + ads + trend,
10                   data=train_comp)
11 summary(lm_comp_log)
12
13 # Predict on test set (predictions will be on log scale)
14 pred_log_price_test <- predict(lm_comp_log, newdata=test_comp)
15 # Transform predictions back to original price scale
16 pred_price_from_log_test <- exp(pred_log_price_test)
17
18 # Calculate Test MSE on original price scale
19 mse_loglm_comp_test <- mean((test_comp$price - pred_price_from_
  log_test)^2, na.rm=TRUE)
20 cat("Test MSE (Log-Linear LM for Computers):", mse_loglm_comp_
  test, "\n")
21 # Marking guide result: MSE2 = 72610.7. In this case, log
  # transform didn't improve MSE.
```

3.2.11 2e: GAM for price

Question In 2d you fitted a model where **price** is a particular nonlinear function of the other variables. You should now investigate another non-linear models. First, fit a GAM-model, plot the result and evaluate the predictions. Is there a reason to not allow for some variables to have a non-linear relationship with **price**?

Base R Solution (using gam)

```
1 library(gam)
2 # Assuming train_comp, test_comp from 2a
3 # Formula for GAM: s() for continuous, direct for factors
4 # From marking guide: s(speed)+s(hd)+s(ram)+screen+cd+multi+
  premium+s(ads)+s(trend)
5 gam_comp1 <- gam(price ~ s(speed, df=4) + s(hd, df=4) + s(ram,
  df=4) + screen + cd + multi +
6               premium + s(ads, df=4) + s(trend, df=4), data=
  train_comp)
7 # summary(gam_comp1)
8
9 # Plot partial effects
10 # par(mfrow=c(3,3)) # To fit all plots, adjust as needed
11 # plot(gam_comp1, se=TRUE, col="blue", ask=TRUE)
12 # Reason to allow non-linear: Some relationships (e.g. price vs
  hd or speed) might
13 # exhibit diminishing returns or other non-linear patterns.
  Plots will confirm.
14
15 # Evaluate Test MSE
16 pred_gam_comp_test <- predict(gam_comp1, newdata=test_comp)
17 mse_gam_comp_test <- mean((test_comp$price - pred_gam_comp_test)
  ^2, na.rm=TRUE)
18 cat("Test MSE (GAM for Computers):", mse_gam_comp_test, "\n")
19 # Marking guide result: MSE3 = 60947.91. GAM significantly
  improved predictions.
```

3.2.12 2f: Explanation of backfitting for GAMs

Question Give a brief explanation of how backfitting works and in what situations it is possible to fit a generalized additive model with ordinary least square regression.

Explanation *Backfitting Algorithm:* For a GAM model $y_i = \beta_0 + \sum_{j=1}^p f_j(x_{ij}) + \epsilon_i$:

1. Initialize: $\hat{\beta}_0 = \bar{y}$, and $\hat{f}_j(x_{ij}) = 0$ for all j (or some initial guess, e.g., from linear model).
2. Cycle through predictors $j = 1, \dots, p, 1, \dots, p, \dots$ until convergence:
 - For current predictor j , compute partial residuals: $r_{ij} = y_i - \hat{\beta}_0 - \sum_{k \neq j} \hat{f}_k(x_{ik})$.
 - Update \hat{f}_j by fitting a smoother (e.g., spline, loess) to the partial residuals r_{ij} against x_{ij} . That is, $\hat{f}_j \leftarrow \text{smoother}(r_{.j} \sim x_{.j})$.
 - Often, \hat{f}_j are centered to ensure identifiability.
3. Convergence is reached when the functions \hat{f}_j do not change much between iterations.

GAM with OLS: A GAM can be fitted with OLS if each function $f_j(x_j)$ can be represented by a linear combination of a set of basis functions. For example:

- **Polynomial Regression:** $f_j(x_j) = \beta_{j1}x_j + \beta_{j2}x_j^2 + \dots$. The "basis functions" are x_j, x_j^2, \dots .
- **Regression Splines:** $f_j(x_j)$ is represented by a sum of spline basis functions (e.g., truncated power basis, B-spline basis). The model $y = \beta_0 + \sum_j \sum_l \beta_{jl} b_{jl}(x_j) + \epsilon$ is then linear in the β_{jl} coefficients and can be fit with OLS.

If non-parametric smoothers like smoothing splines (where λ is chosen by GCV) or local regression are used within backfitting, the overall procedure is not a single OLS fit, but each step of backfitting might involve an OLS-like weighted least squares or penalized least squares.

3.2.13 2g: Bagged trees for price

Question Use bagged trees to predict **price**. Evaluate the predictions. Compute variable importance measures and comment on the results. Explain the logic behind the measure for variable importance that you are using. (Hint: If the computation takes a long time you can reduce the number of trees fitted using the argument `ntree`.)

Base R Solution (using randomForest)

```

1 library(randomForest)
2 # Assuming train_comp, test_comp from 2a
3 # For bagging, mtry = number of predictors.
4 # price ~ . means all other columns are predictors.
5 # ncol(train_comp) - 1 (if price is a column, and all others are
   predictors)
6 # If train_comp has 10 predictors plus 'price' and 'logp' (from
   2d), p=10.
7 # Marking guide uses mtry=9, perhaps 'logp' was still in data.
   Assuming p=10 here.
8 num_predictors_comp <- ncol(train_comp) - 1 # Assuming price is
   the only response
9 if ("logp" %in% names(train_comp)) num_predictors_comp <- num_
   predictors_comp - 1
10
11 set.seed(4598) # For reproducibility
12 bag_comp <- randomForest(price ~ . - logp, data=train_comp, #
   Exclude logp if it exists
13                             mtry=num_predictors_comp,
14                             ntree=100, # Reduced from default 500
   for speed as hinted
15                             importance=TRUE)
16 # print(bag_comp)
17
18 # Evaluate predictions
19 pred_bag_comp_test <- predict(bag_comp, newdata=test_comp)
20 mse_bag_comp_test <- mean((test_comp$price - pred_bag_comp_test)
   ^2, na.rm=TRUE)
21 cat("Test MSE (Bagging for Computers):", mse_bag_comp_test, "\n"
   )
22 # Marking guide result: MSE4 = 28272.51 (significant improvement
   )
23
24 # Variable Importance

```

```

25 importance_bag_comp <- importance(bag_comp)
26 print(importance_bag_comp)
27 varImpPlot(bag_comp)
28 # Logic for IncNodePurity (default for regression):
29 # "IncNodePurity (or %IncMSE if type=1 is used) measures the
    total decrease in node impurity
30 # (RSS for regression trees) from splitting on that variable,
    averaged over all trees.
31 # A higher value indicates greater importance."
32 # Comment: "Ram, trend, speed, hd appear as most important
    variables."

```

3.2.14 2h: Explanation of bagging

Question Give a brief explanation of bagging.

Explanation ***Bagging (Bootstrap Aggregating):** Bagging is an ensemble learning technique designed to improve the stability and accuracy of machine learning algorithms, particularly those with high variance like decision trees. The process involves:*

1. **Bootstrap Sampling:** Create B new training datasets by sampling with replacement from the original training dataset. Each bootstrap sample has the same size as the original.
2. **Model Fitting:** Fit a separate model (e.g., a decision tree) independently to each of the B bootstrap samples. These trees are typically grown deep and not pruned.
3. **Aggregation:**
 - For **regression**, the predictions from the B models are averaged to get the final prediction.
 - For **classification**, a majority vote is taken among the B model predictions.

The main benefit of bagging is variance reduction. By averaging many (potentially noisy) models fit to slightly different versions of the data, the overall variance of the ensemble prediction is reduced, often leading to improved predictive performance.

3.3 Exam Spring 2023

Overall Dataset: Churn.csv (Provided on Canvas) Refer to variable definitions table at the end of this exam section.

Task 1: Predict Average Bill (bill_avg)

3.3.1 1a: Data preparation and splitting

Question If necessary, recode categorical variables as factors and remove variables that cannot be used in the analysis. Also, remove variables if you mean that they are unreasonable to include. One such reason could be that a variable is not available at the time the prediction is made but there might be other reasons too. Motivate your choices in words. Split the data in a 50/50 split to create a training and a test dataset. Use the seed 86554354, when you split the data, `set.seed(86554354)`.

Base R Solution (basert på løsningsforslag)

```
1 # Churn <- read.csv("Churn.csv") % \textit{(Assuming data is
  read)} %
2
3 # Remove id
4 xdata_churn <- Churn[, -1] % \textit{(Assuming 'id' is the first
  column)} %
5
6 # Recode specified variables to factors
7 factor_cols_churn <- c("is_tv_subscriber", "is_movie_package_
  subscriber", "churn")
8 for (col in factor_cols_churn) {
9   xdata_churn[[col]] <- as.factor(xdata_churn[[col]])
10 }
11 # print(sapply(xdata_churn, class))
12
13 # Split data
14 set.seed(86554354) # Seed from exam question (v1 from marking
  instructions might differ)
15 n_churn <- nrow(xdata_churn)
16 train_idx_churn <- sample(1:n_churn, floor(n_churn/2))
17 train_churn <- xdata_churn[train_idx_churn, ]
18 test_churn <- xdata_churn[-train_idx_churn, ]
19
20 # print(paste("Train Churn dimensions:", paste(dim(train_churn),
  collapse="x")))
21 # print(paste("Test Churn dimensions:", paste(dim(test_churn),
  collapse="x")))
22 # Motivation: 'id' is an identifier and not predictive. 'is_tv_
  subscriber',
23 # 'is_movie_package_subscriber', and 'churn' are categorical (0/
  1 originally)
24 # and explicitly converted to factors for proper handling by
  modeling functions.
25 # The question of whether 'churn' is available when predicting '
  bill_avg' is raised
26 # in the marking scheme; it's assumed available for this task.
```

Tidyverse Solution

```
1 # library(dplyr)
2 # Churn <- read.csv("Churn.csv")
3 #
4 # xdata_churn_tidy <- Churn %>%
5 #   select(-id) %>%
6 #   mutate(across(all_of(c("is_tv_subscriber", "is_movie_package_
  _subscriber", "churn")), as.factor))
7 #
8 # set.seed(86554354)
9 # train_idx_churn_tidy <- sample(1:nrow(xdata_churn_tidy), floor
  (nrow(xdata_churn_tidy)/2))
10 # train_churn_tidy <- xdata_churn_tidy[train_idx_churn_tidy, ]
11 # test_churn_tidy <- xdata_churn_tidy[-train_idx_churn_tidy, ]
12 # glimpse(train_churn_tidy)
```

3.3.2 1b: Descriptive methods for bill_avg

Question Use descriptive methods to find useful predictors for *bill_avg*. Write in words which R functions you used, present the most interesting results as tables and graphs and comment on them.

Base R / Tidyverse Solution (Conceptual - Marking guide uses

```
plot(bill_avg ~ ., data=train))  
1 # Assuming train_churn from 1a  
2 # Visual exploration (Base R as per marking guide)  
3 # par(mfrow=c(3,3)) # To fit all plots (9 predictors excluding  
  id and bill_avg itself)  
4 # plot(bill_avg ~ ., data=train_churn)  
5 # graphics.off()  
6 # Comments (based on marking guide comment):  
7 # "bill_avg seems to vary with all predictors, possibly with the  
  exception of  
8 # download_avg and upload_avg, where patterns are difficult to  
  see visually."  
9 # For factors like 'is_tv_subscriber', 'churn', boxplots are  
  generated by plot(y~.).  
10 # For numeric like 'subscription_age', scatterplots are  
    generated.  
11  
12 # Tidyverse for specific plots (examples)  
13 # library(ggplot2)  
14 # ggplot(train_churn, aes(x=is_tv_subscriber, y=bill_avg)) +  
  geom_boxplot()  
15 # ggplot(train_churn, aes(x=subscription_age, y=bill_avg)) +  
  geom_point() + geom_smooth()
```

3.3.3 1c: Best OLS for bill_avg

Question Produce the best possible predictions of *bill_avg* using standard linear regression (OLS) and evaluate them. Motivate your choices of variables, how you evaluate the predictions and the evaluation measure used.

Base R Solution (Comparing full model vs. model without download/upload_avg)

```
1 # Assuming train_churn, test_churn from 1a  
2 # Model 1: Full model with all predictors  
3 ols_full_churn <- lm(bill_avg ~ ., data=train_churn)  
4 # summary(ols_full_churn)  
5 pred_ols_full_churn <- predict(ols_full_churn, newdata=test_  
  churn)  
6 mse_ols_full_churn <- mean((test_churn$bill_avg - pred_ols_full_  
  churn)^2, na.rm=TRUE)  
7 cat("Test MSE (Full OLS for bill_avg):", mse_ols_full_churn, "\n  
  ")  
8 # Marking guide value: 117.8603  
9  
10 # Model 2: Reduced model (based on 1b, removing download_avg,  
  upload_avg)  
11 ols_reduced_churn <- lm(bill_avg ~ . - download_avg - upload_avg  
  , data=train_churn)  
12 # summary(ols_reduced_churn)  
13 pred_ols_reduced_churn <- predict(ols_reduced_churn, newdata=  
  test_churn)  
14 mse_ols_reduced_churn <- mean((test_churn$bill_avg - pred_ols_  
  reduced_churn)^2, na.rm=TRUE)  
15 cat("Test MSE (Reduced OLS for bill_avg):", mse_ols_reduced_  
  churn, "\n")  
16 # Marking guide value: 146.7376  
17  
18 # Motivation:
```

```

19 # Variables: Compared a full model with one excluding download_
    avg and upload_avg, based on visual
20 # inspection in 1b suggesting weak relationships.
21 # Evaluation Measure: Test MSE is used to assess predictive
    accuracy on unseen data.
22 # A lower Test MSE indicates better out-of-sample prediction.
23 # Conclusion: The full model performed much better (lower Test
    MSE) than the reduced model.

```

3.3.4 1d: LASSO regression for bill_avg

Question *Fit a LASSO regression with all variables. Here you should use the tools you have learned to find appropriate tuning parameters. Are you standardizing the predictors or not? Motivate your choice. Compare the estimated coefficients with the estimated coefficient from an OLS regression on all predictors. (Hint: Using categorical (factor) variables in LASSO, you will have to create dummy variables. `dummy_cols()` from `fastDummies` or `model.matrix()` can be used. `glmnet()` requires matrices as input).*

Base R Solution (using glmnet)

```

1 library(glmnet)
2 # Assuming train_churn, test_churn from 1a. Factors are already
    created.
3 # For glmnet, factors need to be converted to dummy variables.
4 # We use the processed data before splitting for model.matrix,
    then split X and y.
5 # xdata_churn was defined in 1a with factors.
6
7 # Create model matrix for predictors (converts factors to
    dummies)
8 x_matrix_churn <- model.matrix(bill_avg ~ . -1, data=xdata_churn
    ) # -1 removes intercept if glmnet adds it
9 y_vector_churn <- xdata_churn$bill_avg
10
11 # Split X and y matrices into train/test using train_idx_churn
    from 1a
12 trainX_churn <- x_matrix_churn[train_idx_churn, ]
13 trainY_churn <- y_vector_churn[train_idx_churn]
14 testX_churn <- x_matrix_churn[-train_idx_churn, ]
15 testY_churn <- y_vector_churn[-train_idx_churn] % \textit{(
    Actually test_churn$bill_avg)} %
16
17 # Determine lambda by cross-validation (alpha=1 for LASSO)
18 set.seed(86554354) % \textit{(Ensure consistent seed if CV is
    stochastic)} %
19 cv_lasso_churn <- cv.glmnet(trainX_churn, trainY_churn, alpha=1,
    standardize=TRUE)
20 # standardize=TRUE is default and recommended for LASSO/Ridge
    when variables are on different scales.
21 # Motivation for standardization: LASSO penalty applies equally
    to all coefficients. If predictors
22 # have different scales, those with larger scales might be
    unfairly penalized or vice versa.
23 # Standardization puts all predictors on a common scale.
24
25 best_lambda_lasso_churn <- cv_lasso_churn$lambda.min
26 # print(paste("Best lambda (min):", best_lambda_lasso_churn))
27

```

```

28 # Fit LASSO with optimal lambda on training data
29 lasso_model_churn <- glmnet(trainX_churn, trainY_churn, alpha=1,
    lambda=best_lambda_lasso_churn, standardize=TRUE)
30 lasso_coefs <- coef(lasso_model_churn)
31 # print("LASSO Coefficients:")
32 # print(lasso_coefs)
33
34 # OLS model from 1c (fitted on train_churn, which has factors
    not dummies)
35 # ols_full_churn <- lm(bill_avg ~ ., data=train_churn)
36 # ols_coefs <- coef(ols_full_churn)
37 # print("OLS Coefficients (factors handled by lm):")
38 # print(ols_coefs)
39
40 # Comparison:
41 # Lasso coefficients are shrunk towards zero. Some might be
    exactly zero.
42 # OLS coefficients (for the dummy variables created from factors
    ) will generally be larger in magnitude.
43 # Marking scheme showed factor variables were converted to 0/1
    numeric for glmnet,
44 # and coefficients were very similar, none set to zero,
    indicating LASSO restriction not very large.
45 # This implies xdata in marking scheme was xdata_num after
    converting factors to numeric.
46 # If xdata_num <- xdata; for(i in 1:length(fa)) xdata_num[,fa[i]
    ]] <- as.numeric(xdata_num[,fa[i]])-1
47 # Then use xdata_num for trainX, trainy.
48 # cbind(ols_full_churn$coefficients, lasso_coefs) % \textit{(
    Align names carefully for direct comparison)}%

```

3.3.5 1e: Evaluate LASSO model

Question Compute predictions for *bill_avg* with the model fitted in 1d) and evaluate them with an appropriate measure.

Base R Solution

```

1 # Assuming lasso_model_churn (fitted on trainX_churn) and testX_
    churn from 1d
2 pred_lasso_churn_test <- predict(lasso_model_churn, newx=testX_
    churn)
3 mse_lasso_churn_test <- mean((test_churn$bill_avg - pred_lasso_
    churn_test)^2, na.rm=TRUE)
4 cat("Test MSE (LASSO for bill_avg):", mse_lasso_churn_test, "\n"
    )
5 # Marking guide value: 117.8655 (very similar to OLS in this
    case)

```

3.3.6 1f: Regression tree for bill_avg

Question Fit a regression tree to *bill_avg*. Explain the choices you are making. Interpret the tree.

Base R Solution (using tree)

```

1 library(tree)
2 # Assuming train_churn from 1a (with factors)
3 # Choices: Using all predictors. Default tree.control parameters
    initially.

```

```

4 tree_bill_avg <- tree(bill_avg ~ ., data=train_churn)
5 # summary(tree_bill_avg)
6 # plot(tree_bill_avg)
7 # text(tree_bill_avg, pretty=0)
8
9 # Interpretation (based on marking scheme tree plot which is
  #   complex):
10 # - Follow branches based on predictor conditions.
11 # - Terminal nodes give the predicted average bill_avg for
  #   observations in that region.
12 # - Example from marking scheme plot:
13 #   "Lowest avg bill (4.589) for: download_avg < 333.2 AND
  #     download_over_limit < 0.5 AND
14 #     upload_avg < 15.25 AND subscription_age < 0.075 (almost new
  #     customers)."
15 #   "Highest avg bill (264.6) for: download_avg >= 333.2 AND is_
  #     tv_subscriber=0 (No)
16 #     AND download_avg >= 861.5."
17 # Tree structure highlights important variables and interaction-
  #   like effects through sequential splits.

```

3.3.7 1g: Evaluate regression tree

Question Predict *bill_avg* with the regression tree in 1f) and evaluate the predictions.

Base R Solution

```

1 # Assuming tree_bill_avg from 1f and test_churn from 1a
2 pred_tree_bill_avg_test <- predict(tree_bill_avg, newdata=test_
  churn)
3 mse_tree_bill_avg_test <- mean((test_churn$bill_avg - pred_tree_
  bill_avg_test)^2, na.rm=TRUE)
4 cat("Test MSE (Regression Tree for bill_avg):", mse_tree_bill_
  avg_test, "\n")
5 # Marking guide value: 120.2793 (comparable to OLS/LASSO,
  slightly worse)

```

3.3.8 1h: Random forest for bill_avg

Question Fit a random forest to *bill_avg*. Plot a variable importance measure for the predictors, interpret it, and, briefly, explain the measure. (If computations are too slow, reduce *ntree* or use smaller training data.)

Base R Solution (using randomForest)

```

1 library(randomForest)
2 # Assuming train_churn from 1a
3 set.seed(86554354) % \textit{(Consistent seed)} %
4 # Marking guide used ntree=50 for speed
5 rf_bill_avg <- randomForest(bill_avg ~ ., data=train_churn,
  ntree=50, importance=TRUE, na.action
  =na.roughfix)
6
7 # importance=TRUE is needed for varImpPlot. na.action handles
  #   potential NAs if any.
8
9 # Variable Importance
10 # print(importance(rf_bill_avg))
11 varImpPlot(rf_bill_avg, main="Variable Importance for bill_avg (
  Random Forest)")

```

```

12 # Explanation of IncNodePurity (default for regression):
13 # "IncNodePurity measures the total decrease in node impurity (
    RSS for regression trees)
14 # from splitting on that variable, averaged over all trees in
    the forest.
15 # A higher value indicates that the variable is more important
    for partitioning the data
16 # and improving the homogeneity of nodes regarding bill_avg."
17 # Interpretation: "Based on the plot, [predictor1] and [
    predictor2] appear most important.
18 # download_avg and upload_avg, which had unclear patterns in
    initial plots, now show high
19 # importance, contrasting with the single tree and OLS findings
    (from marking guide)."
```

3.3.9 1i: Evaluate random forest

Question Use the model in 1h) to predict *bill_avg* and evaluate the predictions.

Base R Solution

```

1 # Assuming rf_bill_avg from 1h and test_churn from 1a
2 pred_rf_bill_avg_test <- predict(rf_bill_avg, newdata=test_churn
    , na.action=na.pass)
3 mse_rf_bill_avg_test <- mean((test_churn$bill_avg - pred_rf_bill
    _avg_test)^2, na.rm=TRUE)
4 cat("Test MSE (Random Forest for bill_avg):", mse_rf_bill_avg_
    test, "\n")
5 # Marking guide value: 96.54472 (significantly lower than OLS,
    LASSO, single tree)
```

3.3.10 1j: Features of customers with high/low bill_avg

Question Based on your analysis in 1a)-1i, what are the features of customers with high and, respectively, low average bills?

Solution Outline (Textual) Synthesize findings from all models, focusing on consistent patterns and insights from the best performing model (Random Forest).

Task 2: Predict churn

3.3.11 2a: Data formatting, descriptive statistics for churn

Question Make sure that all variables are on the right format for your analysis. Use tables and graphs and common sense to remove variables that you think will not be helpful. Motivate your choices thoroughly. Use descriptive statistics to find promising predictors for *churn*.

Solution Outline (Base R / Tidyverse)

```

1 # Assuming xdata_churn from 1a (id removed, specified columns
    are factors)
2 # and train_churn (the training split)
3 # Data format should be okay from 1a if factors were made
    correctly.
```

```

4 # Remove unhelpful variables: 'bill_avg' might be considered a
  # consequence of churn or post-churn,
5 # or a predictor. Marking guide implies it's kept. No other
  # obvious removals for churn prediction.
6
7 # Descriptive stats on train_churn:
8 # Base R:
9 # cat_preds_churn <- names(train_churn)[sapply(train_churn, is.
  # factor) & names(train_churn)!="churn"]
10 # num_preds_churn <- names(train_churn)[sapply(train_churn, is.
  # numeric)]
11 # for (pred in c(cat_preds_churn, num_preds_churn)) {
12 #   if (is.factor(train_churn[[pred]])) {
13 #     print(paste("Table for churn vs", pred))
14 #     print(prop.table(table(Churn=train_churn$churn, Predictor=
  # train_churn[[pred]]), margin=2))
15 #   } else {
16 #     #boxplot(train_churn[[pred]] ~ train_churn$churn, main=
  # paste(pred, "by Churn"))
17 #   }
18 # }
19 # Tidyverse (as in marking guide for means):
20 # library(dplyr)
21 # train_churn %>%
22 #   mutate(churn_numeric = as.numeric(churn)-1) %>% # if churn
  # is factor "0","1" or "No","Yes"
23 #   group_by(churn) %>%
24 #   summarise(across(where(is.numeric) & !is.factor, list(mean=
  # mean)),
25 #             across(where(is.factor) & !matches("churn"), ~mean
  # (as.numeric(.)-1, na.rm=TRUE))) %>%
26 #   t() %>% print()
27 # plot(remaining_contract ~ churn, data=train_churn, main="
  # Remaining Contract by Churn")
28
29 # Promising predictors (from marking scheme):
30 # is_movie_package_subscriber (lower for churners), remaining_
  # contract (much lower for churners),
31 # download_avg (lower for churners), upload_avg (lower for
  # churners),
32 # download_over_limit (higher for churners).

```

3.3.12 2b: Bootstrap CI for $P(\text{churn}=1)$

Question Let Y be a stochastic variable equal to 1 if a customer churn and 0 otherwise and let $p = P(Y = 1)$ be the unconditional probability to churn. The first 50 (original, not from the training or test data) observations of the variable **churn** contains observations drawn from the stochastic variable Y . Use the bootstrap to compute a 95% confidence interval for p . Compare the standard approximation $\hat{p} \pm 1.96\sqrt{\hat{p}(1-\hat{p})/n}$ where \hat{p} is the sample fraction of churners and n is the number of observations.

Base R Solution

```

1 # Using original xdata_churn (after factor conversion in 1a)
2 # churn_numeric_orig <- as.numeric(xdata_churn$churn) - 1 #
  # Convert factor to 0/1
3 # first_50_churn <- churn_numeric_orig[1:50]
4 # n_obs_50 <- 50

```

```

5
6 # Bootstrap function for proportion
7 prop_churn_stat <- function(data, index) {
8   return(mean(data[index])) # Mean of 0/1 variable is proportion
9   of 1s
10 }
11 set.seed(86554354) # Consistent seed
12 boot_churn_prop <- boot(data=first_50_churn, statistic=prop_
13   churn_stat, R=1000)
14 # print(boot_churn_prop)
15
16 # Bootstrap Percentile CI
17 ci_perc_churn <- boot.ci(boot_churn_prop, type="perc")
18 print("Bootstrap Percentile CI for P(churn=1):")
19 print(ci_perc_churn)
20 # E.g., (0.68, 0.90) from marking guide.
21
22 # Standard Normal Approximation CI
23 p_hat_50 <- mean(first_50_churn)
24 se_p_hat_50 <- sqrt(p_hat_50 * (1 - p_hat_50) / n_obs_50)
25 ci_norm_approx_churn <- c(p_hat_50 - 1.96 * se_p_hat_50, p_hat_50
26   + 1.96 * se_p_hat_50)
27 print("Normal Approximation CI for P(churn=1):")
28 print(ci_norm_approx_churn)
29 # E.g., (0.689, 0.911) from marking guide.
30 # Comparison: "The results are very similar, indicating the
31   normal approximation works well for this sample size and
32   proportion."

```

3.3.13 2c: Logistic regression for churn

Question *Fit a logistic regression with all variables to `churn`. Interpret the coefficients in front of `is_tv_subscriber` and `is_movie_package_subscriber`. For all coefficients, is the sign as you expected?*

Base R Solution

```

1 # Assuming train_churn from 1a (with factors, including original
2   'churn' factor)
3 logit_churn_allvars <- glm(churn ~ ., data=train_churn, family=
4   binomial)
5 summary_logit_churn <- summary(logit_churn_allvars)
6 print(summary_logit_churn)
7
8 # Interpretation of coefficients:
9 # coefs_churn <- coef(logit_churn_allvars)
10 # odds_ratios_churn <- exp(coefs_churn)
11 # For is_tv_subscriber1 (assuming '1' means Yes):
12 # OR_tv = exp(coefs_churn["is_tv_subscriber1"]) e.g., exp(-1.75)
13   = 0.17
14 # "Holding other variables constant, customers with a TV
15   subscription have 0.17 times
16   the odds of churning compared to those without a TV
17   subscription (i.e., 83% lower odds)."
18 # For is_movie_package_subscriber1:
19 # OR_movie = exp(coefs_churn["is_movie_package_subscriber1"]) e.
20   g., exp(-0.06) = 0.94
21 # "Holding other variables constant, customers with a movie
22   package have 0.94 times

```



```

16 # the odds of churning compared to those without (i.e., 6% lower
    odds)."
17 # Expected signs:
18 # Negative for TV/Movie sub (loyalty), subscription_age (more
    invested), remaining_contract (locked in).
19 # Positive for service_failure_count, download_over_limit.
20 # Ambiguous for bill_avg, download_avg, upload_avg (could be
    good service or high cost).
21 # Check significance (p-values). Marking guide: bill_avg, upload
    _avg not significant.

```

3.3.14 2d: Evaluate logistic regression, consider removing variables

Question Use the logistic regression from 2c) to predict *churn*. Evaluate the predictions in an appropriate way. Are the predictions improved if you remove some variables from the model?

Base R Solution

```

1 # Assuming logit_churn_allvars from 2c (fitted on train_churn)
    and test_churn
2 probs_logit_test_churn <- predict(logit_churn_allvars, newdata=
    test_churn, type="response")
3 # Default threshold 0.5 for initial evaluation
4 pred_class_logit_churn_05 <- ifelse(probs_logit_test_churn >
    0.5,
5                                     levels(train_churn$churn)[2],
                                     levels(train_churn$churn)
                                     [1])
6 pred_class_logit_churn_05 <- factor(pred_class_logit_churn_05,
    levels=levels(train_churn$churn))
7
8 conf_matrix_logit_05 <- table(Actual=test_churn$churn, Predicted
    =pred_class_logit_churn_05)
9 print("Confusion Matrix (Full Model, Threshold 0.5):")
10 print(conf_matrix_logit_05)
11 # accuracy_logit_05 <- sum(diag(conf_matrix_logit_05)) / sum(
    conf_matrix_logit_05)
12 # print(paste("Accuracy (Full Model):", accuracy_logit_05))
13 # print("Proportions table (Full Model):")
14 # print(prop.table(conf_matrix_logit_05, margin=1))
15 # Marking scheme results for full model: (row proportions)
16 # FALSE TRUE
17 # 0 0.81606725 0.18393275
18 # 1 0.08262943 0.91737057 (Good at predicting non-churn, good at
    predicting churn when it occurs)
19
20 # Model with non-significant variables (bill_avg, upload_avg)
    removed:
21 logit_churn_reduced <- glm(churn ~ . -bill_avg -upload_avg, data
    =train_churn, family=binomial)
22 # summary(logit_churn_reduced)
23 probs_logit_reduced_test <- predict(logit_churn_reduced, newdata
    =test_churn, type="response")
24 pred_class_logit_reduced_05 <- ifelse(probs_logit_reduced_test >
    0.5,
25                                     levels(train_churn$churn)
                                     [2], levels(train_churn$
    churn)[1])

```



```

26 pred_class_logit_reduced_05 <- factor(pred_class_logit_reduced_
    05, levels=levels(train_churn$churn))
27 conf_matrix_logit_reduced_05 <- table(Actual=test_churn$churn,
    Predicted=pred_class_logit_reduced_05)
28 print("Confusion Matrix (Reduced Model, Threshold 0.5):")
29 print(conf_matrix_logit_reduced_05)
30 # print("Proportions table (Reduced Model):")
31 # print(prop.table(conf_matrix_logit_reduced_05, margin=1))
32 # Marking scheme: "removal of non-significant variables did not
    improve the test confusion matrix much."
33 # (Compare the prop.tables or specific metrics like sensitivity/
    specificity).

```

3.3.15 2e: Random forest for churn

Question Use a random forest to predict *churn* and evaluate the predictions. (If computations are too slow, reduce *ntree* or use smaller training data.)

Base R Solution (using randomForest)

```

1 library(randomForest)
2 # Assuming train_churn, test_churn from 1a
3 set.seed(86554354) % \textit{(Consistent seed)} %
4 rf_churn <- randomForest(churn ~ ., data=train_churn,
5                           ntree=50, # Reduced for speed as per
5                               hint/marking scheme
6                           importance=TRUE, na.action=na.roughfix)
7
8 # Predict classes directly
9 pred_class_rf_churn <- predict(rf_churn, newdata=test_churn, na.
    action=na.pass)
10 conf_matrix_rf_churn <- table(Actual=test_churn$churn, Predicted
    =pred_class_rf_churn)
11 print("Confusion Matrix (Random Forest):")
12 print(conf_matrix_rf_churn)
13 # accuracy_rf_churn <- sum(diag(conf_matrix_rf_churn)) / sum(
    conf_matrix_rf_churn)
14 # print(paste("Accuracy (Random Forest):", accuracy_rf_churn))
15 # print("Proportions table (Random Forest):")
16 # print(prop.table(conf_matrix_rf_churn, margin=1))
17 # Marking scheme results (row proportions):
18 # FALSE TRUE
19 # 0 0.94564187 0.05435813
20 # 1 0.06290371 0.93709629
21 # "The random forest improved the predictions considerably" (
    compared to logistic regression).
22 # Higher true positive for churn (0.937 vs 0.917) and higher
    true negative (0.945 vs 0.816).

```

3.3.16 2f: Typical features of customers who churn

Question Based on your analysis in 2a)-2e, what are the typical features of customers who churn?

Solution Outline (Textual) Synthesize findings from descriptive stats, logistic regression (significant coefficients, signs), and random forest (variable importance if examined, though not explicitly asked for plot here).

3.4 Exam Spring 2024

For variable definitions, see the table at the end of this exam section.

Task 1: Methodological Topics

3.4.1 1a: Explain R-function

Question Explain what the following R-function is doing.

Listing 55: R-function f for 1a (Exam 2024)

```
1 f <- function(x0, x, y, K=3) {  
2   d <- abs(x - x0)  
3   o <- order(d)[1:K]  
4   x1 <- x[o]  
5   y1 <- y[o]  
6   xydata <- data.frame(x1=x1, y1=y1)  
7   reg <- lm(y1 ~ x1, data=xydata)  
8   ypred <- predict(reg, newdata=data.frame(x1=x0, y1=1)) % \  
   textit{(y1=1 is placeholder)} %  
9   return(ypred)  
10 }
```

Explanation This R function $f(x_0, x, y, K=3)$ implements a local regression prediction method. 1. It takes a target predictor value x_0 , a vector of training predictor values x , corresponding training response values y , and the number of neighbors K (default 3). 2. It calculates the absolute distances (d) between x_0 and all values in x . 3. It finds the indices (o) of the K values in x that are closest to x_0 . 4. It selects these K closest predictor values (x_1) and their corresponding response values (y_1). 5. It creates a data frame $xydata$ from these K neighbors. 6. It fits a simple linear regression model (reg) of y_1 on x_1 using only these K neighbors. 7. Finally, it uses this locally fitted linear model to predict the response ($ypred$) for the original target point x_0 . The method resembles K -Nearest Neighbors (KNN) regression but instead of averaging the responses of the neighbors, it fits a linear model to them for prediction. The $y_1=1$ in $predict()$ is a placeholder and not used when predicting for a new x_1 .

3.4.2 1b: Optimal K for function f using LOOCV

Question Consider the following small dataset: $x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$, $y = [5.26, 9.13, 11.17, 15.64, 25.32, 25.55, 41.39, 48.17, 58.65, 68.24]$. Use leave-one-out cross-validation (LOOCV) to determine the optimal K in the function f for this dataset.

Base R Solution (basert på løsningsforslag)

```
1 x_1b <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)  
2 y_1b <- c(5.26, 9.13, 11.17, 15.64, 25.32, 25.55, 41.39, 48.17, 58.65, 68.24)  
3 n_1b <- length(x_1b)  
4  
5 # Function f from 1a (ensure it's defined)  
6 f <- function(x0, x, y, K=3) {  
7   d <- abs(x - x0); o <- order(d)[1:K]; x1 <- x[o]; y1 <- y[o]
```

```

8   xydata <- data.frame(x1=x1, y1=y1); reg <- lm(y1 ~ x1, data=
   xydata)
9   return(predict(reg, newdata=data.frame(x1=x0))) # Removed y1=1
   placeholder
10 }
11
12 # LOOCV function (as per solution structure)
13 loo_cv_func <- function(K_val, x_full, y_full) {
14   n_loo <- length(x_full)
15   squared_errors <- numeric(n_loo)
16   for (i in 1:n_loo) {
17     x_train <- x_full[-i]
18     y_train <- y_full[-i]
19     x_test_single <- x_full[i]
20     y_test_single <- y_full[i]
21
22     # Ensure K is not greater than number of available training
   points for local lm
23     K_to_use <- min(K_val, length(x_train))
24     if (K_to_use < 2) K_to_use <- 2 # lm needs at least 2 points
   for y~x
25
26     pred_loo <- f(x_test_single, x_train, y_train, K=K_to_use)
27     squared_errors[i] <- (y_test_single - pred_loo)^2
28   }
29   return(mean(squared_errors))
30 }
31
32 # Iterate for different K values (solution implies K from 2 to
   10, but K=n-1 for LOOCV training set)
33 # K can range from 2 (min for lm) up to n-1 (using all training
   points for local lm)
34 possible_K_vals <- 2:(n1b-1)
35 loocv_mse_values <- sapply(possible_K_vals, loo_cv_func, x_full=
   x1b, y_full=y1b)
36
37 # Plot results and find optimal K
38 plot(possible_K_vals, loocv_mse_values, type="b", xlab="K (
   Number of Neighbors)",
39       ylab="LOOCV MSE", main="LOOCV to find Optimal K")
40 optimal_K1b <- possible_K_vals[which.min(loocv_mse_values)]
41 points(optimal_K1b, min(loocv_mse_values), col="red", pch=19,
   cex=1.5)
42 cat("Optimal K found by LOOCV:", optimal_K1b, "\n")
43 # Exam solution PDF indicates K=5 is optimal.

```

3.4.3 1c: Plot predictions with optimal K

Question *Plot y against x and add a line with the predictions based on the optimal K .*

Base R Solution

```

1 # Assuming x1b, y1b from 1b, and optimal_K1b = 5
2 optimal_K_plot <- 5
3 predictions1c <- numeric(n1b)
4 # To generate a smooth line, we'd typically predict on a grid of
   x0 values.
5 # However, to match the solution's likely intent (predicting on
   training points for the line):

```

```

6 for (i in 1:n_1b) {
7   # Here, for plotting the fit on training data, f uses the full
      dataset x_1b, y_1b as its "training"
8   # to find neighbors for x_1b[i]. This is for visualizing the
      fitted function.
9   predictions_1c[i] <- f(x_1b[i], x_1b, y_1b, K=optimal_K_plot)
10 }
11
12 plot(x_1b, y_1b, xlab="x", ylab="y", main=paste("Data and Fitted
      Line with K =", optimal_K_plot))
13 order_x_1b <- order(x_1b)
14 lines(x_1b[order_x_1b], predictions_1c[order_x_1b], col="red",
      lwd=2)
15 legend("topleft", legend=paste("Fit with K =", optimal_K_plot),
      col="red", lty=1, lwd=2)

```

3.4.4 1d: Modify function f for multiple predictors

Question In the function f , there is only one predictor. One way to allow for more than one predictor is to compute d in the f -function in (a) differently. Explain how such a modification can be done; exemplify with a case with two predictors.

Explanation and Conceptual R Modification

Example with two predictors (X_1, X_2): Let input x_0 be a vector $c(x_{01}, x_{02})$, and x be a matrix where rows are observations and columns are X_1, X_2 . The modified d calculation within f would be: *To adapt function f for multiple predictors ($p > 1$), the distance calculation d needs to be changed from $abs(x - x_0)$ to a multivariate distance. The most common is Euclidean distance.*

Example with two predictors (X_1, X_2): Let input x_0 be a vector $c(x_{01}, x_{02})$, and x be a matrix where rows are observations and columns are X_1, X_2 . The modified d calculation within f would be:

Listing 56: Conceptual modification for distance in f (2 predictors)

```

1 # Inside function f_multi(x0_vec, x_matrix, y_vec, K)
2 # x0_vec is c(x0_pred1, x0_pred2)
3 # x_matrix has columns for pred1, pred2
4
5 # Calculate Euclidean distances
6 # diff_sq <- sweep(x_matrix, 2, x0_vec, "-")^2 # (x_i1-x01)^2, (
      x_i2-x02)^2 for each row
7 # d_multi <- sqrt(rowSums(diff_sq))
8
9 # The rest of the logic for finding K nearest neighbors (o, xl,
      yl) remains similar,
10 # but xl will be a matrix of K rows and p columns.
11 # The lm call becomes: reg <- lm(yl ~ ., data=as.data.frame(xl))
12 # And newdata for predict needs to be a data frame with column
      names matching xl.
13 # newdata_pred <- as.data.frame(matrix(x0_vec, nrow=1,
14 #                                     dimnames=list(NULL,
      colnames(x_matrix))))

```

Crucially, if predictors are on different scales, they should be standardized before calculating Euclidean distances to prevent variables with larger scales from dominating the distance metric. The linear model $lm(yl \sim .)$ would then be a

multiple linear regression on the K selected neighbors' p (standardized, if applicable) predictor values.

3.4.5 1e: Implement backfitting for this method

Question Another way to allow for several predictors is to use backfitting. Explain how you would implement backfitting for this method (you do not need to do it).

Explanation

Backfitting Algorithm with function f (example for $p = 2$ predictors X_1, X_2):

1. **Initialize:** $\hat{\beta}_0 = \text{mean}(Y)$. Initialize $\hat{f}_1(x_{i1}) = 0$ and $\hat{f}_2(x_{i2}) = 0$ for all observations i .
2. **Iterate** until convergence (i.e., \hat{f}_j functions stabilize):
 - (a) **Update** $\hat{f}_1(X_1)$: Compute partial residuals: $r_{i1} = y_i - \hat{\beta}_0 - \hat{f}_2(x_{i2})$. For each unique value x_{u1} of X_1 (or for each x_{i1}): $\hat{f}_1(x_{u1}) \leftarrow \text{Call } f(x_{u1}, X_1, r_1, K)$. (Typically, one would re-center: $\hat{f}_1 \leftarrow \hat{f}_1 - \text{mean}(\hat{f}_1)$).
 - (b) **Update** $\hat{f}_2(X_2)$: Compute partial residuals: $r_{i2} = y_i - \hat{\beta}_0 - \hat{f}_1(x_{i1})$ (using updated \hat{f}_1). For each unique value x_{u2} of X_2 : $\hat{f}_2(x_{u2}) \leftarrow \text{Call } f(x_{u2}, X_2, r_2, K)$. (Re-center: $\hat{f}_2 \leftarrow \hat{f}_2 - \text{mean}(\hat{f}_2)$).

The final prediction for a new observation $(\mathbf{x}_{01}, \mathbf{x}_{02})$ is $\hat{Y}_0 = \hat{\beta}_0 + \hat{f}_1(\mathbf{x}_{01}) + \hat{f}_2(\mathbf{x}_{02})$. The function f from 1a acts as the univariate smoother for each predictor on its partial residuals. The optimal K for each f_j can be chosen using CV, possibly within each backfitting iteration or as a global parameter. Backfitting allows fitting an additive model $Y = \beta_0 + f_1(X_1) + f_2(X_2) + \dots + f_p(X_p) + \epsilon$, where each f_j is estimated using a univariate smoother, such as our function f (from 1a, adapted for single predictor use).

Backfitting Algorithm with function f (example for $p = 2$ predictors X_1, X_2):

1. **Initialize:** $\hat{\beta}_0 = \text{mean}(Y)$. Initialize $\hat{f}_1(x_{i1}) = 0$ and $\hat{f}_2(x_{i2}) = 0$ for all observations i .
2. **Iterate** until convergence (i.e., \hat{f}_j functions stabilize):
 - (a) **Update** $\hat{f}_1(X_1)$: Compute partial residuals: $r_{i1} = y_i - \hat{\beta}_0 - \hat{f}_2(x_{i2})$. For each unique value x_{u1} of X_1 (or for each x_{i1}): $\hat{f}_1(x_{u1}) \leftarrow \text{Call } f(x_{u1}, X_1, r_1, K)$. (Typically, one would re-center: $\hat{f}_1 \leftarrow \hat{f}_1 - \text{mean}(\hat{f}_1)$).
 - (b) **Update** $\hat{f}_2(X_2)$: Compute partial residuals: $r_{i2} = y_i - \hat{\beta}_0 - \hat{f}_1(x_{i1})$ (using updated \hat{f}_1). For each unique value x_{u2} of X_2 : $\hat{f}_2(x_{u2}) \leftarrow \text{Call } f(x_{u2}, X_2, r_2, K)$. (Re-center: $\hat{f}_2 \leftarrow \hat{f}_2 - \text{mean}(\hat{f}_2)$).

The final prediction for a new observation $(\mathbf{x}_{01}, \mathbf{x}_{02})$ is $\hat{Y}_0 = \hat{\beta}_0 + \hat{f}_1(\mathbf{x}_{01}) + \hat{f}_2(\mathbf{x}_{02})$. The function f from 1a acts as the univariate smoother for each predictor on its partial residuals. The optimal K for each f_j can be chosen using CV, possibly within each backfitting iteration or as a global parameter.

Task 2: Analysis of airline.csv Data - Customer Satisfaction Target variable: *satisfaction*. Often converted to binary for classification.

3.4.6 2a: Recode categorical variables and split data

Question *If necessary, recode categorical variables as factors. Motivate your choices in words. (Implicit: Split data for training/testing as per general instructions and solution PDF structure for this exam).*

Base R Solution (based on solution PDF)

```
1 airline <- read.csv("airline.csv", stringsAsFactors = FALSE) % \
  textit{Read data} %
2 # Convert 'satisfaction' to factor (assuming 'satisfied' vs '
  dissatisfied' for classification)
3 # This step depends on how 'satisfaction' is defined in the
  original CSV.
4 # If it's text "satisfied"/"dissatisfied", it's already good for
  factor.
5 # If numeric, might need binning. Let's assume it becomes a
  factor.
6 # airline$satisfaction <- as.factor(airline$satisfaction)
7
8 # Recode variables with <= 7 unique values as factors (as per
  solution for this exam)
9 for (i in 1:ncol(airline)) {
10   if (length(unique(airline[,i])) <= 7 && !is.factor(airline[,i
      ])) {
11     airline[,i] <- as.factor(airline[,i])
12   }
13 }
14 # Example: Gender, Customer_Type, Type_of_Travel, Class are
  likely candidates.
15 # Satisfaction scores (0-5) might also be converted if not
  already factors.
16 # Motivation: Ensure categorical predictors are treated as such
  by modeling functions.
17
18 # Split data (as per solution PDF structure for this exam)
19 set.seed(123) # From solution PDF for this task
20 n_airline <- nrow(airline)
21 train_idx_airline <- sample(1:n_airline, floor(n_airline/2))
22 training_airline <- airline[train_idx_airline, ]
23 test_airline <- airline[-train_idx_airline, ]
24 # sapply(training_airline, class)
```

Tidyverse Solution

```
1 library(dplyr)
2 airline_tidy <- read.csv("airline.csv", stringsAsFactors = FALSE
  ) %>%
3   mutate(across(where(~length(unique(.)) <= 7 & !is.factor(.)),
     as.factor))
4   # May need specific mutate for 'satisfaction' if it's not
     binary textual
5   # e.g. mutate(satisfaction = factor(ifelse(satisfaction_score
     >= 4, "satisfied", "dissatisfied")))
6
7 set.seed(123)
8 train_idx_airline_tidy <- sample(1:nrow(airline_tidy), floor(
  nrow(airline_tidy)/2))
9 training_airline_tidy <- airline_tidy[train_idx_airline_tidy, ]
10 test_airline_tidy <- airline_tidy[-train_idx_airline_tidy, ]
11 # glimpse(training_airline_tidy)
```

3.4.7 2b: Motivation for data usage (Inference: "Why dissatisfied?")

Question In tasks (a)-(g) the methods and models are used to answer the question "Why are some customers dissatisfied?". With this in mind, motivate your choice to evaluate the models on all/training/test data.

Solution/Motivation (based on exam solution PDF)

For tasks (a)-(g), which focus on understanding why customers are dissatisfied (an inferential goal rather than pure out-of-sample prediction), one could argue for using the entire dataset to maximize statistical power for inference. However, the provided solution chooses to **split the data into training and test sets from the start, and perform all descriptive analysis and initial model fitting (for inference) on the training data only**. The motivations for this approach are:

1. **Comparability:** To allow direct comparison of model performance (e.g., R-squared, variable importance) with models developed later for the explicit prediction task (h-l), which must be evaluated on test data.
2. **Preventing Data Leakage:** Using only training data for initial exploration and model specification for the inferential part acts as a safeguard against inadvertently using information from the test set to guide these early modeling choices. This ensures that when the test set is used for the later prediction task, it is truly "unseen".

3.4.8 2c: Descriptive statistics for

Question Use descriptive statistics to find variables associated with *satisfaction*. First, explain which type of descriptive statistics (types of tables and figures) that you are using. Give examples of R code but do not show all code and output if you are doing the same thing many times. Summarize your results as text, mentioning the relevant numbers.

Base R Solution (Conceptual based on exam solution)

```
1 # Assuming training_airline from 2a. Assume 'satisfaction' is a
   # binary factor ("satisfied", "dissatisfied").
2 # Overall satisfaction rate:
3 # print(prop.table(table(training_airline$satisfaction))) # e.g
   # ., 55% satisfied
4
5 # Function for conditional proportions (as per exam solution)
6 proptab_satisfaction <- function(data, predictor_col_name) {
7   tbl <- table(data[[predictor_col_name]], data$satisfaction)
8   return(round(100 * prop.table(tbl, margin = 1), 1)) #
   # Proportion satisfied/dissatisfied per predictor level
9 }
10 # Example: proptab_satisfaction(training_airline, "Gender")
11 #           satisfaction
12 # Gender      dissatisfied satisfied
13 #   Female           35         65
14 #   Male            56         44
15 # (Values from exam solution PDF for illustration)
16
17 # For numeric variables: Boxplots and grouped summaries
18 # boxplot(Age ~ satisfaction, data=training_airline, ylab="Age")
19 # by(training_airline$Flight_Distance, training_airline$
   # satisfaction, summary)
```



```

20
21 # Summary of results (textual, based on exam solution PDF):
22 # - Overall 55% satisfied.
23 # - Gender: Females (65% satisfied) more satisfied than Males
    (44%).
24 # - Customer_Type: Loyal customers more satisfied.
25 # - Type_of_Travel, Class: Business travelers/class more
    satisfied.
26 # - Specific satisfaction Qs (Inflight_entertainment,
    Cleanliness, etc.): Positively associated with overall
    satisfaction.
27 # - Age: Satisfied customers tend to be older on average.
28 # - Flight_Distance, Delays: No strong or obvious association
    with overall satisfaction from boxplots/summaries.

```

Tidyverse Solution (Conceptual)

```

1 # library(dplyr); library(ggplot2)
2 # Assuming training_airline_tidy from 2a.
3 # training_airline_tidy %>% count(satisfaction) %>% mutate(prop
    = n/sum(n))
4
5 # Conditional proportions
6 # training_airline_tidy %>%
7 #   count(Gender, satisfaction) %>%
8 #   group_by(Gender) %>%
9 #   mutate(prop = round(100 * n / sum(n), 1)) %>% print()
10
11 # Boxplots
12 # ggplot(training_airline_tidy, aes(x=satisfaction, y=Age, fill=
    satisfaction)) + geom_boxplot()
13 # Grouped summaries
14 # training_airline_tidy %>% group_by(satisfaction) %>%
15 #   summarise(mean_age = mean(Age, na.rm=TRUE), median_dist =
    median(Flight_Distance, na.rm=TRUE))

```

3.5 2d: Logistic Regression (Why dissatisfied?)

Listing 57: 2d: Logistic Regression - Inferential (airline)

```

1 logreg_infer <- glm(satisfaction ~ Gender + Customer_Type + Age
    + Type_of_Travel +
2     Class + Flight_Distance + Food_and_drink + Inflight_
    entertainment +
3     Online_support + Baggage_handling + Checkin_service +
    Arrival_Delay_in_Minutes +
4     Inflight_wifi_service + Leg_room_service + On_board_service
    + Gate_location + Seat_comfort,
5     data=training_airline, family=binomial)
6
7 probs_logreg_infer_test <- predict(logreg_infer, newdata=test_
    airline, type="response")
8 preds <- ifelse(probs_logreg_infer_test > 0.5, "satisfied", "
    dissatisfied")
9 conf_matrix <- table(Actual = test_airline$satisfaction,
    Predicted = preds)
10 print(conf_matrix)

```


3.6 2e: Classification Tree (Why dissatisfied?)

Listing 58: 2e: Classification Tree - Inferential

```
1 library(tree)
2 tree_model <- tree(satisfaction ~ Gender + Customer_Type + Age +
3   Type_of_Travel +
4   Class + Flight_Distance + Food_and_drink + Inflight_
5   entertainment +
6   Online_support + Baggage_handling + Checkin_service +
7   Arrival_Delay_in_Minutes +
8   Inflight_wifi_service + Leg_room_service + On_board_service
9   + Gate_location + Seat_comfort,
10  data=training_airline)
11 plot(tree_model)
12 text(tree_model, pretty=0)
```

3.7 2f: Random Forest (Why dissatisfied?)

Listing 59: 2f: Random Forest - Inferential

```
1 library(randomForest)
2 rf_model <- randomForest(satisfaction ~ Gender + Customer_Type +
3   Age + Type_of_Travel +
4   Class + Flight_Distance + Food_and_drink + Inflight_
5   entertainment +
6   Online_support + Baggage_handling + Checkin_service +
7   Arrival_Delay_in_Minutes +
8   Inflight_wifi_service + Leg_room_service + On_board_service
9   + Gate_location + Seat_comfort,
10  data=training_airline, importance=TRUE)
11 varImpPlot(rf_model)
```

3.8 2g: Why are some customers dissatisfied?

Customers who are dissatisfied often report low satisfaction with inflight entertainment, seat comfort, online support, and baggage handling. These variables were consistently important in the models. Loyal customers and business class passengers are generally more satisfied. Demographic variables like age and gender also play a role.

3.9 2h: Assumptions for Prediction Task (Pre-flight variables)

Variables known before flight: Gender, Age, Customer Type, Type of Travel, Class, and Flight Distance. Satisfaction ratings and delay variables are post-flight and not available for prediction before departure.

3.10 2i: Motivation for Train/Test Split in Prediction

Since the goal is to predict dissatisfaction for new customers, it is crucial to evaluate the model on unseen test data to assess generalization performance.

3.11 2j: Logistic Regression (Prediction)

Listing 60: 2j: Logistic Regression - Prediction

```
1 logreg_pred <- glm(satisfaction ~ Gender + Customer_Type + Age +
2   Type_of_Travel + Class + Flight_Distance,
3   data=training_airline, family=binomial)
4 probs_pred <- predict(logreg_pred, newdata=test_airline, type="
5   response")
6 preds <- ifelse(probs_pred > 0.5, "satisfied", "dissatisfied")
7 conf_matrix <- table(Actual = test_airline$satisfaction,
8   Predicted = preds)
9 print(conf_matrix)
```

3.12 2k: Random Forest (Prediction)

Listing 61: 2k: Random Forest - Prediction

```
1 rf_pred <- randomForest(satisfaction ~ Gender + Customer_Type +
2   Age + Type_of_Travel + Class + Flight_Distance,
3   data=training_airline, importance=TRUE)
4 preds_rf <- predict(rf_pred, newdata=test_airline)
5 conf_matrix <- table(Actual = test_airline$satisfaction,
6   Predicted = preds_rf)
7 print(conf_matrix)
```

3.13 2l: Future Data Collection

To improve predictions, collect data on pre-flight interactions: booking channel, time between booking and flight, seat selection, and support tickets. These can act as proxies for future dissatisfaction and help model potential outcomes more accurately.

4 Tutorials

This section summarizes key tasks and R code from the provided exercise sets, assuming they correspond to the course tutorials.

4.1 Tutorial 1

4.1.1 Task 1: Linear Regression and KNN (One Predictor)

Context Given a small dataset (x, y pairs).

1a: Prediction with Linear Regression (Manual/Calculator) *Question: What is the prediction for $x = 3$ if the linear regression model is used? Solution Approach: Manually calculate $\hat{\alpha}_0, \hat{\alpha}_1$ using formulas for simple linear regression, then predict $\hat{y} = \hat{\alpha}_0 + \hat{\alpha}_1 \cdot 3$.*

Listing 62: Manual OLS Calculation (Conceptual - Ex Set 1, T1a)

```
1 # x_t1 <- 1:5
2 # y_t1 <- c(1.88, 4.54, 10.12, 9.14, 11.26)
3 # n_t1 <- length(x_t1)
```

```

4 # mean_x_t1 <- mean(x_t1)
5 # mean_y_t1 <- mean(y_t1)
6 #
7 # beta1_hat_t1 <- sum((x_t1 - mean_x_t1) * (y_t1 - mean_y_t1)) /
  sum((x_t1 - mean_x_t1)^2)
8 # beta0_hat_t1 <- mean_y_t1 - beta1_hat_t1 * mean_x_t1
9 #
10 # prediction_x3_t1 <- beta0_hat_t1 + beta1_hat_t1 * 3
11 # print(prediction_x3_t1) % \textit{Solution gives 7.388} %

```

1b: Fit Linear Regression with `lm()` and Predict *Question: Use `lm()` to fit the same model and `predict()` for $x=3$.*

Listing 63: Using `lm` and `predict` (Ex Set 1, T1b)

```

1 # x_t1 <- 1:5
2 # y_t1 <- c(1.88, 4.54, 10.12, 9.14, 11.26)
3 # model_t1b <- lm(y_t1 ~ x_t1)
4 # predict_x3_t1b <- predict(model_t1b, newdata = data.frame(x_t1
  = 3))
5 # print(predict_x3_t1b) % \textit{Solution gives 7.388} %

```

1c: KNN Prediction *Question: Prediction for $x=3$ if KNN with $K=1$, $K=3$, $K=5$ is used. Solution Approach (Manual/Calculator): For $K=1$: Find y -value corresponding to x closest to 3. For $K=3$: Find 3 y -values for x 's closest to 3, then average them. For $K=5$: Average all 5 y -values.*

Listing 64: KNN Predictions (Conceptual - Ex Set 1, T1c)

```

1 # y_t1 <- c(1.88, 4.54, 10.12, 9.14, 11.26)
2 # K=1 (x=3 is closest): 10.12
3 # K=3 (x=2,3,4 are closest): mean(c(4.54, 10.12, 9.14)) % \
  textit{Result: 7.933333} %
4 # K=5 (all points): mean(y_t1) % \textit{Result: 7.388} %

```

1d: Understanding KNN R-function *Question: Explain each row of the provided KNN R-function.*

Listing 65: KNN R-function (Ex Set 1, T1d)

```

1 knn_func_t1d <- function(x0, x, y, K=20) { % \textit{Note:
  Exercise sheet has K=20, example uses K=3} %
2   d <- abs(x-x0) % \textit{# Calculate absolute
  distances from x0 to all x} %
3   o <- order(d)[1:K] % \textit{# Get indices of the
  K smallest distances} %
4   ypred <- mean(y[o]) % \textit{# Average y-values of
  the K nearest neighbors} %
5   return(ypred) % \textit{# Return the
  prediction} %
6 }
7 # Experiment: K_exp=3; x0_exp=3; x_exp=1:5; y_exp=c
  (1.88,4.54,10.12,9.14,11.26)
8 # d_exp <- abs(x_exp - x0_exp) % Result: 2 1 0 1 2 %
9 # o_exp <- order(d_exp)[1:K_exp] % Result: 3 2 4 (indices for x
  =3, x=2, x=4) %
10 # ypred_exp <- mean(y_exp[o_exp]) % Result: mean(y[c(3,2,4)]) =
  7.933333 %

```

1e: Choosing Method Based on Plot *Question: Which method (LR or KNN $K=1,2,3$) for a given scatter plot? Why? Solution Approach: If plot shows strong linear trend, Linear Regression is preferred for efficiency and interpretability. If non-linear, KNN might be better, with K chosen to balance bias/variance.*

4.1.2 Task 2: Data Splitting and Basic Linear Model

2a: Load, Summarize, Help for College

```
1 library(ISLR)
2 # summary(College)
3 # help(College)
```

2b: 50/50 Training/Test Split

```
1 set.seed(123)
2 n_college <- nrow(College)
3 train_indicator_college <- sample(1:n_college, size=floor(n_
  college/2))
4 train_college <- College[train_indicator_college,]
5 test_college <- College[-train_indicator_college,]
```

2c: Fit Linear Model for Apps Private + Accept

```
1 lm_apps_t2c <- lm(Apps ~ Private + Accept, data=train_college)
2 # summary(lm_apps_t2c)
```

2d: Compute Training MSE

```
1 pred_train_t2d <- predict(lm_apps_t2c, newdata=train_college)
2 mse_train_t2d <- mean((train_college$Apps - pred_train_t2d)^2)
3 # print(mse_train_t2d) % \textit{Solution shows 1437492} %
```

2e: Model with Accept only, compute Training MSE

```
1 lm_apps_t2e <- lm(Apps ~ Accept, data=train_college)
2 pred_train_t2e <- predict(lm_apps_t2e, newdata=train_college)
3 mse_train_t2e <- mean((train_college$Apps - pred_train_t2e)^2)
4 # print(mse_train_t2e) % \textit{Solution shows 1440012, larger
  than 2d, as expected} %
```

2f: Compute Test MSE for models from 2c and 2e

```
1 pred_test_t2c <- predict(lm_apps_t2c, newdata=test_college)
2 mse_test_t2c <- mean((test_college$Apps - pred_test_t2c)^2)
3 # print(paste("Test MSE (Private + Accept):", mse_test_t2c)) % \
  textit{Sol: 1849668} %
4
5 pred_test_t2e <- predict(lm_apps_t2e, newdata=test_college)
6 mse_test_t2e <- mean((test_college$Apps - pred_test_t2e)^2)
7 # print(paste("Test MSE (Accept only):", mse_test_t2e)) % \
  textit{Sol: 1854227} %
8 # Model with Private + Accept has slightly better test MSE here.
```

2h: KNN for Apps Accept, compare Test MSE

```
1 # Using knn_func_t1d from earlier, assuming K=3 as per solution's use of it
2 # Need to apply it for each test observation
3 # x0_knn_t2h <- test_college$Accept
4 # x_knn_t2h <- train_college$Accept
5 # y_knn_t2h <- train_college$Apps
6 #
7 # pred_knn_t2h <- sapply(x0_knn_t2h, knn_func_t1d, x=x_knn_t2h,
8   y=y_knn_t2h, K=3)
9 # mse_knn_t2h <- mean((test_college$Apps - pred_knn_t2h)^2)
10 # print(paste("Test MSE (KNN K=3 for Apps ~ Accept):", mse_knn_t2h)) % \textit{Sol: 4473360} %
# KNN performs much worse than linear regression here.
```

4.1.3 Task 3: Exercise 3.10a-f from ISLR Book

This task involves fitting and interpreting a multiple linear regression model for *Sales* ~ *Price* + *Urban* + *US* on the *ISLR::Carseats* dataset. Steps include fitting, interpreting coefficients, identifying significant predictors, and comparing models.

```
1 # library(ISLR)
2 # data(Carseats)
3 # (a) Fit model
4 # model_carseats_310a <- lm(Sales ~ Price + Urban + US, data=
5   Carseats)
6 # (b) Interpret coefficients
7 # summary(model_carseats_310a)
8 # (c) Write equation
9 # (d) Identify significant predictors (Price, US based on p-
10   values)
11 # (e) Fit model with significant predictors only
12 # model_carseats_310e <- lm(Sales ~ Price + US, data=Carseats)
13 # (f) Compare R-squared and Adjusted R-squared (very similar for
14   these two models).
15 # (g) Optional: Split, train, test.
16 # set.seed(12345)
17 # n_cs <- nrow(Carseats)
18 # train_idx_cs <- sample(1:n_cs, n_cs/2)
19 # train_cs_310 <- Carseats[train_idx_cs,]
20 # test_cs_310 <- Carseats[-train_idx_cs,]
21 # lm3_cs <- lm(Sales ~ Price + Urban + US, data=train_cs_310)
22 # pred3_cs <- predict(lm3_cs, newdata=test_cs_310)
23 # mse3_cs <- mean((test_cs_310$Sales - pred3_cs)^2) % Sol:
24   6.662185 %
# lm4_cs <- lm(Sales ~ Price + US, data=train_cs_310)
# pred4_cs <- predict(lm4_cs, newdata=test_cs_310)
# mse4_cs <- mean((test_cs_310$Sales - pred4_cs)^2) % Sol:
6.580072 %
# Reduced model slightly better on this test split.
```

4.2 Tutorial 2

4.2.1 Task 1: Logistic Regression and KNN for Binary Classification

Context *Small dataset ($x=1:7$, y binary).*

1a: Fit Logistic Regression, predict for x0=4

```
1 # x_t2 <- 1:7; y_t2 <- c(0,1,0,1,1,1,1)
2 # data_t2 <- data.frame(x=x_t2, y=as.factor(y_t2)) % \textit{
  Ensure y is factor for glm binomial} %
3 # m1_t2 <- glm(y ~ x, data=data_t2, family="binomial")
4 # summary(m1_t2)
5 # Estimated equation:  $P(Y=1|X=x) = 1 / (1 + \exp(-(coef(m1\_t2)[1]
  + coef(m1\_t2)[2]x)))$ 
6 # For x0=4: predict(m1_t2, newdata=data.frame(x=4), type="
  response") % Sol: 0.8628, predict Y=1 %
```

1b: Plot probabilities from Logistic Regression

```
1 # plot(x_t2, as.numeric(y_t2)-1, ylab="P(Y=1) or Y") # Plot 0/1
  Y values
2 # probs_m1_t2 <- predict(m1_t2, type="response")
3 # lines(x_t2, probs_m1_t2, col="blue")
```

1c: KNN probability estimation for x0=4 with K=1,3,5

```
1 # knn_prob_t2 <- function(x0, x_train, y_train, K_val) {
2 #   d <- abs(x_train - x0)
3 #   o <- order(d)[1:K_val]
4 #   return(mean(as.numeric(y_train[o])-1)) # Mean of 0/1 y-
  values
5 # }
6 # knn_prob_t2(4, x_t2, data_t2$y, K=1) % Sol: 1 (y at x=4 is 1)
  %
7 # knn_prob_t2(4, x_t2, data_t2$y, K=3) % Sol: mean(y at x=3,4,5)
  = mean(c(0,1,1))=0.667 %
8 # knn_prob_t2(4, x_t2, data_t2$y, K=5) % Sol: mean(y at x
  =2,3,4,5,6)=mean(c(1,0,1,1,1,1))=0.8 %
```

1d: Plot KNN probabilities for K=1,3,5 and logistic

```
1 # plot(x_t2, as.numeric(data_t2$y)-1, ylab="P(Y=1) or Y", xlab="
  x")
2 # for (K_val_plot in c(1,3,5)) {
3 #   probs_knn_plot <- sapply(x_t2, knn_prob_t2, x_train=x_t2, y_
  train=data_t2$y, K_val=K_val_plot)
4 #   lines(x_t2, probs_knn_plot, lty=K_val_plot+1, col=K_val_plot
  +1)
5 # }
6 # lines(x_t2, probs_m1_t2, col="blue", lty=1) # Logistic
7 # legend("bottomright", legend=c("Logistic", "KNN K=1", "KNN K
  =3", "KNN K=5"),
8 #   col=c("blue", 2, 3, 4), lty=c(1,2,3,4))
```

1e: Confusion matrices for logistic and KNN (K=3)

```
1 # Logistic predictions (threshold 0.5)
2 # pred_log_t2e <- ifelse(probs_m1_t2 > 0.5, 1, 0)
3 # table(Actual=as.numeric(data_t2$y)-1, Predicted=pred_log_t2e)
4 # prop.table(table(Actual=as.numeric(data_t2$y)-1, Predicted=
  pred_log_t2e), margin=1)
5
6 # KNN K=3 predictions
7 # probs_knn3_t2e <- sapply(x_t2, knn_prob_t2, x_train=x_t2, y_
  train=data_t2$y, K_val=3)
8 # pred_knn3_t2e <- ifelse(probs_knn3_t2e > 0.5, 1, 0)
9 # table(Actual=as.numeric(data_t2$y)-1, Predicted=pred_knn3_t2e)
```

```

10 # prop.table(table(Actual=as.numeric(data_t2$y)-1, Predicted=
    pred_knn3_t2e), margin=1)
11 # Solution indicates identical predictions for this small
    dataset.

```

4.2.2 Task 3: Logistic Regression vs LDA (Default data)

3a: Fit Logistic and LDA, 50/50 split

```

1 # library(ISLR); library(MASS)
2 # data(Default)
3 # Default$default_numeric <- ifelse(Default$default == "Yes", 1,
    0) % \textit{Make numeric for comparison} %
4
5 # set.seed(1) % (Or a seed specified in tutorial if different) %
6 # n_def <- nrow(Default)
7 # train_idx_def <- sample(1:n_def, floor(n_def/2))
8 # train_def <- Default[train_idx_def, ]
9 # test_def <- Default[-train_idx_def, ]
10
11 # logreg_def <- glm(default ~ balance + income + student, data=
    train_def, family="binomial")
12 # lda_def <- lda(default ~ balance + income + student, data=
    train_def)
13 #
14 # prob_logreg_def_test <- predict(logreg_def, newdata=test_def,
    type="response")
15 # prob_lda_def_test <- predict(lda_def, newdata=test_def)$
    posterior[, "Yes"] % \textit{Or correct class index} %
16 # head(data.frame(LogRegProb=prob_logreg_def_test, LDAProb=prob_
    lda_def_test))

```

3b: Predict test data, confusion matrices. Remove variables?

```

1 # threshold <- 0.5
2 # pred_log_class <- ifelse(prob_logreg_def_test > threshold, "
    Yes", "No")
3 # conf_log <- table(Actual=test_def$default, Predicted=pred_log_
    class)
4 # print("Logistic Confusion Matrix:"); print(conf_log); print(
    prop.table(conf_log,1))
5 #
6 # pred_lda_class <- ifelse(prob_lda_def_test > threshold, "Yes",
    "No")
7 # conf_lda <- table(Actual=test_def$default, Predicted=pred_lda_
    class)
8 # print("LDA Confusion Matrix:"); print(conf_lda); print(prop.
    table(conf_lda,1))
9 # Solution note: Both good at Y=0, not Y=1. Lowering threshold
    might help.
10 # Removing variables: Try fitting models without 'student' or '
    income' if they were
11 # not significant or if LDA assumptions are violated by them.
    Compare results.

```

3c: Suitability of predictors for LDA. Other methods? *Question: Are student, balance, income all suitable for LDA? Other methods? Solution Notes: - LDA assumes normality of predictors within classes and equal covariance. - 'balance' and 'income' (continuous) might be reasonably normal. 'student' (binary*

factor) violates normality. - If normality/equal covariance is strongly violated, Logistic Regression is more robust. - QDA could be better if covariance matrices differ between default/non-default groups. - Non-parametric methods like KNN or Trees don't make these distributional assumptions.

4.2.3 Task 4 : Cross-Validation Methods (Auto data)

This task explains data prep for **Auto** (create binary 'y' from 'mpg', 'age' from 'year', remove 'mpg,name,year') and then applies validation set, LOOCV, and k-fold CV to a logistic regression model for 'y'.

Listing 66: CV Methods Example (Ex Set 2, T4)

```

1 # library(ISLR)
2 # Auto_t4 <- Auto
3 # Auto_t4$y <- as.factor(ifelse(Auto_t4$mpg > median(Auto_t4$mpg
4   ), "high", "low"))
5 # Auto_t4$age <- 83 - Auto_t4$year
6 # Auto_t4 <- Auto_t4[, !(names(Auto_t4) %in% c("mpg", "name", "
7   year"))]
8
9 # (b) Validation set (50/50 split)
10 # set.seed(1) % (Assume a seed) %
11 # n_auto_t4 <- nrow(Auto_t4)
12 # train_idx_t4 <- sample(1:n_auto_t4, floor(n_auto_t4/2))
13 # train_auto_t4 <- Auto_t4[train_idx_t4, ]
14 # test_auto_t4 <- Auto_t4[-train_idx_t4, ]
15 # m1_t4 <- glm(y ~ ., data=train_auto_t4, family="binomial")
16 # prob_t4_val <- predict(m1_t4, newdata=test_auto_t4, type="
17   response")
18 # pred_t4_val <- ifelse(prob_t4_val > 0.5, "high", "low")
19 # # table(test_auto_t4$y, pred_t4_val); mean(pred_t4_val != test
20   _auto_t4$y)
21
22 # (d) LOOCV (Manual loop shown in exercise, or use boot::cv.glm)
23 # library(boot)
24 # m_all_t4 <- glm(y ~ ., data=Auto_t4, family="binomial")
25 # cv_error_loo_t4 <- cv.glm(Auto_t4, m_all_t4, K=nrow(Auto_t4))$
26   delta[1] % (Requires cost function for error rate) %
27 # Manual loop from exercise set:
28 # error_rate_LOO_t4 <- mean(pred_LOO_t4_class != Auto_t4$y)
29 # % (Where pred_LOO_t4_class is from the loop. Solution gets
30   0.089 (high=0.923, low=0.117 error prop)) %
31
32 # (e) 8-fold CV (Manual loop shown in exercise, or use boot::cv.
33   glm with K=8)
34 # error_rate_k8_t4 <- mean(pred_k8_t4_class != Auto_t4$y)
35 # % (Solution gets 0.094 (high=0.903, low=0.117 error prop)) %
36
37 # (f) Preference for predictors: All vs. some (e.g., age +
38   weight)
39 # Fit LOOCV with reduced model: y ~ age + weight
40 # Compare LOOCV error rates. Solution: Reduced model was worse.

```

4.3 Tutorial 3

4.3.1 Task 1: Shrinkage Methods (College data)

Predict *Apps* using *Accept*, *Top10perc*, *Expend*. Compare OLS, Ridge, Lasso.

1a: Compare OLS, Ridge, Lasso for different λ s

```
1 # library(ISLR2); library(glmnet)
2 # xdata_t3 <- subset(College, select=c("Apps", "Accept", "
  Top10perc", "Expend"))
3 # x_mat_t3 <- as.matrix(xdata_t3[, -1])
4 # y_vec_t3 <- xdata_t3[, 1]
5
6 # OLS
7 # ols_t3 <- lm(Apps ~ Accept + Top10perc + Expend, data=xdata_t3
  )
8 # coef_ols_t3 <- coef(ols_t3)
9
10 # Ridge (example lambda=100)
11 # ridge_t3_100 <- glmnet(x_mat_t3, y_vec_t3, alpha=0, lambda
  =100)
12 # coef_ridge_t3_100 <- coef(ridge_t3_100)
13 # % (Repeat for other lambdas) %
14
15 # Lasso (example lambda=100)
16 # lasso_t3_100 <- glmnet(x_mat_t3, y_vec_t3, alpha=1, lambda
  =100)
17 # coef_lasso_t3_100 <- coef(lasso_t3_100)
18 # % (Repeat for other lambdas. Compare how coefficients shrink/
  go to zero) %
```

1b: Optimal λ using `cv.glmnet()` Explain what `cv.glmnet()` does: Performs k -fold CV (default 10) over a grid of λ values to find λ that minimizes CV error (e.g., MSE).

Listing 67: Optimal Lambda (Ex Set 3, T1b)

```
1 # cv_ridge_t3 <- cv.glmnet(x_mat_t3, y_vec_t3, alpha=0)
2 # lambda_ridge_opt_t3 <- cv_ridge_t3$lambda.min % Sol: 364.8993
  %
3 # cv_lasso_t3 <- cv.glmnet(x_mat_t3, y_vec_t3, alpha=1)
4 # lambda_lasso_opt_t3 <- cv_lasso_t3$lambda.min
```

1c: Re-estimate with optimal λ , compare coefficients

```
1 # ridge_opt_t3 <- glmnet(x_mat_t3, y_vec_t3, alpha=0, lambda=
  lambda_ridge_opt_t3)
2 # lasso_opt_t3 <- glmnet(x_mat_t3, y_vec_t3, alpha=1, lambda=
  lambda_lasso_opt_t3)
3 # print(cbind(OLS=coef_ols_t3, Ridge=coef(ridge_opt_t3), Lasso=
  coef(lasso_opt_t3)))
```

1d: Evaluate OLS, Ridge, Lasso by testMSE (validation set)

```
1 # n_t3 <- nrow(xdata_t3)
2 # set.seed(1) % (Assume a seed) %
3 # ind_t3 <- sample(1:n_t3, size=floor(n_t3/2))
4 # train_t3 <- xdata_t3[ind_t3,]; test_t3 <- xdata_t3[-ind_t3,]
5 # Xtrain_t3 <- as.matrix(train_t3[, -1]); ytrain_t3 <- train_t3
  [, 1]
6 # Xtest_t3 <- as.matrix(test_t3[, -1]); ytest_t3 <- test_t3
  [, 1]
7
8 # OLS_train_t3 <- lm(Apps ~ ., data=train_t3)
9 # pred_ols_t3 <- predict(OLS_train_t3, newdata=test_t3)
```

```

10 # mse_ols_t3 <- mean((ytest_t3 - pred_ols_t3)^2) % Sol: 1293261
    %
11
12 # Ridge (using lambda_ridge_opt_t3 from full data CV for
    simplicity, ideally re-CV on train_t3)
13 # ridge_train_t3 <- glmnet(Xtrain_t3, ytrain_t3, alpha=0, lambda
    =lambda_ridge_opt_t3)
14 # pred_ridge_t3 <- predict(ridge_train_t3, newx=Xtest_t3)
15 # mse_ridge_t3 <- mean((ytest_t3 - pred_ridge_t3)^2) % Sol:
    1399610 %
16 # % (LASSO similarly. Solution: OLS better than Ridge here.) %

```

4.3.2 Task 2: Non-linear Models (Small Dataset)

Polynomial regression vs. regression splines.

2a: Plot y against x

2b: Polynomial regression K=7, plot predictions

```

1 # x_t3_task2 <- 1:8
2 # y_t3_task2 <- c(8.88, 4.54, 5.12, 1.14, 2.26, 8.43, 10.92,
    14.47)
3 # data_t3_task2 <- data.frame(x=x_t3_task2, y=y_t3_task2)
4 # poly7_t3 <- lm(y ~ poly(x, 7, raw=TRUE), data=data_t3_task2) %
    (raw=TRUE for x, x^2...) %
5 # plot(data_t3_task2$x, data_t3_task2$y)
6 # lines(data_t3_task2$x, predict(poly7_t3), col="red") % (
    Perfect fit as N=P) %

```

2c: Regression spline (manual basis) with knots at 3, 6. Plot.

```

1 # h_func <- function(x, xi) pmax((x-xi)^3, 0)
2 # spline_manual_t3 <- lm(y ~ x + I(x^2) + I(x^3) + h_func(x,3) +
    h_func(x,6), data=data_t3_task2)
3 # lines(data_t3_task2$x, predict(spline_manual_t3), col="blue")
4 # % (Expect worse predictions for new x than poly7, especially
    outside [1,8]) %

```

2d: Regression spline with bs() (knots 3,6). Compare preds.

```

1 # library(splines)
2 # spline_bs_t3 <- lm(y ~ bs(x, knots=c(3,6)), data=data_t3_task2
    )
3 # lines(data_t3_task2$x, predict(spline_bs_t3), col="green")
4 # % (Predictions should be identical to 2c if basis is
    equivalent, which it is for cubic) %

```

2e: LOOCV for different knot placements

```

1 # testMSE_spline <- function(knot1, knot2, x_data, y_data) {
2 #   # ... (LOOCV loop as in earlier examples, fitting lm with bs
    (x, knots=c(knot1,knot2))) ...
3 #   return(mean_SE_from_LOOCV)
4 # }
5 # # testMSE_spline(3, 6, x_t3_task2, y_t3_task2) % Sol: 2044 %
6 # # testMSE_spline(3.5, 5.5, x_t3_task2, y_t3_task2) % Sol: 377
    (better) %

```

4.3.3 Task 3: Extension of KNN (Nadaraya-Watson)

Implement KNN as weighted average, then Nadaraya-Watson.

3a: Properties of weights w_i (Sum to 1, large w_i for small $|x_i - x_0|$).

3b: Implement KNN as weighted average (knn2)

3c: Modify knn2 to Nadaraya-Watson (nw), plot for different h .

```
1 # weight_nw <- function(x0, x_vec, h_val=0.2) dnorm((x_vec-x0)/h_val) / sum(dnorm((x_vec-x0)/h_val))
2 # nw_func <- function(x0, x_vec, y_vec, h_val=0.2) {
3 #   w <- weight_nw(x0, x_vec, h_val)
4 #   return(sum(wy_vec))
5 # }
6 # # Simulate x_sim, y_sim from exercise
7 # # set.seed(12); x_sim <- seq(-50,50,length=100)
8 # # y_sim <- 1-0.05x_sim-0.02x_sim^2+0.003x_sim^3+50rnorm(100)
9 # # plot(x_sim, y_sim)
10 # # pred_nw_h1 <- sapply(x_sim, nw_func, x_vec=x_sim, y_vec=y_sim, h_val=1)
11 # # lines(x_sim, pred_nw_h1, col="red")
12 # # pred_nw_h5 <- sapply(x_sim, nw_func, x_vec=x_sim, y_vec=y_sim, h_val=5)
13 # # lines(x_sim, pred_nw_h5, col="blue")
```

3d: Meaning of bandwidth h (Controls "localness" of estimator).

4.4 Tutorial 4

4.4.1 Task 1 : Regression Trees (Hitters data)

Predict $\log\text{Salary}$ using Hits and Years .

1a: Explain RSS function and optimise() for first split.

```
1 # library(ISLR); data(Hitters)
2 # Hitters <- Hitters[!is.na(Hitters$Salary),]
3 # Hitters$logSalary <- log(Hitters$Salary)
4 # RSS_func_t4 <- function(s, x1_vec, y_vec) {
5 #   yR1 <- mean(y_vec[x1_vec < s], na.rm=TRUE)
6 #   yR2 <- mean(y_vec[x1_vec >= s], na.rm=TRUE)
7 #   rss <- sum((y_vec[x1_vec < s] - yR1)^2, na.rm=TRUE) +
8 #     sum((y_vec[x1_vec >= s] - yR2)^2, na.rm=TRUE)
9 #   return(rss)
10 # }
11 # opt_hits <- optimise(RSS_func_t4, lower=0, upper=250,
12 #   x1_vec=Hitters$Hits, y_vec=Hitters$logSalary)
13 # % $min=117.5, $obj=160.97 %
14 # opt_years <- optimise(RSS_func_t4, lower=0, upper=50,
15 #   x1_vec=Hitters$Years, y_vec=Hitters$logSalary)
16 # % $min=4.5, $obj=115.06 %
17 # # Years gives lower RSS, so first split is on Years < 4.5
```

1b: Fit tree with tree(), plot. Compare first split.

```
1 # library(tree)
2 # tree1_t4 <- tree(logSalary ~ Hits + Years, data=Hitters)
3 # plot(tree1_t4); text(tree1_t4, pretty=0)
4 # % First split in tree should be Years < 4.5 (or very close) %
```

1c: Predict for new observation (Hits=125, Years=2.5) Follow path in tree1_t4: Years=2.5 (\neq 4.5) -> Left. Hits=125 (\neq 114 if that's next split) -> Right in subtree. Read leaf value. Sol: 5.264

1d: Fit tree with all predictors

```
1 # tree2_t4 <- tree(logSalary ~ . - Salary, data=Hitters) #
  Exclude original Salary
2 # summary(tree1_t4) % RSS is deviance: 69.06 for tree1 %
3 # summary(tree2_t4) % RSS is deviance: 43.03 for tree2 (lower as
  expected) %
4 # rss1_t4 <- sum(residuals(tree1_t4)^2)
5 # rss2_t4 <- sum(residuals(tree2_t4)^2)
```

4.4.2 Task 2: Bagging and Random Forest (Hitters data)

Benchmark: tree2 from T1d.

2a: TestMSE for tree2 (validation set)

```
1 # set.seed(123)
2 # n_hit <- nrow(Hitters)
3 # ind_hit <- sample(1:n_hit, size=floor(n_hit/2))
4 # train_hit <- Hitters[ind_hit,]; test_hit <- Hitters[-ind_hit,]
5 # tree2_train_t4 <- tree(logSalary ~ . - Salary, data=train_hit)
6 # pred2_test_t4 <- predict(tree2_train_t4, newdata=test_hit)
7 # mse2_test_t4 <- mean((test_hit$logSalary - pred2_test_t4)^2) %
  Sol: 0.428 %
```

2b: Bagging of tree2. Variable Importance.

```
1 # library(randomForest)
2 # set.seed(123) % (Ensure consistent seed for RF if comparing) %
3 # # For bagging, mtry = number of predictors (19 for Hitters
  excluding Salary, logSalary)
4 # bagging_t4 <- randomForest(logSalary ~ . - Salary, data=train_
  hit, mtry=19, importance=TRUE)
5 # predbag_t4 <- predict(bagging_t4, newdata=test_hit)
6 # msebag_t4 <- mean((test_hit$logSalary - predbag_t4)^2) % Sol:
  0.3015 %
7 # varImpPlot(bagging_t4)
8 # % x-axis is MeanDecreaseGini (for classification) or
  IncNodePurity (RSS decrease for regression).
9 # Help says for regression: total decrease in node impurities
  (RSS) from splitting on the variable,
10 # averaged over all trees. CRuns most important. %
```

2c: Random Forest. How is it an extension of bagging?

```
1 # library(randomForest)
2 # set.seed(123)
3 # rf_t4 <- randomForest(logSalary ~ . - Salary, data=train_hit,
  mtry=3, importance=TRUE) % (mtry=3 from solution) %
```

```

4 # predrf_t4 <- predict(rf_t4, newdata=test_hit)
5 # mserf_t4 <- mean((test_hit$logSalary - predrf_t4)^2)
6 # % Random Forest is an extension because at each split, only a
  random subset (mtry) of predictors
7 #   is considered, decorrelating trees and potentially
  improving variance reduction. %

```

4.4.3 Task 3: Boosting (Hitters data)

Predict logSalary with boosted tree. Compute testMSE.

```

1 # library(gbm)
2 # # Using train_hit, test_hit from 2a
3 # set.seed(123)
4 # boost_t4 <- gbm(logSalary ~ . - Salary, data=train_hit,
  distribution="gaussian",
5 #               n.trees=1000, interaction.depth=4, shrinkage
  =0.01)
6 # predgbm_t4 <- predict(boost_t4, newdata=test_hit, n.trees
  =1000)
7 # msegbm_t4 <- mean((test_hit$logSalary - predgbm_t4)^2) % Sol:
  0.2666 %
8 # % Boosting often yields lowest MSE among tree ensembles here.
  %

```

4.4.4 Task 4: Support Vector Machines (Simulated Data 'xy.csv')

Classes "1" and "-1".

4a: Possible to find maximal marginal classifier from plot? *(No, classes overlap).*

4b: Recreate plot from 'xy.csv'.

4c: Use SVC (linear kernel) to classify. Plot.

```

1 # xy_data_t4 <- read.csv("xy.csv")
2 # names(xy_data_t4) <- c("x1", "x2", "y_class") % (Assuming
  column names) %
3 # xy_data_t4$y_factor <- as.factor(xy_data_t4$y_class)
4 # library(e1071)
5 # sumfit_linear_t4 <- svm(y_factor ~ x1 + x2, data=xy_data_t4,
  kernel="linear",
6 #                       cost=10, scale=FALSE)
7 # plot(sumfit_linear_t4, xy_data_t4, x2 ~ x1) % (Adjust formula
  for plot if needed) %

```

4d: Meaning of tuning parameter C=10. *(Allows for some misclassifications/margin violations, controls trade-off between margin width and violations. Larger C = narrower margin, fewer violations).*

4e: SVM with polynomial kernel. Better separation?

```

1 # sumfit_poly_t4 <- svm(y_factor ~ x1 + x2, data=xy_data_t4,
  kernel="polynomial",
2 #                       cost=10, scale=FALSE) % (Default degree
  =3) %

```

```

3 # plot(sumfit_poly_t4, xy_data_t4, x2 ~ x1)
4 # % Polynomial kernel can create non-linear boundary, might
   separate training data better,
5 # but risk of overfitting. %

```

4.4.5 Task 5: PCA (advertising.csv)

PCA on TV, radio, newspaper. Interpret first two PCs from loadings.

```

1 # advertising_t5 <- read.csv("advertising.csv")[, c("TV", "radio",
   "newspaper")]
2 # pca_t5 <- prcomp(advertising_t5, scale.=TRUE, center.=TRUE)
3 # summary(pca_t5)
4 # print(pca_t5$rotation)
5 # % PC1: High loading on TV (-0.999), small on others.
   Represents overall ad spend dominated by TV.
6 # PC2: High positive on newspaper (0.93), moderate positive
   on radio (0.36), small on TV. Represents
7 # print media focus vs. TV. (Signs might be flipped but
   interpretation similar). %

```

4.4.6 Task 6: Cluster Analysis (USArrests data)

6a: K-means clustering (K=3) on Murder, Assault. Plot.

```

1 # data(USArrests)
2 # x_km_t6 <- scale(USArrests[, c("Murder", "Assault")]) # Scale
   data
3 # set.seed(1) % (For reproducibility of kmeans) %
4 # km3_t6 <- kmeans(x_km_t6, centers=3, nstart=20)
5 # plot(x_km_t6, col=km3_t6$cluster, pch=19, main="K-Means (K=3)
   on Murder & Assault")
6 # points(km3_t6$centers, col=1:3, pch=4, cex=3, lwd=3)
7 # text(x_km_t6, labels=rownames(USArrests), col=km3_t6$cluster,
   pos=3, cex=0.7)

```

6b: Hierarchical clustering. Compare.

```

1 # Using scaled data x_km_t6 from above
2 # dist_t6 <- dist(x_km_t6)
3 # hclust_complete_t6 <- hclust(dist_t6, method="complete")
4 # plot(hclust_complete_t6, main="Hierarchical Clustering (
   Complete Linkage)", cex=0.7)
5 # clusters_hclust_k3_t6 <- cutree(hclust_complete_t6, k=3)
6 # table(KMeans=km3_t6$cluster, Hierarchical=clusters_hclust_k3_
   t6)
7 # % Compare cluster assignments. Differences due to different
   algorithms/criteria. %

```

5 Assignment: Compulsory Assignment BAN404 (Spring 2025x)

Predicting log(CEO Salary) using the *ceosal2* dataset.

5.1 Initial Data Loading and Preparation

Loading Packages and Data

```
1 # Ensure wooldridge is installed: install.packages("wooldridge")
2 library(wooldridge)
3 library(dplyr) # For tidyverse data manipulation
4 library(ggplot2) # For tidyverse plotting
5 library(glmnet) # For LASSO
6 library(FNN) # For knn() function (alternative to manual
   for Task 3)
7 library(gam) # For GAMs
8 library(boot) # For cv.glm (though LOOCV can be manual for
   lm)
9
10 data(ceosal2)
11 # View data structure and summary
12 # str(ceosal2)
13 # summary(ceosal2)
14 # help(ceosal2) # To understand variables
```

Data Splitting (50/50 Train/Test) As per the assignment's suggested solution, an initial 50/50 split is performed.

Listing 68: Splitting data into 50/50 training and test sets

```
1 set.seed(1234) # Seed from assignment
2 n_total <- nrow(ceosal2)
3 ntrain <- floor(n_total/2)
4 train_indices <- sample(1:n_total, size=ntrain)
5
6 # These original splits might be used if subsequent processing
   is done per task
7 # For consistency with assignment solution, a global processed
   dataset is often made first
8 original_train_data <- ceosal2[train_indices,]
9 original_test_data <- ceosal2[-train_indices,]
```

5.2 Task 1: Descriptive Statistics and Predictor Investigation

Question Describe relevant features of the output variable (*lsalary*) with descriptive statistics (tables and graphs). Also use descriptive statistics to investigate promising predictors for *lsalary*. Remove variables that cannot possibly be available when the salary of a new CEO should be predicted.

Solution

Processing Data for Task 1 Assumptions based on provided solution: New CEO hired internally. *comten* (company tenure) and *comtensq* are available. *ceoten* (CEO tenure in current company) and *ceotensq* are NOT available for a new CEO. *salary* (untransformed) is obviously not available. *college* is removed due to low variability in the training data example in the solution.

Listing 69: Data processing for Task 1 (Applied to whole dataset first, then split)

```
1 # Create the 'lsalary' variable if it's not already present (it
   is in ceosal2)
```



```

2 # ceosal2$lsalary is log(salary)
3
4 # Variables to remove
5 vars_to_remove_t1 <- c("salary", "ceoten", "ceotensq", "college"
6 )
7 ceosal_task1_filtered <- ceosal2[, !(names(ceosal2) %in% vars_to
8 _remove_t1)]
9
10 # Convert 'grad' to factor
11 ceosal_task1_filtered$grad <- as.factor(ceosal_task1_filtered$
12 grad)
13
14 # Now split the PROCESSED data
15 train_t1 <- ceosal_task1_filtered[train_indices,]
16 test_t1 <- ceosal_task1_filtered[-train_indices,]

```

Descriptive Statistics for lsalary (on train_t1)

```

1 # Base R
2 summary(train_t1$lsalary)
3 hist(train_t1$lsalary, main="Histogram of log(Salary) - Training
4 Data", xlab="log(Salary)", col="lightblue", breaks=12)
5 boxplot(train_t1$lsalary, main="Boxplot of log(Salary) -
6 Training Data", ylab="log(Salary)")
7
8 # Tidyverse
9 # library(dplyr); library(ggplot2)
10 # train_t1 %>% select(lsalary) %>% summary()
11 # ggplot(train_t1, aes(x=lsalary)) +
12 #   geom_histogram(aes(y=..density..), fill="lightblue", color="
13 black", bins=12) +
14 #   geom_density(alpha=0.2, fill="red") +
15 #   ggtitle("Distribution of log(Salary) - Training Data")

```

Interpretation: Comment on the distribution of `lsalary` - is it symmetric, skewed, any outliers? The log transformation often makes salary data more symmetric.

Investigating Promising Predictors for lsalary (on train_t1)

```

1 # Base R (as in solution)
2 # par(mfrow=c(3,3)) # Adjust layout based on number of
3 predictors remaining
4 # plot(lsalary ~ ., data=train_t1)
5 # graphics.off()
6
7 # Tidyverse (example for a few predictors)
8 # ggplot(train_t1, aes(x=lsales, y=lsalary)) + geom_point() +
9 #   geom_smooth(method="lm") + ggtitle("lsalary vs lsales")
10 # ggplot(train_t1, aes(x=grad, y=lsalary)) + geom_boxplot() +
11 #   ggtitle("lsalary vs grad")

```

Interpretation of plots: - `lsales` shows a clear positive linear relationship. - `sales` also positive, might capture non-linearity if `lsales` also included. - `mktval`, `lmktval` show positive trends. - For categorical `grad`: Check if means/medians differ significantly.

Listing 70: Statistical test for 'grad' (Task 1)

```

1 # Base R
2 # by(train_t1$lsalary, train_t1$grad, summary)

```



```

3 # t_test_grad_0 <- t.test(train_t1$lsalary[train_t1$grad == 0])
  # If enough observations
4 # t_test_grad_1 <- t.test(train_t1$lsalary[train_t1$grad == 1])
5 # print(t_test_grad_0$conf.int)
6 # print(t_test_grad_1$conf.int)
7 # if (abs(t_test_grad_0$estimate - t_test_grad_1$estimate) is
  small and CIs overlap substantially,
8 # then 'grad' might not be a strong predictor alone).
9
10 # Tidyverse
11 # train_t1 %>% group_by(grad) %>% summarise(mean_lsalar = mean(
  lsalary), sd_lsalar = sd(lsalary))

```

Conclusion from Task 1 based on solution: *lsales* is the most promising. Others warrant investigation in modeling. *grad* seems weak.

5.3 Task 2: Linear Regression and LASSO

Question Use linear regression (on all or a subset of the predictors) and LASSO to predict *lsalary*. Evaluate the predictions on test data using an appropriate error measure (MSE).

Solution We use *train_t1* and *test_t1* from the end of Task 1 preparation.

OLS with All Predictors

```

1 ols_model_all <- lm(lsalary ~ ., data=train_t1)
2 # summary(ols_model_all) # For coefficient details
3
4 ols_all_preds_test <- predict(ols_model_all, newdata=test_t1)
5 ols_all_mse_test <- mean((test_t1$lsalary - ols_all_preds_test)
  ^2)
6 cat("Test MSE (OLS All Predictors):", ols_all_mse_test, "\n")
7 # Solution had Test MSE = 0.3044908

```

OLS with lsales Only

```

1 ols_model_ksales <- lm(lsalary ~ lsales, data=train_t1)
2 # summary(ols_model_ksales)
3
4 ols_ksales_preds_test <- predict(ols_model_ksales, newdata=test_
  t1)
5 ols_ksales_mse_test <- mean((test_t1$lsalary - ols_ksales_preds_
  test)^2)
6 cat("Test MSE (OLS lsales Only):", ols_ksales_mse_test, "\n")
7 # Solution had Test MSE = 0.309084

```

Comment: The test MSEs are very similar, suggesting *lsales* captures most of the predictive power. The solution mentions LOOCV for a more robust comparison, where the *lsales*-only model was better.

LASSO Regression

```

1 library(glmnet)
2 # Prepare matrices for glmnet (factors need to be dummified)
3 # 'grad' is the only factor in train_t1/test_t1
4 x_train_lasso_t2 <- model.matrix(lsalary ~ ., data=train_t1)
  [, -1] # -1 to remove intercept column
5 y_train_lasso_t2 <- train_t1$lsalary
6 x_test_lasso_t2 <- model.matrix(lsalary ~ ., data=test_t1)
  [, -1]

```

```

7
8 set.seed(1234) # For cv.glmnet
9 cv_lasso_t2 <- cv.glmnet(x_train_lasso_t2, y_train_lasso_t2,
  alpha=1, standardize=TRUE)
10 best_lambda_t2 <- cv_lasso_t2$lambda.min
11 # print(paste("Best lambda for LASSO:", best_lambda_t2))
12
13 lasso_model_t2 <- glmnet(x_train_lasso_t2, y_train_lasso_t2,
  alpha=1,
14                          lambda=best_lambda_t2, standardize=TRUE
  )
15 # print("LASSO Coefficients:")
16 # print(coef(lasso_model_t2))
17 # Solution notes its LASSO (on its training split) picked lsales
  and lmktval.
18
19 lasso_preds_test_t2 <- predict(lasso_model_t2, newx=x_test_lasso
  _t2)
20 lasso_mse_test_t2 <- mean((test_t1$lsalary - lasso_preds_test_t2
  )^2)
21 cat("Test MSE (LASSO):", lasso_mse_test_t2, "\n")
22 # Solution LOOCV MSE for LASSO was 0.3777, worse than lsales-
  only OLS.
23 # Compare your test MSE here with the OLS models.

```

5.4 Task 3: KNN Regression

Question Predict *lsalary* by KNN, using only *lsales* as a predictor. Determine *K* using leave-one-out cross-validation. Also, evaluate the predictions and compare with the results in 2.

Solution The assignment solution uses LOOCV on the full processed dataset (*ceosal_final_filtered*) to determine *K*. For strictness, *K* should be determined using only the training portion (*train_t1*) or an inner CV loop if the final evaluation is on *test_t1*. We follow the provided solution's approach for *K* determination for comparability with its results.

Determine Optimal K using LOOCV

```

1 # Using ceosal_final_filtered from Task 1 for LOOCV to find K
2 # (This uses data that will also be in the "test" part of the
  original 50/50 split)
3 knn_reg_func_t3 <- function(x0, x_train_vec, y_train_vec, K_val)
  {
4   d <- abs(x_train_vec - x0)
5   o <- order(d)
6   # Ensure K_val is not larger than the number of available
    neighbors
7   K_actual <- min(K_val, length(o))
8   return(mean(y_train_vec[o[1:K_actual]]))
9 }
10
11 lsales_all_t3 <- ceosal_task1_filtered$lsales % \textit{Using
  the dataset after initial processing} %
12 lsalary_all_t3 <- ceosal_task1_filtered$lsalary
13 n_all_t3 <- nrow(ceosal_task1_filtered)
14 k_max_t3 <- 40 # As per solution
15 loocv_mse_k_t3 <- numeric(k_max_t3 - 1)

```

```

16
17 for (k_val_t3 in 2:k_max_t3) {
18   se_fold_t3 <- numeric(n_all_t3)
19   for (i in 1:n_all_t3) {
20     xtrain_knn_t3 <- lsales_all_t3[-i]
21     ytrain_knn_t3 <- lsalary_all_t3[-i]
22     xtest_knn_t3 <- lsales_all_t3[i]
23     ytest_knn_t3 <- lsalary_all_t3[i]
24
25     pred_knn_fold_t3 <- knn_reg_func_t3(xtest_knn_t3, xtrain_knn
      _t3, ytrain_knn_t3, K_val=k_val_t3)
26     se_fold_t3[i] <- (ytest_knn_t3 - pred_knn_fold_t3)^2
27   }
28   loocv_mse_k_t3[k_val_t3-1] <- mean(se_fold_t3)
29 }
30
31 # plot(2:k_max_t3, loocv_mse_k_t3, type="b", xlab="K", ylab="
      LOOCV MSE", main="LOOCV for KNN K selection")
32 optimal_K_t3 <- (2:k_max_t3)[which.min(loocv_mse_k_t3)]
33 # cat("Optimal K from LOOCV on full processed data:", optimal_K_
      t3, "\n")
34 # Solution found K=24 (index 23 in their 2:kmax loop).
35 min_loocv_mse_knn_t3 <- min(loocv_mse_k_t3)
36 # cat("LOOCV MSE for KNN with K=", optimal_K_t3, ":", min_loocv_
      mse_knn_t3, "\n")
37 # Solution result for K=24: 0.2840139.

```

Comparison with Task 2 Results LOOCV MSE for KNN ($K=[\text{optimal_K_t3}]$): $[\text{min_loocv_mse_knn_t3}]$ From solution's LOOCV for Task 2: - OLS (lsales only) LOOCV MSE: 0.2685442 - LASSO LOOCV MSE: 0.3777567 - OLS (all predictors) LOOCV MSE: 0.4253121 Conclusion: KNN with only *lsales* (optimal K) performs better than LASSO and full OLS in terms of LOOCV MSE, but slightly worse than the simple OLS model using only *lsales*.

5.5 Task 4: KNN with Multiple Predictors and GAM

Question (Part 1 - KNN with Multiple Predictors) Explain how you would use KNN if you had more than one predictor. Two alternative ways to do this have been mentioned in the course. If you would like to you can also write R-code to do this but this is not required to pass the assignment.

Solution (Textual) (Same textual explanation as in Hypothetical Exam 2025 for: 1. Multivariate Distance Metric for KNN, 2. GAM Framework with KNN as Univariate Smoother via Backfitting).

Question (Part 2 - GAM) Argue for which, if any, variables might have a nonlinear relationship with *lsalary* and find a good model within the generalized additive models (GAM) framework. Evaluate the predictions.

Solution (using gam) From Task 1, plots suggested *sales* and *mktval* might have non-linear relationships. The solution investigates these using LOOCV.

Listing 71: GAM for lsalary (Task 4)

```

1 library(gam)

```

```

2 # Using ceosal_final_train, ceosal_final_test (or ceosal_task1_
   filtered for LOOCV as in solution)
3
4 # Visual inspection on training data (as in solution)
5 # gam_inspect_t4 <- gam(lsalary ~ s(sales, df=4) + s(mktval, df
   =4) + lsales + lmktval +
6 #                                     age + grad + comten + profits
   + comtensq + profmarg,
7 #                                     data=train_t1) % \textit{(Using train_t1
   , consistent with solution's Task 1 plots)} %
8 # par(mfrow=c(1,2))
9 # plot(gam_inspect_t4, se=TRUE, terms=c("s(sales, df = 4)", "s(
   mktval, df = 4)"))
10 # graphics.off()
11 # Solution comment based on this: "contribution of non-linear
   mktval looks questionable".
12
13 # LOOCV for GAM with s(sales) and other predictors linearly.
14 # (Using the full processed dataset ceosal_task1_filtered for
   LOOCV as per solution)
15 n_gam_t4 <- nrow(ceosal_task1_filtered)
16 se_gam_s_sales_t4 <- numeric(n_gam_t4)
17
18 # Formula based on solution: only s(sales) is non-linear, others
   linear.
19 # (The solution eventually only uses s(sales) in its final GAM
   LOOCV comparison)
20 formula_gam_s_sales <- lsalary ~ s(sales, df=4)
21 # To compare with other models, it's better to include other
   linear terms if they were in OLS.
22 # For this example, let's test the simple s(sales) model as in
   the solution's final GAM part.
23
24 # for(i in 1:n_gam_t4) {
25 #   train_gam_fold_t4 <- ceosal_task1_filtered[-i,]
26 #   test_gam_fold_t4 <- ceosal_task1_filtered[i,]
27 #
28 #   # Ensure factor levels are handled if factors were included
   in formula_gam_s_sales
29 #   # For lsalary ~ s(sales, df=4), this is not an issue.
30 #   gam_fit_fold_t4 <- gam(formula_gam_s_sales, data=train_gam_
   fold_t4)
31 #   pred_gam_fold_t4 <- predict(gam_fit_fold_t4, newdata=test_
   gam_fold_t4)
32 #   se_gam_s_sales_t4[i] <- (test_gam_fold_t4$lsalary - pred_gam
   _fold_t4)^2
33 # }
34 # mse_gam_s_sales_loocv_t4 <- mean(se_gam_s_sales_t4)
35 # cat("LOOCV MSE (GAM with s(sales, df=4) only):", mse_gam_s_
   sales_loocv_t4, "\n")
36 # Solution result for GAM with s(sales) only: 0.2918272.
37
38 # Comparison:
39 # KNN (K=24, lsales only) LOOCV MSE: 0.2840139
40 # OLS (lsales only) LOOCV MSE: 0.2685442
41 # Conclusion: "GAM with s(sales) is slightly worse than KNN with
   lsales, and also worse than
42 # simple OLS with lsales. The log-transformation in lsales seems
   to capture the main
43 # non-linearity effectively for this dataset."

```