ADVANCED DATA MINING AND PREDICTIVE ANALYTICS

(BA-64037-003)

SPRING 2024

ASSIGNMENT – 2

DIVYA CHANDRASEKARAN

ID: 811284790

Question – 1

What is the key idea behind bagging? Can bagging deal both with high variance (overfitting) and high bias (underfitting)?

Answer:

Bagging, also known as bootstrap aggregating, is an ensemble learning strategy that enhances prediction performance by aggregating multiple models. The central concept of bagging is to diminish the variance of single-model predictions through averaging. This approach is especially beneficial for models susceptible to overfitting, characterized by strong performance on training data but weak performance on new, unseen data.

Yes, bagging can address both high variance and high bias issues..

High Variance (Overfitting):

- In case of high variance (overfitting), bagging helps by lowering the variance without substantially altering the bias.
- Bagging takes advantage of the fact that individual models trained on different subsets of the data will have different sources of error.
- By averaging the predictions of these models, the random errors tend to cancel out, reducing the overall variance of the final model.
- This helps to mitigate the problem of overfitting, where a single model may capture the noise in the training data too closely.

High Bias (Underfitting):

- For models with high bias (underfitting), while bagging might not enhance the bias directly, it can still boost overall performance through variance reduction. Typically, bagging shows the most effectiveness with models prone to high variance.
- Bagging can also help to reduce the bias of the final model, although this is more limited.
- By combining multiple models, each of which may have a slightly different bias, the final model can capture more complex patterns in the data, reducing the overall bias.
- However, if the individual models have a high bias, the final bagged model will also have a high bias, and may still underfit the data.

For instance, let's consider a scenario where individual decision trees, known for their high variance, serve as the foundational models. Each of these trees could potentially overfit the data. However, by averaging the outputs of several such trees, as seen in Random Forests, the variance in the predictions decreases, making the overall model less prone to overfitting.

In summary, bagging is primarily effective in reducing the variance of a single model, but it can also help to some extent in reducing the bias of the final model. Nevertheless, it can also aid models with high bias, provided the bias is moderately low.

QUESTION 2

Why bagging models are computationally more efficient when compared to boosting models with the same number of weak learners?

Answer:

Bagging and boosting are both types of ensemble learning methods that utilize multiple weak learners to enhance prediction accuracy.

Bagging is often more computationally efficient compared to boosting because it allows for the parallel training of individual weak learners. This efficiency stems from the fact that these learners operate independently and do not require sequential training. Moreover, bagging tends to be more resilient to outliers, meaning that anomalies in the training data have a diminished effect on the model's overall performance. In contrast, boosting models train learners sequentially, where each model depends on the errors of the previous model to adjust its own training process.

Conversely, boosting, when it comes to simplicity in training, involves training weak learners in sequence, where each learner focuses on correcting the mistakes made by its predecessors. This sequential dependency can make boosting considerably more computationally intensive than bagging, particularly with larger datasets. Boosting, however, involves adjusting the weights of incorrectly predicted instances and can involve additional complexities like gradient computation in the case of algorithms like Gradient Boosting.

Example: Utilizing a multi-core processor, one can train an ensemble of 100 decision trees simultaneously in a bagging setup, such as a Random Forest. However, with AdaBoost, a well-known boosting technique, each tree must be trained sequentially, with each new tree increasingly concentrating on the instances that previous trees misclassified.

QUESTION-3

James is thinking of creating an ensemble mode to predict whether a given stock will go up or down in the next week. He has trained several decision tree models but each model is not performing any better than a random model. The models are also very similar to each other. Do you think creating an ensemble model by combining these tree models can boost the performance? Discuss your answer.

Answer:

Merging James' decision tree models into an ensemble could potentially enhance the accuracy of predicting stock price movements, although success is not certain. Ensemble methods are known to be effective in a range of machine learning applications, including the prediction of stock markets, by leveraging the combined capabilities of multiple models to reduce error and increase adaptability.

However, given that James' current decision tree models perform no better than chance, it's essential to resolve the fundamental problems before combining them into an ensemble. If the

individual models are already underperforming, merely aggregating or merging their outputs is unlikely to markedly boost overall precision.

The similarity among James' decision tree models indicates potential overfitting to the training data. Overfitting happens when a model is too closely aligned with the training data, compromising its ability to perform well with new data. This often stems from using too many features or excessively complex models.

Simply combining similar poor performers of these models is unlikely to enhance performance significantly. For an ensemble to be effective, the individual models should at least be somewhat accurate in their own right or make different kinds of errors that can cancel out during aggregation.

Once James rectifies the issues with each decision tree, he might then explore constructing an ensemble model. Ensemble methods tend to perform better when the individual models exhibit diverse strengths and weaknesses.

Here are some recommended strategies for James for showing better performance:

1. Enhance Model Variability: James should consider increasing the diversity of the decision trees by adjusting parameters such as tree depth, the features selected at each split, or even using various types of algorithms.

2. Data and Feature Analysis: A thorough reevaluation of the data, along with adjustments to feature selection and preprocessing methods, could reveal more effective predictive features or expose issues with data quality.

3. Explore Different Ensemble Methods: If bagging proves inadequate, James could look into boosting to better address predictions in challenging cases, or try stacking, which integrates different model types.

For example, if James's trees are uniformly shallow, they might be overlooking critical data complexities. Deepening some trees or varying the feature selection across different trees could introduce the diversity needed for a successful ensemble.

QUESTION – 5

Consider the following Table that classifies some objects into two classes of edible (+) and non- edible (-), based on some characteristics such as the object color, size and shape. What would be the Information gain for splitting the dataset based on the "Size" attribute?

Answer:

Calculate the entropy of the original dataset based on the "Edible?" attribute.

$Number\ of\ edible\ objects\ (+)$: **9**

$Numbers\ of\ non-edible\ objects\ (-)$: **7**

$total\ number\ of\ objects$: **16**

$$Edible\ Entropy\ (parent)\text{Entropy (S)} = -\left(\text{p}(+) \times log_2(\text{p}(+)) + \text{p}(-) \times log_2(p(-))\right)$$

$$Entropy\ (S) = -[\left(\frac{9}{16}\right) \times log_2\left(\frac{9}{16}\right) + \left(\frac{7}{16}\right) \times log_2\left(\frac{7}{16}\right)$$

$$entropy = \mathbf{0.988}$$

$$entropy\ of\ small\ size = -\left(\left(\frac{6}{8}\right) \cdot log_2\left(\frac{6}{8}\right)\right) - \left(\frac{2}{8}\right) \cdot log_2\left(\frac{2}{8}\right)) = \mathbf{0.811}$$

$$large\ size\ entropy = -\left(\left(\frac{3}{8}\right) \cdot log_2\left(\frac{3}{8}\right)\right) - \left(\left(\frac{5}{8}\right) \cdot log_2\left(\frac{5}{8}\right)\right) = \mathbf{0.954}$$

The average Entropy of the child is as follows:

$$\left(0.811\left(\tfrac{8}{16}\right)\right) + \left(0.955\left(\tfrac{8}{16}\right)\right) = \mathbf{0.883}$$

$$Information\ gain =$$

$$Entropy(Orginal\ Dataset) - Weighted\ Average\ of\ the\ Entropy\ of\ the\ Split\ Datasets$$

$$Information\ Gain = (0.988 - 0.883) = \mathbf{0.105}$$

The information gain of **0.105** indicates that the "Size" attribute is moderately useful for predicting whether an object is edible or not. This is a relatively small information gain, but it is still significant. It means that we can expect to improve the accuracy of our model by using the "Size" attribute to predict whether an object is edible or not.

QUESTION – 5

Why is it important that the m parameter (number of attributes available at each split) to be optimally set in random forest models? Discuss the implications of setting this parameter too small or too large.

Answer:

Why is it crucial to optimally configure the "m" parameter (number of attributes available at each split) in random forest models? Let's delve into the repercussions of setting this parameter either too small or too large.

Setting "m" too small leads to underfitting, where each decision tree in the random forest accesses only a limited subset of features during training. This limitation can hinder the trees' ability to grasp the full complexity of the data, resulting in subpar predictive performance. Moreover, a small "m" reduces diversity within the ensemble, diminishing one of random forests' strengths: creating varied decision trees through random feature selection. Consequently, the ensemble's predictive capability may diminish.

On the other hand, setting "m" too large can reduce decorrelation among individual trees. This reduction in decorrelation is a key mechanism for enhancing prediction accuracy in random forests. Additionally, a larger "m" can lead to overly complex decision trees prone to overfitting, especially when data includes noise or irrelevant features. Overfitting negatively impacts generalization to new data, resulting in poorer predictive performance.

Therefore, striking the right balance with the "m" parameter is vital for ensuring optimal model performance in random forests, avoiding both underfitting and overfitting scenarios.

PART B

```
#Loading the required libraries

library(ISLR)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

library(glmnet)

## Warning: package 'glmnet' was built under R version 4.3.3

## Loading required package: Matrix

## Loaded glmnet 4.1-8

library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(ggplot2)
library(lattice)
library(Matrix)
library(rpart)
library(rpart.plot)

## Warning: package 'rpart.plot' was built under R version 4.3.2

#For this assignment, we only need the following attributes: “Sales”, “Price”
, “Advertising”, “Population”, “Age”, “Income” and “Education”. The goal of t
he assignment is to build models to predict the sales of the carseats (“Sales
” attribute) using the other attributes. We can use the dplyr select function
to select these attributes.

Carseats_filterd <-Carseats%>%select("Sales","Price","Advertising","Populatio
n","Age","Income","Education")
```

QUESTION- B1

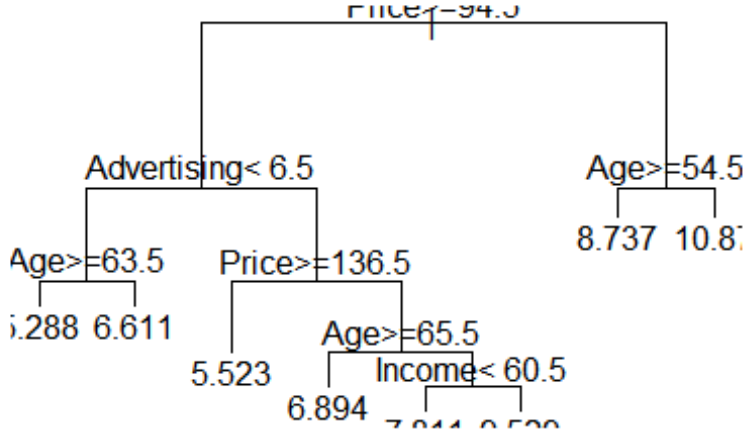
```
#QUESTION B1
#Build a decision tree regression model to predict Sales based on all other a
ttributes ("Price", "Advertising", "Population","Age", "Income" and "Educatio
n"). Which attribute is used at the top of the tree (the root node) for split
ting? Hint: you can either plot () and text() functions or use the summary()
function to see the decision tree rules.

#Checking which attribute comes on the to of the tree

Model <-rpart(Sales~.,data = Carseats_filterd,method = 'anova')
summary(Model)

## Call:
## rpart(formula = Sales ~ ., data = Carseats_filterd, method = "anova")
##   n= 400
##
##            CP nsplit rel error    xerror       xstd
## 1  0.14251535      0 1.0000000 1.0042941 0.06944422
## 2  0.08034146      1 0.8574847 0.9089165 0.06308232
## 3  0.06251702      2 0.7771432 0.9213499 0.06456214
## 4  0.02925241      3 0.7146262 0.8346440 0.05866821
## 5  0.02537341      4 0.6853738 0.8281316 0.05879570
## 6  0.02127094      5 0.6600003 0.8379283 0.05740385
## 7  0.02059174      6 0.6387294 0.8274788 0.05643589
## 8  0.01632010      7 0.6181377 0.8284325 0.05600006
## 9  0.01521801      8 0.6018176 0.7985455 0.05483469
## 10 0.01042023      9 0.5865996 0.8272157 0.05935479
## 11 0.01000559     10 0.5761793 0.8544611 0.05841863
## 12 0.01000000     12 0.5561681 0.8536669 0.05834747
##
## Variable importance
##       Price Advertising         Age      Income  Population   Education
##          49          18          16           8           6           3
##
## Node number 1: 400 observations,    complexity param=0.1425153
##   mean=7.496325, MSE=7.955687
##   left son=2 (329 obs) right son=3 (71 obs)
##   Primary splits:
##       Price       < 94.5  to the right, improve=0.14251530, (0 missing)
##       Advertising < 7.5   to the left,  improve=0.07303226, (0 missing)
##       Age         < 61.5  to the right, improve=0.07120203, (0 missing)
##       Income      < 61.5  to the left,  improve=0.02840494, (0 missing)
##       Population  < 174.5 to the left,  improve=0.01077467, (0 missing)
##
## Node number 2: 329 observations,    complexity param=0.08034146
##   mean=7.001672, MSE=6.815199
##   left son=4 (174 obs) right son=5 (155 obs)
##   Primary splits:
```

```
##       Advertising < 6.5   to the left,  improve=0.11402580, (0 missing)
##       Price       < 136.5 to the right, improve=0.08411056, (0 missing)
##       Age         < 63.5  to the right, improve=0.08091745, (0 missing)
##       Income      < 60.5  to the left,  improve=0.03394126, (0 missing)
##       Population  < 23    to the left,  improve=0.01831455, (0 missing)
##   Surrogate splits:
##       Population < 223   to the left,  agree=0.599, adj=0.148, (0 split)
##       Education  < 10.5  to the right, agree=0.565, adj=0.077, (0 split)
##       Age        < 53.5  to the right, agree=0.547, adj=0.039, (0 split)
##       Income     < 114.5 to the left,  agree=0.547, adj=0.039, (0 split)
##       Price      < 106.5 to the right, agree=0.544, adj=0.032, (0 split)
##
## Node number 3: 71 observations,    complexity param=0.02537341
##   mean=9.788451, MSE=6.852836
##   left son=6 (36 obs) right son=7 (35 obs)
##   Primary splits:
##       Age        < 54.5  to the right, improve=0.16595410, (0 missing)
##       Price      < 75.5  to the right, improve=0.08365773, (0 missing)
##       Income     < 30.5  to the left,  improve=0.03322169, (0 missing)
##       Education  < 10.5  to the right, improve=0.03019634, (0 missing)
##       Population < 268.5 to the left,  improve=0.02383306, (0 missing)
##   Surrogate splits:
##       Advertising < 4.5   to the right, agree=0.606, adj=0.200, (0 split)
##       Price       < 73    to the right, agree=0.592, adj=0.171, (0 split)
##       Population  < 272.5 to the left,  agree=0.592, adj=0.171, (0 split)
##       Income      < 79.5  to the right, agree=0.592, adj=0.171, (0 split)
##       Education   < 11.5  to the left,  agree=0.577, adj=0.143, (0 split)
##
## Node number 4: 174 observations,    complexity param=0.02127094
##   mean=6.169655, MSE=4.942347
##   left son=8 (58 obs) right son=9 (116 obs)
##   Primary splits:
##       Age         < 63.5  to the right, improve=0.078712160, (0 missing)
##       Price       < 130.5 to the right, improve=0.048919280, (0 missing)
##       Population  < 26.5  to the left,  improve=0.030421540, (0 missing)
##       Income      < 67.5  to the left,  improve=0.027749670, (0 missing)
##       Advertising < 0.5   to the left,  improve=0.006795377, (0 missing)
##   Surrogate splits:
##       Income     < 22.5  to the left,  agree=0.678, adj=0.034, (0 split)
##       Price      < 96.5  to the left,  agree=0.672, adj=0.017, (0 split)
##       Population < 26.5  to the left,  agree=0.672, adj=0.017, (0 split)
##
## Node number 5: 155 observations,    complexity param=0.06251702
##   mean=7.935677, MSE=7.268151
##   left son=10 (28 obs) right son=11 (127 obs)
##   Primary splits:
##       Price       < 136.5 to the right, improve=0.17659580, (0 missing)
##       Age         < 73.5  to the right, improve=0.08000201, (0 missing)
##       Income      < 60.5  to the left,  improve=0.05360755, (0 missing)
##       Advertising < 13.5  to the left,  improve=0.03920507, (0 missing)
```

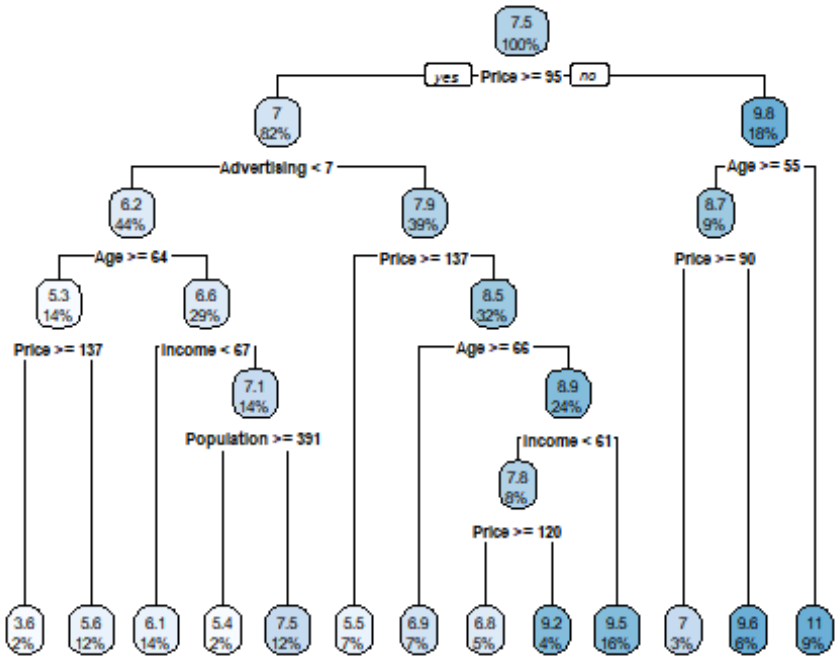
```
##       Population  < 399   to the left,  improve=0.01037956, (0 missing)
##   Surrogate splits:
##       Advertising < 24.5  to the right, agree=0.826, adj=0.036, (0 split)
##
## Node number 6: 36 observations,    complexity param=0.0163201
##   mean=8.736944, MSE=4.961043
##   left son=12 (12 obs) right son=13 (24 obs)
##   Primary splits:
##       Price       < 89.5  to the right, improve=0.29079360, (0 missing)
##       Income      < 39.5  to the left,  improve=0.19043350, (0 missing)
##       Advertising < 11.5  to the left,  improve=0.17891930, (0 missing)
##       Age         < 75.5  to the right, improve=0.04316067, (0 missing)
##       Education   < 14.5  to the left,  improve=0.03411396, (0 missing)
##   Surrogate splits:
##       Advertising < 16.5  to the right, agree=0.722, adj=0.167, (0 split)
##       Income      < 37.5  to the left,  agree=0.722, adj=0.167, (0 split)
##       Age         < 56.5  to the left,  agree=0.694, adj=0.083, (0 split)
##
## Node number 7: 35 observations
##   mean=10.87, MSE=6.491674
##
## Node number 8: 58 observations,    complexity param=0.01042023
##   mean=5.287586, MSE=3.93708
##   left son=16 (10 obs) right son=17 (48 obs)
##   Primary splits:
##       Price      < 137   to the right, improve=0.14521540, (0 missing)
##       Education  < 15.5  to the right, improve=0.07995394, (0 missing)
##       Income     < 35.5  to the left,  improve=0.04206708, (0 missing)
##       Age        < 79.5  to the left,  improve=0.02799057, (0 missing)
##       Population < 52.5  to the left,  improve=0.01914342, (0 missing)
##
## Node number 9: 116 observations,    complexity param=0.01000559
##   mean=6.61069, MSE=4.861446
##   left son=18 (58 obs) right son=19 (58 obs)
##   Primary splits:
##       Income     < 67    to the left,  improve=0.05085914, (0 missing)
##       Population < 392   to the right, improve=0.04476721, (0 missing)
##       Price      < 127   to the right, improve=0.04210762, (0 missing)
##       Age        < 37.5  to the right, improve=0.02858424, (0 missing)
##       Education  < 14.5  to the left,  improve=0.01187387, (0 missing)
##   Surrogate splits:
##       Education   < 12.5  to the right, agree=0.586, adj=0.172, (0 split)
##       Age         < 58.5  to the left,  agree=0.578, adj=0.155, (0 split)
##       Price       < 144.5 to the left,  agree=0.569, adj=0.138, (0 split)
##       Population  < 479   to the right, agree=0.560, adj=0.121, (0 split)
##       Advertising < 2.5   to the right, agree=0.543, adj=0.086, (0 split)
##
## Node number 10: 28 observations
##   mean=5.522857, MSE=5.084213
##
```

```
## Node number 11: 127 observations,    complexity param=0.02925241
##   mean=8.467638, MSE=6.183142
##   left son=22 (29 obs) right son=23 (98 obs)
##   Primary splits:
##       Age         < 65.5  to the right, improve=0.11854590, (0 missing)
##       Income      < 51.5  to the left,  improve=0.08076060, (0 missing)
##       Advertising < 13.5  to the left,  improve=0.04801701, (0 missing)
##       Education   < 11.5  to the right, improve=0.02471512, (0 missing)
##       Population  < 479   to the left,  improve=0.01908657, (0 missing)
##
## Node number 12: 12 observations
##   mean=7.038333, MSE=2.886964
##
## Node number 13: 24 observations
##   mean=9.58625, MSE=3.834123
##
## Node number 16: 10 observations
##   mean=3.631, MSE=5.690169
##
## Node number 17: 48 observations
##   mean=5.632708, MSE=2.88102
##
## Node number 18: 58 observations
##   mean=6.113448, MSE=3.739109
##
## Node number 19: 58 observations,    complexity param=0.01000559
##   mean=7.107931, MSE=5.489285
##   left son=38 (10 obs) right son=39 (48 obs)
##   Primary splits:
##       Population  < 390.5 to the right, improve=0.10993270, (0 missing)
##       Price       < 124.5 to the right, improve=0.07534567, (0 missing)
##       Advertising < 0.5   to the left,  improve=0.07060488, (0 missing)
##       Age         < 45.5  to the right, improve=0.04611510, (0 missing)
##       Education   < 11.5  to the right, improve=0.03722944, (0 missing)
##
## Node number 22: 29 observations
##   mean=6.893793, MSE=6.08343
##
## Node number 23: 98 observations,    complexity param=0.02059174
##   mean=8.933367, MSE=5.262759
##   left son=46 (34 obs) right son=47 (64 obs)
##   Primary splits:
##       Income      < 60.5  to the left,  improve=0.12705480, (0 missing)
##       Advertising < 13.5  to the left,  improve=0.07114001, (0 missing)
##       Price       < 118.5 to the right, improve=0.06932216, (0 missing)
##       Education   < 11.5  to the right, improve=0.03377416, (0 missing)
##       Age         < 49.5  to the right, improve=0.02289004, (0 missing)
##   Surrogate splits:
##       Education < 17.5  to the right, agree=0.663, adj=0.029, (0 split)
##
```

```
## Node number 38: 10 observations
##   mean=5.406, MSE=2.508524
##
## Node number 39: 48 observations
##   mean=7.4625, MSE=5.381106
##
## Node number 46: 34 observations,    complexity param=0.01521801
##   mean=7.811471, MSE=4.756548
##   left son=92 (19 obs) right son=93 (15 obs)
##   Primary splits:
##       Price       < 119.5 to the right, improve=0.29945020, (0 missing)
##       Advertising < 11.5  to the left,  improve=0.14268440, (0 missing)
##       Income      < 40.5  to the right, improve=0.12781140, (0 missing)
##       Population  < 152   to the left,  improve=0.03601768, (0 missing)
##       Age         < 49.5  to the right, improve=0.02748814, (0 missing)
##   Surrogate splits:
##       Education   < 12.5  to the right, agree=0.676, adj=0.267, (0 split)
##       Advertising < 7.5   to the right, agree=0.647, adj=0.200, (0 split)
##       Age         < 53.5  to the left,  agree=0.647, adj=0.200, (0 split)
##       Population  < 240   to the right, agree=0.618, adj=0.133, (0 split)
##       Income      < 41.5  to the right, agree=0.618, adj=0.133, (0 split)
##
## Node number 47: 64 observations
##   mean=9.529375, MSE=4.5078
##
## Node number 92: 19 observations
##   mean=6.751053, MSE=3.378915
##
## Node number 93: 15 observations
##   mean=9.154667, MSE=3.273025

#According to the above summary details most important attribute on deciding
sales is “price”.So it is the attribute that should come on the top of the tr
ee model. This can be proved by the following plots.

#Less complex plot

decision_tree <- rpart(Sales~.,data = Carseats_filterd,method = "anova",contr
ol =rpart.control(minsplit = 60))
plot(decision_tree)
text(decision_tree)
```




```
#fancy RpartPlot
decision_tree2 <- rpart(Sales~.,data=Carseats_filterd,method='anova')
rpart.plot(decision_tree2)
```



QUESTION – B2

```
#QUESTION B2
#QB2. Consider the following input* • Sales=9 • Price=6.54 • Population=124 • Advertising=0 • Age=76 • Income= 110 • Education=10 What will be the estimated Sales for this record using the decision tree model?

mydata <-data.frame(Price=6.54,Population=124,Advertising=0,Age=76,Income=110,Education=10)
estimated_sales<-predict(decision_tree2,mydata)
estimated_sales

##       1
## 9.58625

#According to the above information, the estimated sales for this record using decision tree model is 9.58625.
```

QUESTION- B4

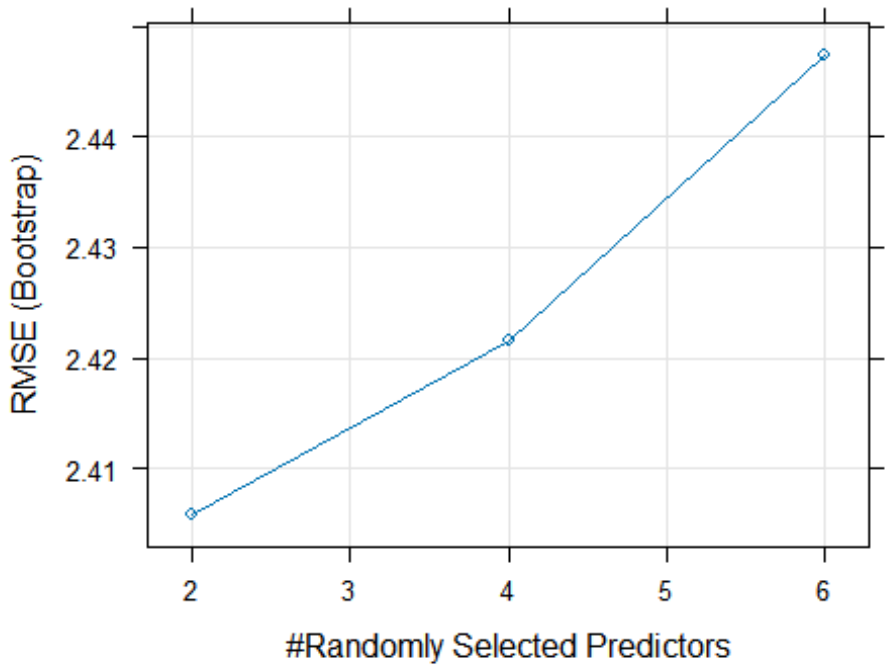
```
#QUESTION B3
#QB3. Use the caret function to train a random forest (method='rf') for the same dataset. Use the caret default settings. By default, caret will examine the "mtry" values of 2,4, and 6. Recall that mtry is the number of attributes available for splitting at each splitting node. Which mtry value gives the best performance?

set.seed(123)
random_forest <-train(Sales~.,data = Carseats_filterd,method='rf')
summary(random_forest)

##                 Length Class      Mode
## call               4   -none-     call
## type               1   -none-     character
## predicted        400   -none-     numeric
## mse              500   -none-     numeric
## rsq              500   -none-     numeric
## oob.times        400   -none-     numeric
## importance         6   -none-     numeric
## importanceSD       0   -none-     NULL
## localImportance    0   -none-     NULL
## proximity          0   -none-     NULL
## ntree              1   -none-     numeric
## mtry               1   -none-     numeric
## forest            11   -none-     list
## coefs              0   -none-     NULL
## y                400   -none-     numeric
## test               0   -none-     NULL
## inbag              0   -none-     NULL
## xNames             6   -none-     character
## problemType        1   -none-     character
```

```
## tuneValue         1    data.frame list
## obsLevels         1    -none-     logical
## param             0    -none-     list
```

```
print(random_forest)
```

```
## Random Forest
##
## 400 samples
##   6 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 400, 400, 400, 400, 400, 400, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     2.405819  0.2852547  1.926801
##   4     2.421577  0.2790266  1.934608
##   6     2.447373  0.2681323  1.953147
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

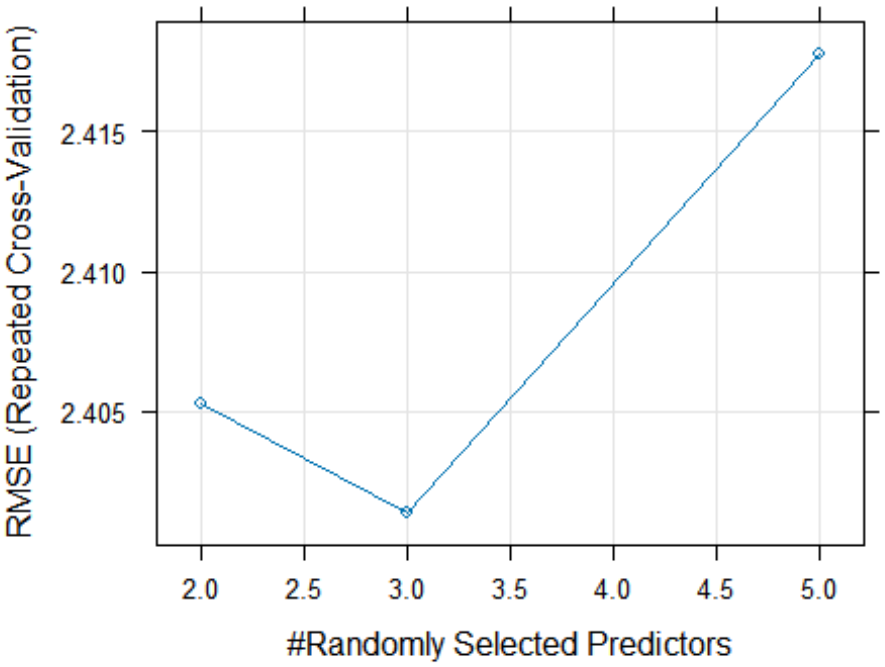
```
plot(random_forest)
```



```
#According to the above summary, RMSE is the lowest when mtry=2. Therefore, m
try 2 gives the best performance.
```

QUESTION- B4

```
#QUESTION B4
#Customize the search grid by checking the model's performance for mtry value
s of 2, 3 and 5 using 3 repeats of 5-fold cross validation.

library(caret)
set.seed(123)
C_grid <- trainControl(method = "repeatedcv",number = 5,repeats = 3,search ="
grid")
C_grid2 <- expand.grid(.mtry=c(2,3,5))
RF_grid <- train(Sales~.,data=Carseats_filterd,method="rf",tuneGrid=C_grid2,t
rControl=C_grid)
print(RF_grid)

## Random Forest
##
## 400 samples
##   6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 3 times)
## Summary of sample sizes: 320, 321, 319, 320, 320, 319, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     2.405235  0.2813795  1.930855
##   3     2.401365  0.2858295  1.920612
##   5     2.417771  0.2821938  1.934886
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 3.

plot(RF_grid)
```




```
#According to the above summary, RMSE is the lowest when mtry=3. Therefore, m
try 3 gives the best performance
```