STAT 4360, Fall 2024
Benny Frisella

Mini Project VI

Section 1

Using 'Hitters' data from the ISLR2 library in R. This data includes observations from 263 baseball players with data from 1986 and salary data from 1987.

1.

a. Figure 1 below models the regression tree fitted with: log(Salary) as the response(due to skewness), the full 19 variables as predictors, and the entire dataset for training.

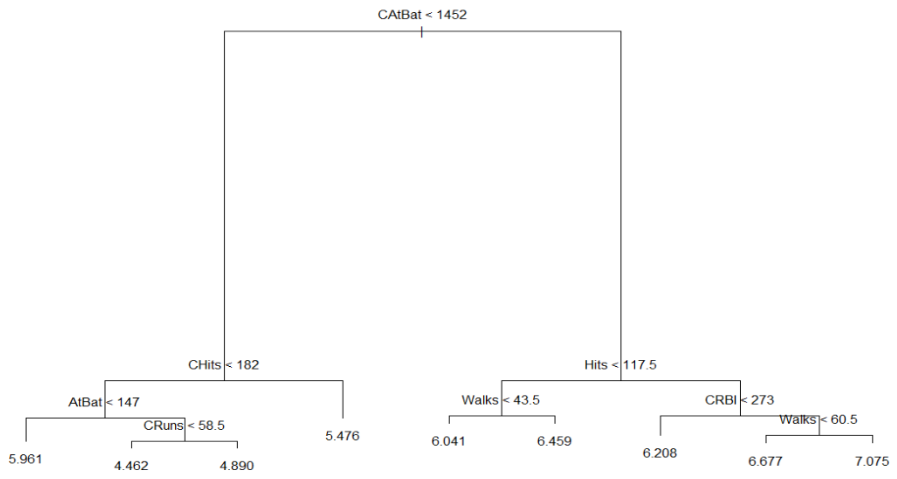


Figure 1

The regions created are:

- R_1: ŷ = 5.961 for CAtBat < 1452 → CHits < 182 → AtBat < 147
- R_2: ŷ = 4.462 for CAtBat < 1452 → CHits <182 → AtBat ≥ 147 → CRuns < 58.5
- R_3: ŷ = 4.890 for CAtBat < 1452 → CHits < 182 → AtBat ≥ 147 → CRuns ≥ 58.5
- R_4: ŷ = 5.476 for CAtBat < 1452 → CHits ≥ 182
- R_5: ŷ = 6.041 for CAtBat ≥ 1452 → Hits < 117.5 → Walks < 43.5
- R_6: ŷ = 6.459 for CAtBat ≥ 1452 → Hits < 117.5 → Walks ≥ 43.5
- R_7: ŷ = 6.208 for CAtBat ≥ 1452 → Hits ≥ 117.5 → CRBI < 273 → Walks < 60.5
- R_8: ŷ = 6.677 for CAtBat ≥ 1452 → Hits ≥ 117.5 → CRBI < 273 → Walks ≥ 60.5
- R_9: ŷ = 7.075 for CAtBat ≥ 1452 → Hits ≥ 117.5 → CRBI ≥ 273.

Where:
- ŷ = predicted average of log(Salary) for the respective region.
- 'C' refers to Career statistics (e.g., Career At Bats, Career Hits, etc.).
- Non-'C' variables represent annual statistics from 1986 (e.g., Hits, Walks).

This regression tree produced a test MSE of 0.3859896 using LOOCV, which could suggest that a simpler model of fewer predictors may perform better as the current model could be overfitting the data.

In summary, it can be seen that *Career At Bats* is at the root of the tree of Figure 1, *Career At Bats likely serves as a proxy for experience and longevity in the league, which correlates with salary.*

Furthermore, when traveling down the right of Figure 1, we see that increases in Hits, Career Runs Batted In, and Walks typically result in increased pay.

Interestingly, when a *player's career and season At Bats* are the lowest, their salary is higher regardless of increases in Career Hits/Runs; possibly reflecting less playing time but higher value per opportunity. This observation is intriguing, but it may also reflect outliers, such as rookies with high salaries or specialized roles and warrants further investigation.

b. An analysis of Cost Complexity showed possible improvements in model performance if the tree was **pruned to *6 predictors;* as the test error stabilizes around that point**. However, pruning resulted in an increase in MSE, as shown below:

- Best Pruned Tree MSE: 0.**387**2823 with **8 predictors**
- Best Un-Pruned Tree MSE: 0.**385**9896 with **9 predictors**

As more predictors were removed, the test MSE rose*. Therefore, the model of 9 predictors selected originally has the best predictive potential while effectively balancing bias and variance. Note, pruning's primary goal is to improve generalization rather than minimize training errors, so this is acceptable.*

As the tree was pruned, the bottom-most predictors were removed one-by-one. As a result, the **most important predictors can be identified from top to bottom in Figure 1**, as previously discussed.

c. **A Bootstrap Aggregation (Bagging) model improves accuracy with a test MSE of 0.2256415.** A summary showed splits occurred at 6 predictors, which aligns with our findings from the Cost Complexity analysis.

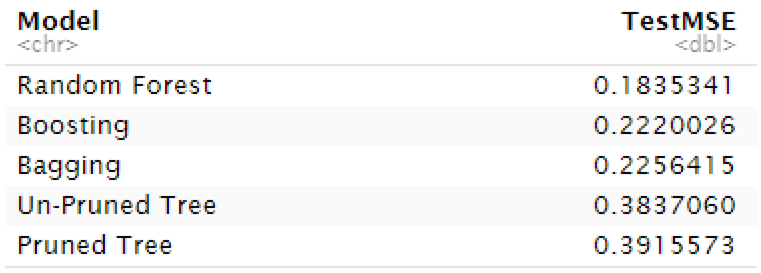
The Variable Importance Plot was used to infer the most important variables as: CAtBat, CRuns, CHits, CRBI, AtBat, Hits from most importance to less importance.

d. **A Random Forest model achieved the best accuracy yet with a test MSE of 0.1846457. Another Variable Importance Plot was used to show the most important variables as: CAtBat, CHits, CRuns, CRBI, CWalks, Hits from most importance to less importance.** The added randomness in Random Forest's model achieved increased separation between the top variable purities but overall lessened the purity values of them all when compared to Bagging purities. This could be due to Random Forest's tendency to distribute importance across more predictors due to decorrelation.

e. **A Boosting Model of 1000 trees, a depth of 1, minimum observations as 10 and a shrinkage parameter of 0.01 returned a test MSE of 0.2220026.** *The relative influence of each successive predictor stabilized after the sixth predictor (Years).*

In terms of descending relative influence, the top predictors are: CAtBat, CHits, CRuns, CRBI, CWalks, and Years as they contribute the most to the overall predictive power of this model.

f. Overall, the best performing models can be seen next in Figure 2.



Model	TestMSE
<chr>	<dbl>
Random Forest	0.1835341
Boosting	0.2220026
Bagging	0.2256415
Un-Pruned Tree	0.3837060
Pruned Tree	0.3915573

Figure 2

Random Forest achieved the lowest MSE and is the best generalized model for prediction out of them all. The model accounts for **71% of the variance explained**, although this measure is acceptable, I hope to improve it through hyperparameter tuning in the future. Previously, I'd recommended a linear regression model with a test MSE of 0.34, but this **Random Forest model nearly halved that regression model's MSE while remaining relatively simple**. **The Random Forest model is the best we currently have.**

2. Exploring the Diabetes dataset again but using newly learned methods and models, I was able to derive a better model for prediction.

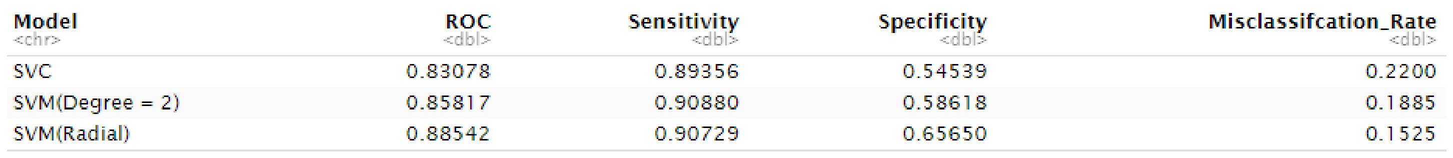
a. Using **Support Vector Classifiers, I consistently achieved a misclassification rate of 0.22** across all adjustments of the cost parameter. Error rate is relatively low, but did not substantially improve with tuning cost. **With a cost parameter of 0.011, penalizing misclassifications more heavily didn't help, meaning the underlying data patterns are unlikely to benefit from more complex boundaries.** Having a low cost also reduced the chance of overfitting while achieving a smoother decision boundary. While the model performed well in predicting True Positives, it only correctly identified True Negatives around half of the time. This leaves room to explore more advanced methods to improve discrimination.

b. A Support Vector Machine (SVM) model with a polynomial **kernel of degree 2** was trained on the Diabetes dataset. The model's hyperparameters, including the **scale parameter (0.55)** and the **cost parameter (C=0.7525**), were tuned using 10-fold cross-validation (CV) to maximize ROC.

The **SVM model achieved a test error rate (misclassification rate) of 0.1885**, indicating a good fit for the data. This performance reflects the model's ability to balance complexity and generalizability, particularly for binary classification in this medical dataset context.

c. **A Support Vector Machine (SVM) with a radial kernel achieved a misclassification rate of 0.1525**. Having both **γ (0.1)** and **cost (1)** chosen optimally via 10-fold CV helped achieve our **lowest error rate yet along with the best Sensitivity (0.907) and Specificity (0.657).**

d. Previous analysis of this Diabetes dataset **showed the POS and OSM methods had a reasonable level of agreement in measuring oxygen saturation; with the bias near zero and small standard error.** However, the standard error of **θ^{hat} was not optimized** as it showed deviance in the estimation of **0.124989**. Figure 3 shows a comparison of the Support Vector models.



Model	ROC	Sensitivity	Specificity	Misclassifcation_Rate
<chr>	<dbl>	<dbl>	<dbl>	<dbl>
SVC	0.83078	0.89356	0.54539	0.2200
SVM(Degree = 2)	0.85817	0.90880	0.58618	0.1885
SVM(Radial)	0.88542	0.90729	0.65650	0.1525

Figure 3

- **SVM(Radial)** appears to be the **best choice overall due to its superior ROC, specificity, and minimized error rate. It balances both error minimization and correct classification of negative cases.**

- **SVM(Degree = 2)** is a close second, **offering the best sensitivity but slightly higher error rate and lower specificity than the radial kernel model.**

- **SVC** should be considered **only if simplicity and faster computation are prioritized**, but its lower specificity and higher misclassification rate make it less optimal compared to the other two models.

3. Considering an *unsupervised Hitters* with the goal of clustering.

a. **Standardizing the variables beforehand is generally a good thing as it allows all variables to contribute equally.** Clustering algorithms like hierarchical clustering and k-means rely on distances between observations, and variables with larger scales can dominate the distance calculations and lead to skewed results.

b. Given that the task is clustering players based on characteristics, which are mostly continuous variables such as performance stats, salaries, **metric-based distance (Euclidean distance) is more suitable here.** If one were concerned with the patterns or relationships between variables, then correlation-based distance could be useful.

c. Standardizing the variables and performing *complete linkage via Euclidean distance (Hierarchical Clustering)* gave the following dendrogram below in Figure 4. This was cut at k = 2 to create two distinct clusters.

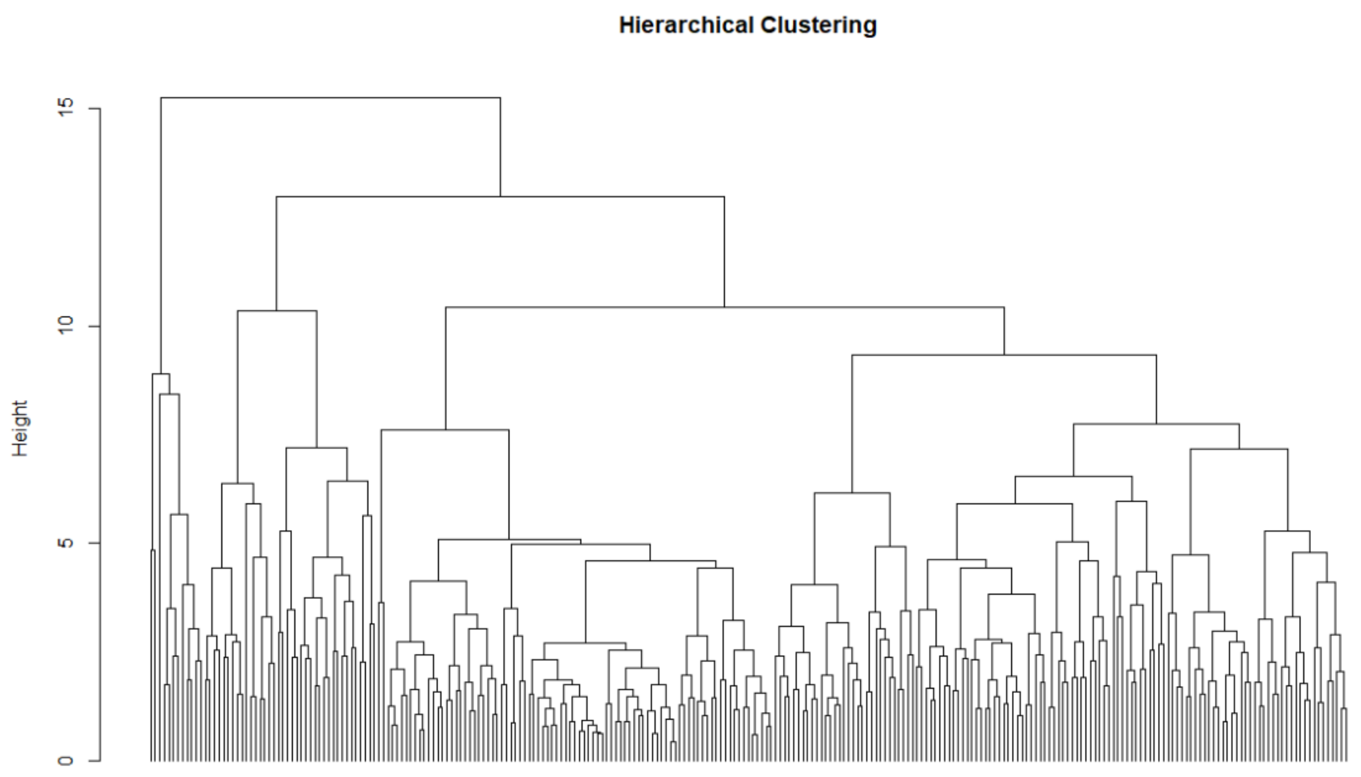
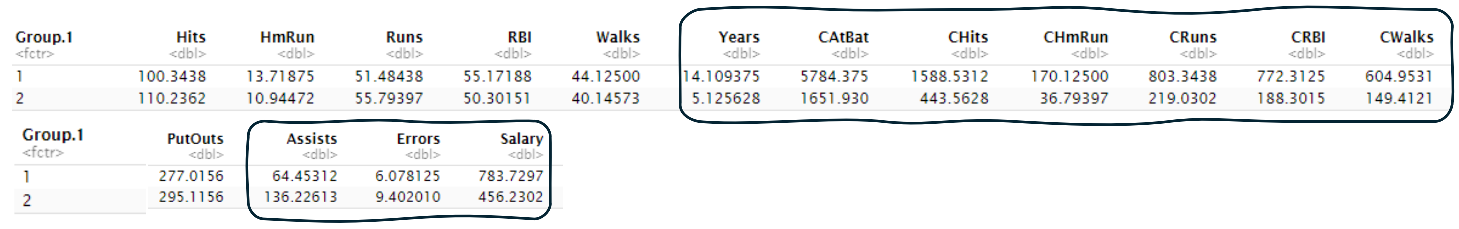


Figure 4

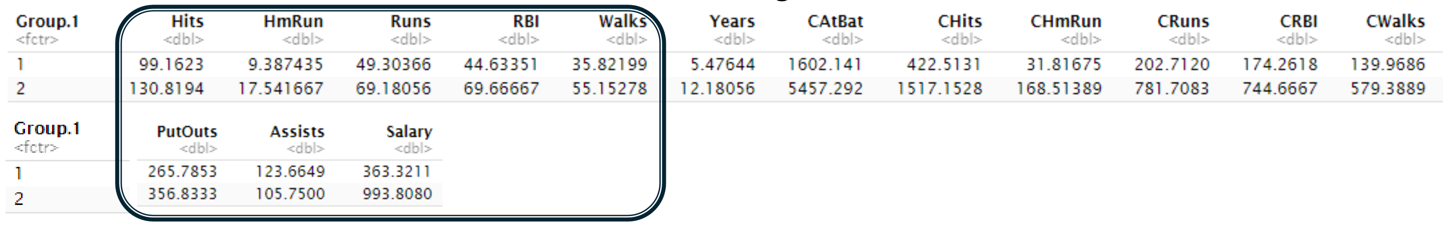


Group.1 <fctr>	Hits <dbl>	HmRun <dbl>	Runs <dbl>	RBI <dbl>	Walks <dbl>	Years <dbl>	CAtBat <dbl>	CHits <dbl>	CHmRun <dbl>	CRuns <dbl>	CRBI <dbl>	CWalks <dbl>	PutOuts <dbl>	Assists <dbl>	Errors <dbl>	Salary <dbl>
1	100.3438	13.71875	51.48438	55.17188	44.12500	14.109375	5784.375	1588.5312	170.12500	803.3438	772.3125	604.9531	277.0156	64.45312	6.078125	783.7297
2	110.2362	10.94472	55.79397	50.30151	40.14573	5.125628	1651.930	443.5628	36.79397	219.0302	188.3015	149.4121	295.1156	136.22613	9.402010	456.2302

Figure 5

Figure 5 highlights the cluster-specific means of each variable, with dramatic differences boxed for clarity. **Cluster 1 includes higher-salaried players, averaging nearly double the salary of Cluster 2.** These players generally have more experience, with higher averages in home runs, RBIs, and walks but lower averages in putouts, assists, and errors, reflecting their offensive focus over defensive contributions.

d. The last model for Hitters featured K-Means clustering with k = 2 and standardized variables.



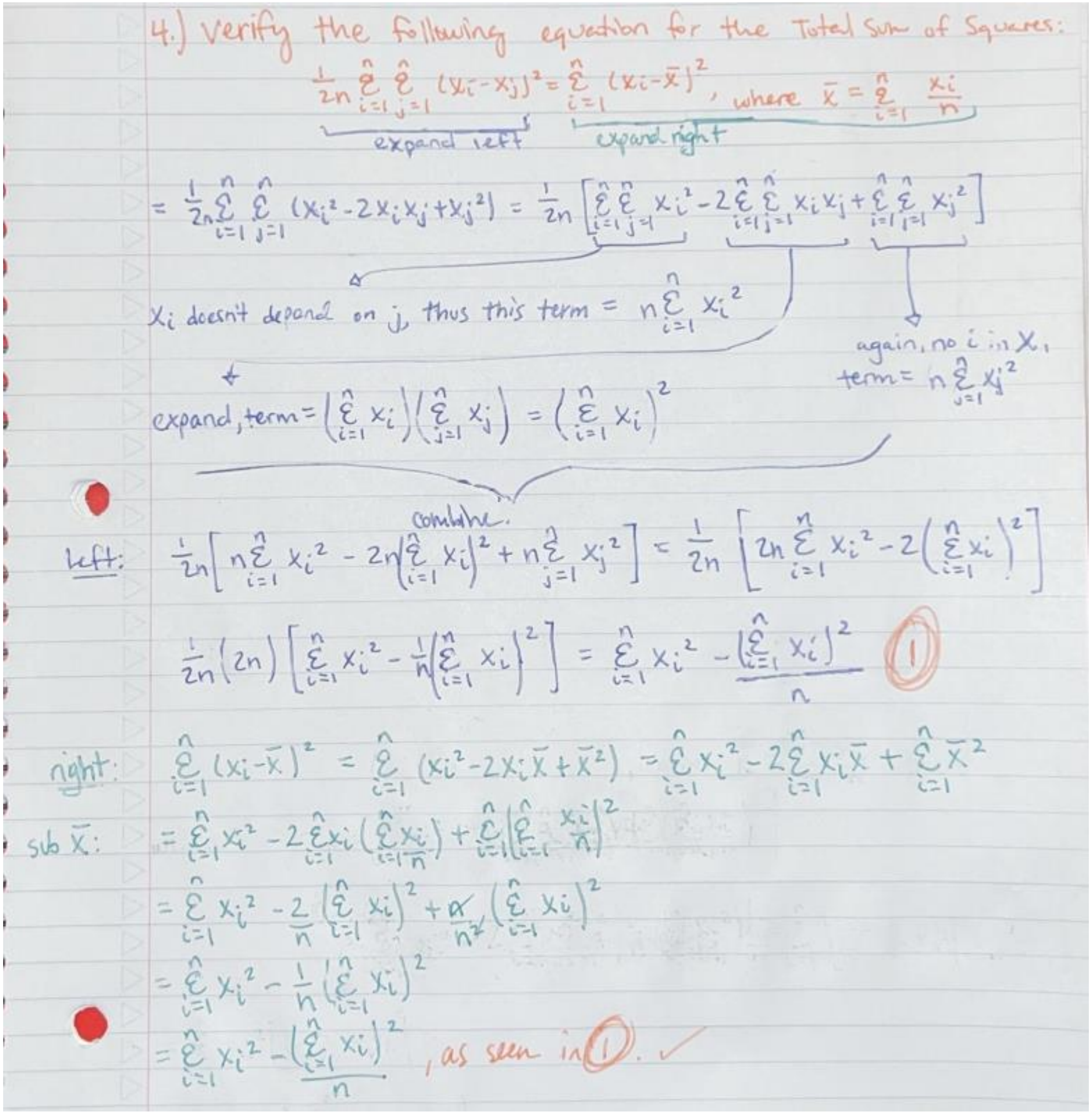
Group.1 <fctr>	Hits <dbl>	HmRun <dbl>	Runs <dbl>	RBI <dbl>	Walks <dbl>	Years <dbl>	CAtBat <dbl>	CHits <dbl>	CHmRun <dbl>	CRuns <dbl>	CRBI <dbl>	CWalks <dbl>	PutOuts <dbl>	Assists <dbl>	Salary <dbl>
1	99.1623	9.387435	49.30366	44.63351	35.82199	5.47644	1602.141	422.5131	31.81675	202.7120	174.2618	139.9686	265.7853	123.6649	363.3211
2	130.8194	17.541667	69.18056	69.66667	55.15278	12.18056	5457.292	1517.1528	168.51389	781.7083	744.6667	579.3889	356.8333	105.7500	993.8080

Figure 6

As seen above in Figure 6, the **high-salary players** were clustered **in 2 with almost _triple_ the average salary of players clustered in 1. Notably, players with increased salary averaged *one-third more hits* and almost _double_ the home runs as their counterparts.**

e. Hierarchical Clustering and K-means produced similar results, with key differences highlighted in Figure 6. **While K-means suggests salary is heavily influenced by offensive performance, the Hierarchical model presents a more balanced view, capturing contributions beyond offense such as defensive skills. Given the nature of the sport, the Hierarchical Clustering model better reflects salary determinants.** However, if a team prioritizes offense above all (home runs, hits, RBIs, etc.), K-means might be more appropriate for predicting player salaries.

4.



4.) verify the following equation for the Total Sum of Squares:

$$\underbrace{\frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i-x_j)^2}_{\text{expand left}}=\underbrace{\sum_{i=1}^{n}(x_i-\bar{x})^2, \text{ where } \bar{x}=\sum_{i=1}^{n}\frac{x_i}{n}}_{\text{expand right}}$$

$$=\frac{1}{2n}\sum_{i=1}^{n}\sum_{j=1}^{n}(x_i^2-2x_ix_j+x_j^2)=\frac{1}{2n}\left[\sum_{i=1}^{n}\sum_{j=1}^{n}x_i^2-2\sum_{i=1}^{n}\sum_{j=1}^{n}x_ix_j+\sum_{i=1}^{n}\sum_{j=1}^{n}x_j^2\right]$$

x_i doesn't depend on j, thus this term $= n\sum_{i=1}^{n}x_i^2$

again, no i in x_j, term $= n\sum_{j=1}^{n}x_j^2$

expand, term $=\left(\sum_{i=1}^{n}x_i\right)\left(\sum_{j=1}^{n}x_j\right)=\left(\sum_{i=1}^{n}x_i\right)^2$

combine.

left: $$\frac{1}{2n}\left[n\sum_{i=1}^{n}x_i^2-2n\left(\sum_{i=1}^{n}x_i\right)^2+n\sum_{j=1}^{n}x_j^2\right]=\frac{1}{2n}\left[2n\sum_{i=1}^{n}x_i^2-2\left(\sum_{i=1}^{n}x_i\right)^2\right]$$

$$\frac{1}{2n}(2n)\left[\sum_{i=1}^{n}x_i^2-\frac{1}{n}\left(\sum_{i=1}^{n}x_i\right)^2\right]=\sum_{i=1}^{n}x_i^2-\frac{\left(\sum_{i=1}^{n}x_i\right)^2}{n} \quad \textcircled{1}$$

right: $$\sum_{i=1}^{n}(x_i-\bar{x})^2=\sum_{i=1}^{n}(x_i^2-2x_i\bar{x}+\bar{x}^2)=\sum_{i=1}^{n}x_i^2-2\sum_{i=1}^{n}x_i\bar{x}+\sum_{i=1}^{n}\bar{x}^2$$

sub $\bar{x}$: $$=\sum_{i=1}^{n}x_i^2-2\sum_{i=1}^{n}x_i\left(\sum_{i=1}^{n}\frac{x_i}{n}\right)+\sum_{i=1}^{n}\left(\sum_{i=1}^{n}\frac{x_i}{n}\right)^2$$

$$=\sum_{i=1}^{n}x_i^2-\frac{2}{n}\left(\sum_{i=1}^{n}x_i\right)^2+\frac{\cancel{n}}{n^{\cancel{2}}}\left(\sum_{i=1}^{n}x_i\right)^2$$

$$=\sum_{i=1}^{n}x_i^2-\frac{1}{n}\left(\sum_{i=1}^{n}x_i\right)^2$$

$$=\sum_{i=1}^{n}x_i^2-\frac{\left(\sum_{i=1}^{n}x_i\right)^2}{n}$$, as seen in $\textcircled{1}$. ✓

Section 2 R Code

~~~
---
title: "Mini Project 6"
author: "Benny Frisella"
date: "2024-12-07"
output: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
~~~
```{r}
library(ISLR)
library(tree)

#?tree

##     Question 1     ##

# (a)

#omit NA
Hitters <- na.omit(Hitters)

summary(Hitters)

#convert categorical columns to factor
categorical <- which(sapply(Hitters, is.character))

Hitters[categorical] <- lapply(Hitters[categorical], as.factor)

Hitters.formula <- log(Salary) ~ .

#str(Hitters)

#create log() Regression Tree #
tree.Hitters <- tree(log(Hitters$Salary) ~ ., Hitters)

summary(tree.Hitters)

#plot
plot(tree.Hitters)
text(tree.Hitters, pretty = 0)

#use LOOCV for test MSE
LOOCV.tree <- cv.tree(tree.Hitters, K = nrow(Hitters))
```

~~~~
tree.MSE <- mean(LOOCV.tree$dev / length(Hitters$Salary))  #average MSE

tree.MSE


```

```{r}

# (b)

#try cost complexity pruning by LOOCV
which.min(LOOCV.tree$size)

#plot estimated test error
plot(LOOCV.tree$size, LOOCV.tree$dev, type = "b", main = "Cost Complexity Pruning",
    ylab = "test error", xlab = "Size")
    mtext("(Using LOOCV)", side = 4, line = 1, cex = 1)

prune.salary <- prune.tree(tree.Hitters, best = 8)

plot(prune.salary)
text(prune.salary, pretty = 0)

#use LOOCV for test MSE
LOOCV.tree <- cv.tree(prune.salary, K = nrow(Hitters))
prune.MSE <- mean(LOOCV.tree$dev / length(Hitters$Salary))  #average MSE

prune.MSE

```

```{r}

# (c)

library(randomForest)
library(caret)

#bagging model w/ B = 1000 trees
bagging.model <- randomForest(log(Salary) ~ ., data = Hitters, ntree = 1000, mtry = ncol(Hitters) - 1)

bagging.model

#get test MSE
testMSE <- bagging.model$mse[1000]  #get value for the last tree
cat("Bagging Test MSE:", testMSE, "\n")

#get important predictors
~~~~

~~~~
important.values <- importance(bagging.model)
print(important.values)

#plot
varImpPlot(bagging.model, main = "Bagging")

#Bagging with LOOCV
bag_model <- train(log(Salary) ~ ., data = Hitters, method = "treebag", trControl = ctrl)

print(bag_model) #rquared = 71%

#get predictions
predictions_bag <- bag_model$pred$pred

#get actual values
actual_bag <- bag_model$pred$obs

#get MSE
mse_bag <- mean((predictions_bag - actual_bag)^2)
print(mse_bag)

```

```{r}

# (d)

#random forest with m ≈ p/3 (default)
randForest.model <- randomForest(log(Salary) ~ ., data = Hitters, ntree = 1000)

# Check Random Forest results
print(randForest.model)

# Estimated Test MSE (Out-of-Bag Error)
randForest.model$mse[1000]

#plot
varImpPlot(randForest.model, main = "Random Forest")



#prep
ctrl <- trainControl(method = "LOOCV")

#Random Forest with LOOCV
rf_model <- train(log(Salary) ~ ., data = Hitters, method = "rf", trControl = ctrl)

#get predictions
predictions <- rf_model$pred$pred
~~~~

~~~~
#get actual values
actual <- rf_model$pred$obs

#get MSE
mse_rand <- mean((predictions - actual)^2)
print(mse_rand)



```


```{r}
#combine
layout(matrix(c(1, 2), nrow = 1, ncol = 2))

varImpPlot(bagging.model, main = "Bagging")
varImpPlot(randForest.model, main = "Random Forest")

```

```{r}

# (e)

library(gbm)

#fit boosting with LOOCV
boosting.model <- train(log(Salary) ~ ., data = Hitters,
           method = "gbm", trControl = trainControl("LOOCV"),
           tuneGrid = expand.grid(interaction.depth = 1,
                     n.trees = 1000,
                     shrinkage = 0.01, #lambda
                     n.minobsinnode = 10),
           verbose = FALSE)

#relative influence
summary(boosting.model)

#see
boosting.model$results

#get MSE
boostMSE <- boosting.model$results$RMSE^2  # RMSE squared gives MSE
print(boostMSE)


```
~~~~

~~~~
```{r}

# (f)

#create test MSE table
model_comparison <- data.frame(
  Model = c("Random Forest", "Boosting", "Bagging", "Un-Pruned Tree", "Pruned Tree"),
  TestMSE = c(mse_rand, boostMSE, mse_bag, tree.MSE, prune.MSE)
)

# Print the comparison table
print(model_comparison)

```

```{r}
### Question 2 ###

# (a)
setwd("~/University of Texas @ Dallas/Statistical Learning/Datasets")

#prep
diabetes <- read.csv("diabetes.csv")

#factor needed
diabetes$Outcome <- as.factor(diabetes$Outcome)

#level error fix
levels(diabetes$Outcome) <- c("Class0", "Class1")

formula <- Outcome ~ .

#10-fold control
ctrl <- trainControl(method = "cv", number = 10, classProbs = TRUE, summaryFunction =
twoClassSummary)

#fit SVC
svc_model <- train(formula, data = diabetes, method = "svmLinear", trControl = ctrl, metric = "ROC",
          tuneLength = 10)
svc_model

#test error
svc_pred <- predict(svc_model, diabetes)
svc_error <- mean(svc_pred != diabetes$Outcome)
cat("SVC Test Error Rate:", svc_error, "\n")

#tune cost
~~~~

~~~
#grid of cost values for tuning
cost_grid <- expand.grid(C = seq(0.001, 0.5, by = 0.005)) #CHANGE THIS, BEST = 0.011

svc_model <- train(formula, data = diabetes, method = "svmLinear", trControl = ctrl,
 tuneGrid = cost_grid,
 metric = "ROC")

#find best model
print(svc_model)

#get cost
best_cost <- svc_model$bestTune$C
cat("Best cost parameter:", best_cost, "\n")

#use cost
tuned_grid <- expand.grid(C = 6.501)

#train again
svm_model <- train(Outcome ~ ., data = diabetes, method = "svmLinear", trControl = ctrl,
          tuneGrid = tuned_grid)

print(svm_model)

```

```{r}
# (b)

#SVM
set.seed(123)
poly_svm_model <- train(Outcome ~ ., data = diabetes, method = "svmPoly",trControl = ctrl,
 tuneGrid = expand.grid(degree = 2, scale = seq(0.1, 1, length = 5), C = seq(0.01, 1, length = 5)),
 metric = "ROC")

#show
print(poly_svm_model)
poly_svm_model$bestTune  #optimal
poly_svm_model$results   #performance

#test MSE
svm_pred <- predict(poly_svm_model, diabetes)
svm_error <- mean(svm_pred != diabetes$Outcome)
cat("SVM Test Error Rate:", svm_error, "\n")


```

```{r}
# (c)
~~~

~~~
#radial SVM
set.seed(123)
rbf_svm_model <- train(Outcome ~ ., data = diabetes, method = "svmRadial", trControl = ctrl,
  tuneGrid = expand.grid(C = seq(0.01, 1, length = 5), sigma = seq(0.01, 0.1, length = 5)),
  metric = "ROC")

#get results
print(rbf_svm_model)
rbf_svm_model$bestTune  #optimal parameters
rbf_svm_model$results   #performance

#test MSE
rbf_pred <- predict(rbf_svm_model, diabetes)
rbf_error <- mean(rbf_pred != diabetes$Outcome)
cat("RBF Test Error Rate:", rbf_error, "\n")


```

```{r}

# (e) compare
roc_linear <- 0.83078
sens_linear <- 0.89356
spec_linear <- 0.54539

roc_poly <- 0.85817
sens_poly <- 0.90880
spec_poly <- 0.58618

roc_radial <- 0.88542
sens_radial <- 0.90729
spec_radial <- 0.65650

sv_results <- data.frame(
  Model = c("SVC", "SVM(Degree = 2)", "SVM(Radial)"),
  ROC = c(roc_linear, roc_poly, roc_radial),
  Sensitivity = c(sens_linear, sens_poly, sens_radial),
  Specificity = c(spec_linear, spec_poly, spec_radial),
  Misclassifcation_Rate = c(svc_error, svm_error, rbf_error)
)

# Print the results table
print(sv_results)

```

```{r}
~~~

```
### Question 3 ###

#(c)

library(factoextra) # For visualization

#omit NA
Hitters <- na.omit(Hitters)

#only work with numeric
Hitters_numeric <- Hitters[, sapply(Hitters, is.numeric)]

#standardize
data_standardized <- scale(Hitters_numeric)

#Hierarchical clustering w/ Euclidean distance/complete linkage
diss_matrix <- dist(data_standardized, method = "euclidean")
hclust_model <- hclust(diss_matrix, method = "complete")

#cut model create 2 clusters
clusters <- cutree(hclust_model, k = 2)

#get means
cluster_summary <- aggregate(Hitters[, -c(1)], by = list(Hitters$Cluster), FUN = mean)

salary_summary <- aggregate(Hitters$Salary, by = list(Hitters$Cluster), FUN = mean)

print(cluster_summary)
print(salary_summary)

#dress down/up tree
library(dendextend)

#create dendogram
dend <- as.dendrogram(hclust_model)

#remove leafs
dend <- dend %>% set("labels", rep("", length(labels(dend))))

#plot
plot(dend,
   main = "Hierarchical Clustering",
   xlab = "",
   ylab = "Height",
   sub = "",
   cex = 0.8)
```
```` ``` ````
````

~~~
```{r}

# (d)


#K-means clustering w/ K = 2
set.seed(123)  # For reproducibility
kmeans_model <- kmeans(data_standardized, centers = 2)

#add labels
Hitters$Cluster_Kmeans <- as.factor(kmeans_model$cluster)

#summarize means
kmeans_summary <- aggregate(Hitters[, -c(1, 18)], by = list(Hitters$Cluster_Kmeans), FUN = mean)

#summarize mean salaries
salary_summary_kmeans <- aggregate(Hitters$Salary, by = list(Hitters$Cluster_Kmeans), FUN = mean)

print(kmeans_summary)
print(salary_summary_kmeans)


```
~~~