COMPSCIX 415.2 Homework 9/Final

Vishnu Vardhan

3/30/2018

Contents

1 Bootstrapping (10 points)

1.1 Q1

Follow these steps: • Load the train.csv dataset into R. • Convert all character columns into unordered factors. • Convert the Survived column into an unordered factor because it is loaded as an integer by default. • Take a glimpse of your data to confirm that all of the columns were converted correctly. We will use this same dataset for this entire assignment.

1.1.1 Answer

```
t = read_csv("train.csv")
```

```
## Parsed with column specification:
## cols(
##   PassengerId = col_integer(),
##   Survived = col_integer(),
##   Pclass = col_integer(),
##   Name = col_character(),
##   Sex = col_character(),
##   Age = col_double(),
##   SibSp = col_integer(),
##   Parch = col_integer(),
##   Ticket = col_character(),
##   Fare = col_double(),
##   Cabin = col_character(),
##   Embarked = col_character()
## )
```

```
# After multiple attempts at using purrr and variations..i found
# https://stackoverflow.com/questions/9251326/convert-data-frame-column-format-from-character-to-factor
character_col <- lapply(t, class) == "character"
t[,character_col] <- lapply(t[,character_col], as.factor)

# ordered factors, for better display later
t$Survived <- factor( case_when(
  t$Survived == 0 ~ "Dead",
  t$Survived == 1 ~ "Alive"
), levels = c("Dead", "Alive"))
t$Pclass <- factor(t$Pclass, levels = c("1","2","3"))
glimpse(t)
```

```
## Observations: 891
## Variables: 12
## $ PassengerId <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,...
## $ Survived    <fctr> Dead, Alive, Alive, Alive, Dead, Dead, Dead, Dead...
## $ Pclass      <fctr> 3, 1, 3, 1, 3, 3, 1, 3, 3, 2, 3, 1, 3, 3, 3, 2, 3...
## $ Name        <fctr> Braund, Mr. Owen Harris, Cumings, Mrs. John Bradl...
## $ Sex         <fctr> male, female, female, female, male, male, male, m...
## $ Age         <dbl> 22, 38, 26, 35, 35, NA, 54, 2, 27, 14, 4, 58, 20, ...
## $ SibSp       <int> 1, 1, 0, 1, 0, 0, 0, 3, 0, 1, 1, 0, 0, 1, 0, 0, 4,...
## $ Parch       <int> 0, 0, 0, 0, 0, 0, 0, 1, 2, 0, 1, 0, 0, 5, 0, 0, 1,...
## $ Ticket      <fctr> A/5 21171, PC 17599, STON/O2. 3101282, 113803, 37...
## $ Fare        <dbl> 7.2500, 71.2833, 7.9250, 53.1000, 8.0500, 8.4583, ...
## $ Cabin       <fctr> NA, C85, NA, C123, NA, NA, E46, NA, NA, NA, G6, C...
## $ Embarked    <fctr> S, C, S, S, S, Q, S, S, S, C, S, S, S, S, S, S, Q...
```

1.2 Q2

Use the code below to take 100 bootstrap samples of your data. Confirm that the result is a tibble with a list column of resample objects - each resample object is a bootstrap sample of the titanic dataset.

1.2.1 Answer

Interesting how the bootstrap sample size is the same as the original data size, the only thing different is that it is a "sample" with replacement.

```
# determinted by bootstrap
sample_size = 1000
number_of_samples = 100
titanic_boot <- bootstrap(data = t, n = number_of_samples)
glimpse(titanic_boot)
```

```
## Observations: 100
## Variables: 2
## $ strap <list> [<1.0000, 2.0000, 3.0000, 4.0000, 5.0000, 6.0000, 7.000...
## $ .id   <chr> "001", "002", "003", "004", "005", "006", "007", "008", ...
```

1.3 Q3

Confirm that some of your bootstrap samples are in fact bootstrap samples (meaning they should have some rows that are repeated). You can use the n_distinct() function from dplyr to see that your samples have different numbers of unique rows. Use the code below to help you extract some of the resample objects from the strap column (which is an R list), convert them to tibbles, and then count distinct rows. Use the code below, no changes necessary.

1.3.1 Answer

Clearly, though there 891 samples in the data set, the number of unique samples is less than 600 in every case, showing significant number of repeating samples. Also interesting is that bootstrap creates 100 samples, but each sample as 1000 observations (sample size = 1000)

```
as.tibble(titanic_boot$strap[[1]]) %>% n_distinct()
```

```
## [1] 539
```

```
as.tibble(titanic_boot$strap[[2]]) %>% n_distinct()
```

```
## [1] 566
```

```
as.tibble(titanic_boot$strap[[3]]) %>% n_distinct()
```

```
## [1] 585
```

1.4 Q4

Now, let's demonstrate the Central Limit Theorem using the Age column. We'll iterate through all 100 bootstrap samples, take the mean of Age, and collect the results. • We will define our own function to pull out the mean of Age from each bootstrap sample and • create our own for loop to iterate through. Use the code below and fill in the blanks.

1.4.1 Answer

I used the purrr map function to write the same thing, instead of the code provided.

Using purrr functions

```
age_mean <- function(my_samp) {
  return (mean(as.tibble(my_samp)$Age, na.rm = TRUE))
}

my_means <- as.tibble(titanic_boot$strap %>% map_dbl(.,age_mean))
```

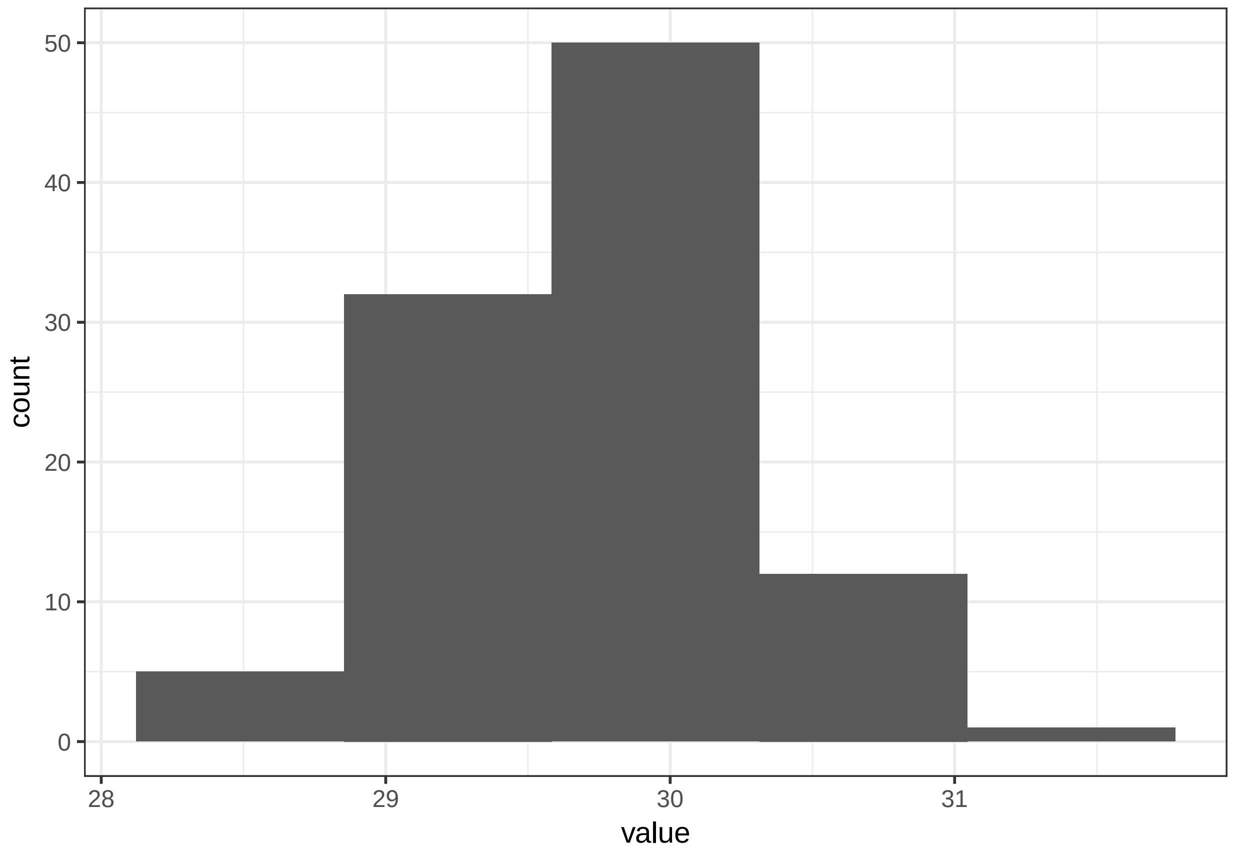
1.5 Q5

Plot a histogram of all_means.

1.5.1 Answer

A histogram shows that the distribution is almost normal.

```r
ggplot(data = my_means) + geom_histogram(aes(x=value), bins=number_of_samples/20) + theme_bw()
```



1.6 Q6

Find the standard error of the sample mean of Age using your boostrap sample means. Compare the empirical standard error to the theoretical standard error.

1.6.1 Answer

The standard error is defined as se = std.deviation thereotical se = sd / sqrt(size)

```r
sprintf ("Empirical standard error of the means   = %f",sd(my_means$value))
```

```
## [1] "Empirical standard error of the means   = 0.550775"
```

```r
age <- t %>% filter ( is.na(Age) == FALSE) %>% select(Age)
sprintf ("Thereotical standard error of the means = %f",sd(age$Age) / sqrt(sample_size))
```

```
## [1] "Thereotical standard error of the means = 0.459368"
```

2 Random Forest (10 points)

On the last homework, we fit a decision tree to the Titanic data set to predict the probability of survival given the features. This week we'll use the random forest and compare our results to the decision tree.

2.1 Q1

Randomly split your data into training and testing using the code below so that we all have the same sets.

2.1.1 Answer

```
set.seed(987)
model_data <- resample_partition(t, c(test = 0.3, train = 0.7))
train_set  <- as.tibble(model_data$train)
test_set   <- as.tibble(model_data$test)
```

2.2 Q2

Fit a decision tree to train_set using the rpart package, and using Pclass, Sex, Age, SibSp, Parch, Fare, Embarked as the features. • Plot the tree using the partykit package. • What do you notice about this tree compared to the one from last week which only contained three features?

2.2.1 Answer

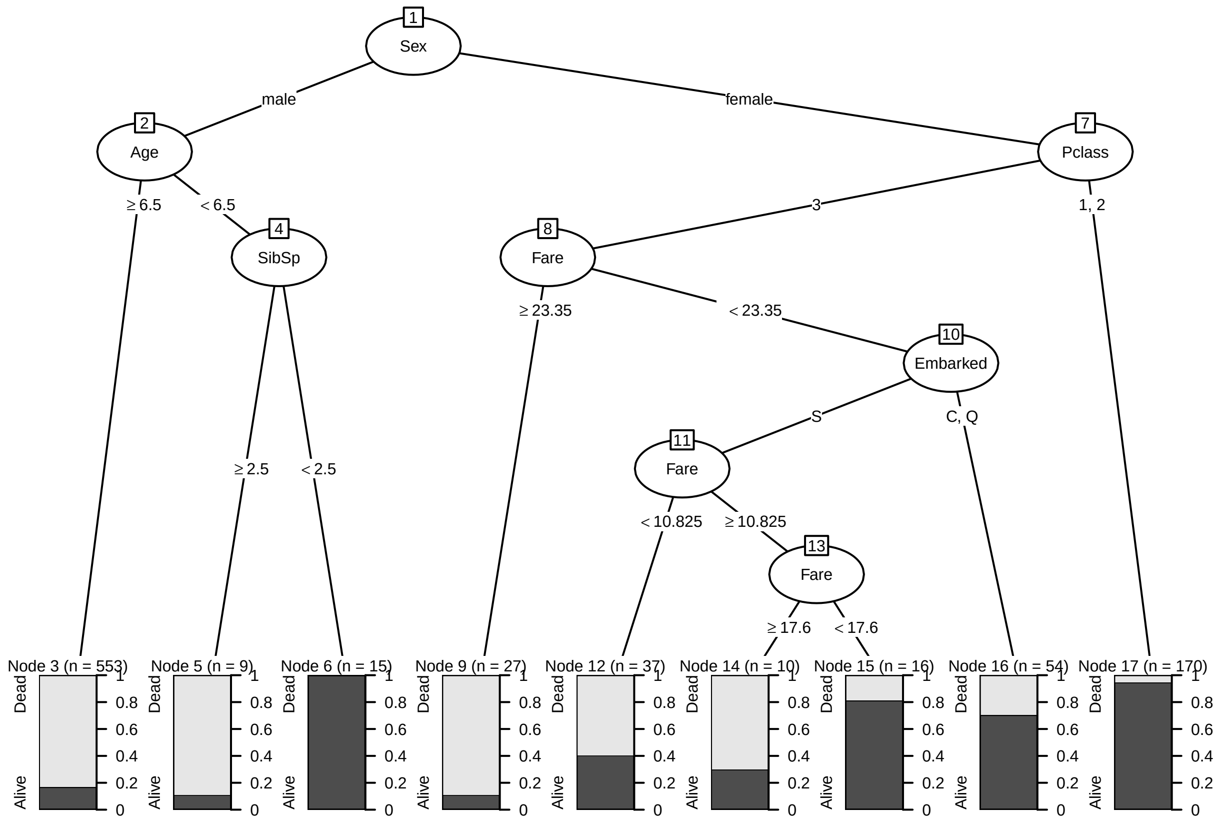
I have not yet completed last weeks assignment, but this tree uses 6 features.

```
model <- rpart ( Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked, data = t, method = "c

print (as.party(model))
```

```
##
## Model formula:
## Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked
##
## Fitted party:
## [1] root
## |   [2] Sex in male
## |   |   [3] Age >= 6.5: Dead (n = 553, err = 16.8%)
## |   |   [4] Age < 6.5
## |   |   |   [5] SibSp >= 2.5: Dead (n = 9, err = 11.1%)
## |   |   |   [6] SibSp < 2.5: Alive (n = 15, err = 0.0%)
## |   [7] Sex in female
## |   |   [8] Pclass in 3
## |   |   |   [9] Fare >= 23.35: Dead (n = 27, err = 11.1%)
## |   |   |   [10] Fare < 23.35
## |   |   |   |   [11] Embarked in S
## |   |   |   |   |   [12] Fare < 10.825: Dead (n = 37, err = 40.5%)
## |   |   |   |   |   [13] Fare >= 10.825
## |   |   |   |   |   |   [14] Fare >= 17.6: Dead (n = 10, err = 30.0%)
## |   |   |   |   |   |   [15] Fare < 17.6: Alive (n = 16, err = 18.8%)
## |   |   |   |   [16] Embarked in C, Q: Alive (n = 54, err = 29.6%)
## |   |   [17] Pclass in 1, 2: Alive (n = 170, err = 5.3%)
##
## Number of inner nodes:    8
## Number of terminal nodes: 9
```

```r
plot  (as.party(model),gp = gpar(fontsize = 6))
```



2.3 Q3

Fit a random forest to train_set using the randomForest package, and using Pclass, Sex, Age, SibSp, Parch, Fare, Embarked as the features. We'll use 500 trees and sample four features at each split. Use the code below and fill in the blanks.

2.3.1 Answer

```r
rf_model <- randomForest(Survived ~ Pclass + Sex + Age + SibSp + Parch + Fare + Embarked,
                         data = train_set, ntrees = 500, mtry = 4, na.action = na.roughfix )
```

2.4 Q4

Compare the performance of the decision tree with the random forest using the ROCR package and the AUC. Which model performs the best?

2.4.1 Answer

As shown below, the area under the curve is greater for the random forest model (0.86 vs 0.81) as compared to decision tree. Thus the random forest is the better model.

```r
rf_prediction   = predict(rf_model, newdata = test_set, type = "prob")
tree_prediction = predict(model, newdata = test_set, type = "prob")

rf_prediction_evaluator = prediction(predictions = rf_prediction[,2],
                                     labels = test_set[[2]],
                                     label.ordering = c("Dead", "Alive") )

tree_prediction_evaluator = prediction(predictions = tree_prediction[,2],
                                       labels = test_set[[2]],
                                       label.ordering = c("Dead", "Alive") )


rf_auc   <- performance(rf_prediction_evaluator, measure = "auc")@y.values
tree_auc <- performance(tree_prediction_evaluator, measure = "auc")@y.values

sprintf("Area under the curve for random forest is: %s", rf_auc)
```

```
## [1] "Area under the curve for random forest is: 0.862157303370787"
```

```r
sprintf("Area under the curve for decision tree is: %s",tree_auc)
```

```
## [1] "Area under the curve for decision tree is: 0.816355140186916"
```

2.5 Q5

Plot the ROC curves for the decision tree and the random forest above, on the same plot with a legend that differentiates and specifies which curve belongs to which model. Use the code below to get you started. Hints: • You will have to modify the plot_roc() function to plot the two curves together with different colors and a legend. • This is easier to do if the data for plotting the two curves are in one tibble. You can combine tibbles using the bind_rows() function.

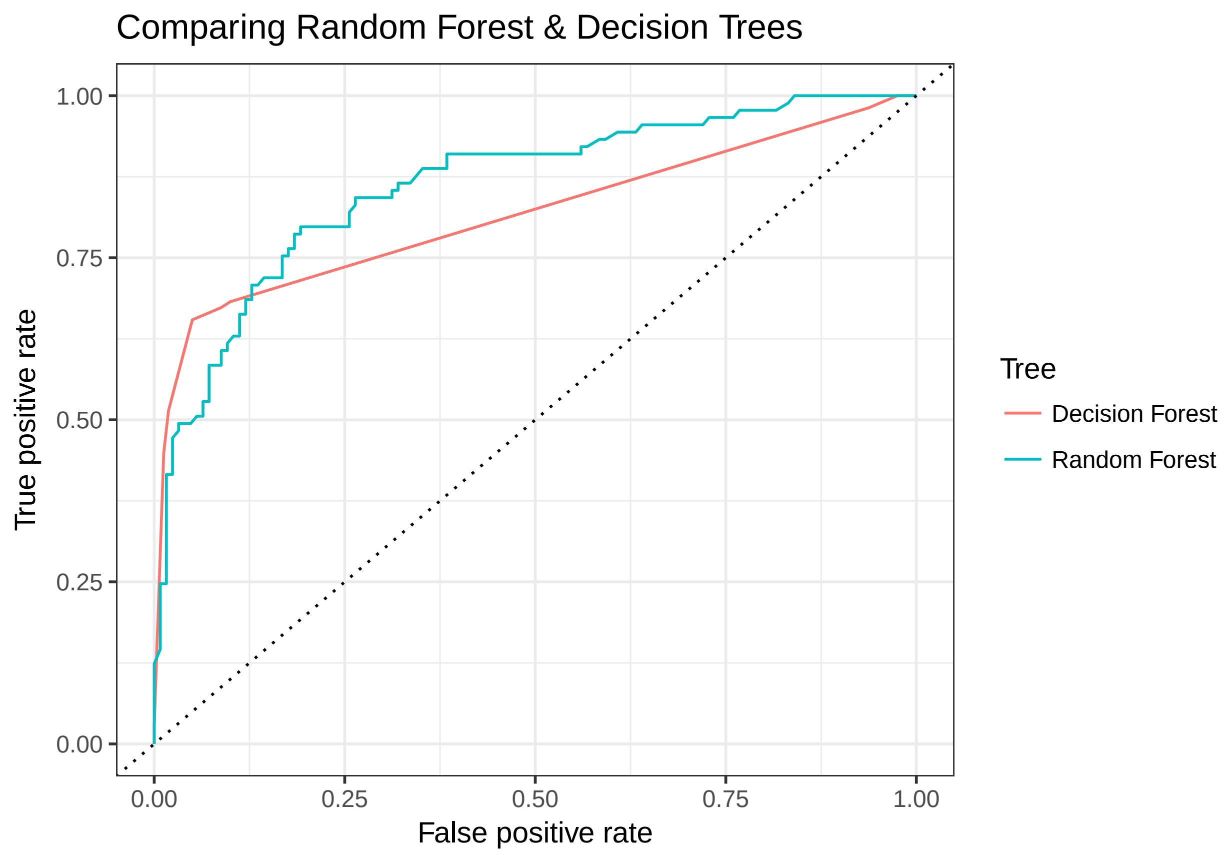
2.5.1 Answer

I did not use the function below since i did not think it was required.

```r
rf_perf_data   <- performance(rf_prediction_evaluator, measure = "tpr", x.measure = "fpr")
tree_perf_data <- performance(tree_prediction_evaluator, measure = "tpr", x.measure = "fpr")

raw_tree_perf_data <- bind_rows(
  tibble(fpr = rf_perf_data@x.values[[1]], tpr = rf_perf_data@y.values[[1]],
         Tree = "Random Forest"),
  tibble(fpr = tree_perf_data@x.values[[1]], tpr = tree_perf_data@y.values[[1]],
         Tree = "Decision Forest"))

ggplot(data = raw_tree_perf_data, aes(fpr,tpr)) +
  geom_line(aes(color = Tree)) +
  labs(title = "Comparing Random Forest & Decision Trees",
       x = 'False positive rate', y = 'True positive rate') +
  geom_abline(slope = 1, lty = 3) +
  theme_bw()
```



2.6 Q6

Answer these questions about the ROC curves: • which model performs better: decision tree or random forest? • what is the approximate false positive rate, for both the decision tree and the random forest, if we attain a true positive rate of approximately 0.75? Answers do not need to be exact - just ballpark it by looking at the plots.

2.6.1 Answer

The Random Forest model is better than the decision tree. The false positive rate @ a true positive rate for 0.75 is • Random Forest ~ 17% • Decision Tree ~ 30%