SidhuAmandeep_HW4

Amandeep Sidhu

2023-04-04

Question 1:

Build a complete pipeline with a data set of your choice and a tree-based model of your choice in R (using tidymodels) or Python (using scikit-learn). For each step, include a paragraph explaining why you did that step the way you did (what components were included and, possibly, what you decided not to do).

```
library(tidymodels)
library(BART)
```

Step 1: a brief description of where the data came from

```
#read in the data set
df_wdbc <- read.csv("data.csv")
```

The data set being used is called the 'Breast Cancer Wisconsin (Diagnostic) Data Set' and was obtained at the following link: https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data?resource=download. The important features involved is the diagnosis attribute (which says whether a patient has a malignant or benign cancer is present). The remaining attributes are simply measurements about the imaging that was recorded by the medical professionals. The measurements were computed from an imagine of a fine needle aspirate (FNA) of a breast mass sampled. The class distribution overall is 357 benign and 212 malignant cancer. This data set was first used by professionals in San Jose to linearly program breast cancer diagnosis and still to this day is used for machine learning algorithm testing.

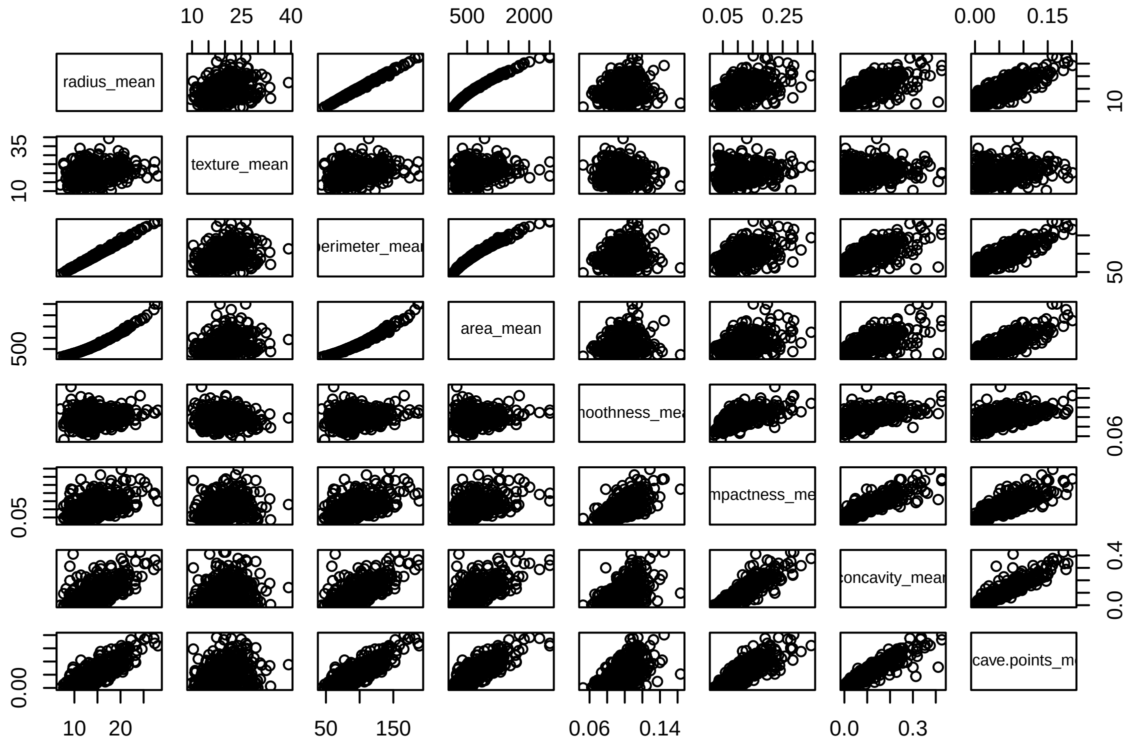
Step 2: some initial investigation of the data [which textual or graphical summaries did you investigate? Did you find anything unusual?]

Let us run some preliminary investigation on the data.

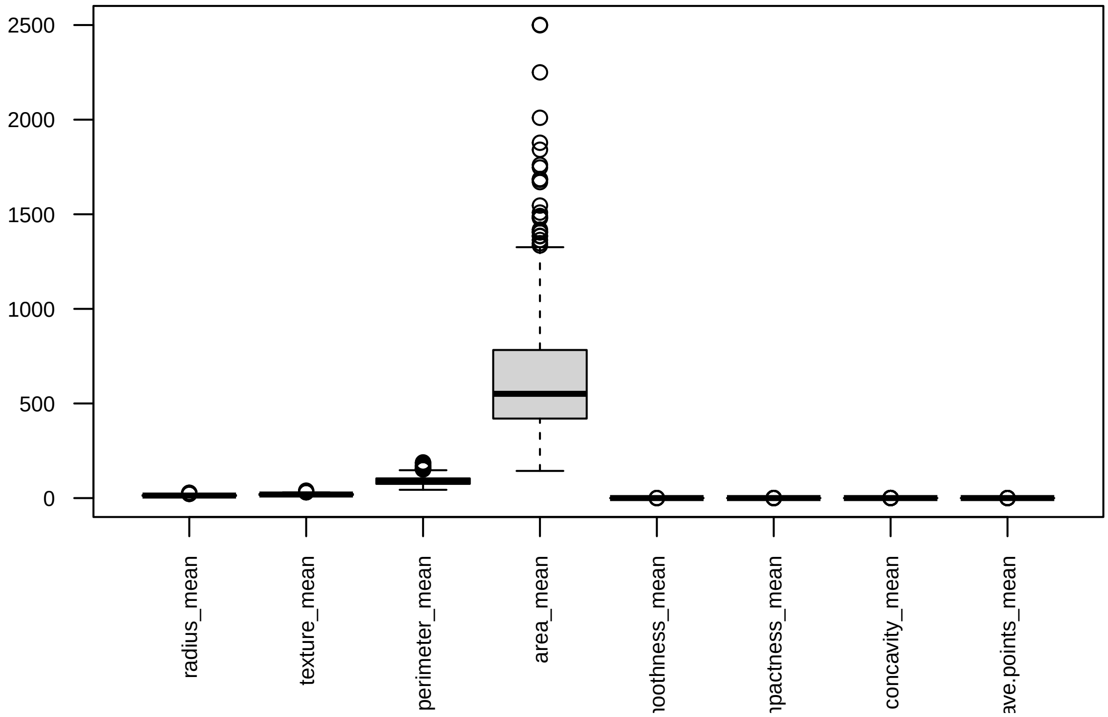
```
#for sake of space I will only choose the first few values to display
summary(df_wdbc[,c(3:10)])
```

```
##   radius_mean      texture_mean   perimeter_mean     area_mean
##  Min.   : 6.981   Min.   : 9.71   Min.   : 43.79   Min.   : 143.5
##  1st Qu.:11.700   1st Qu.:16.17   1st Qu.: 75.17   1st Qu.: 420.3
##  Median :13.370   Median :18.84   Median : 86.24   Median : 551.1
##  Mean   :14.127   Mean   :19.29   Mean   : 91.97   Mean   : 654.9
##  3rd Qu.:15.780   3rd Qu.:21.80   3rd Qu.:104.10   3rd Qu.: 782.7
##  Max.   :28.110   Max.   :39.28   Max.   :188.50   Max.   :2501.0
##  smoothness_mean   compactness_mean  concavity_mean    concave.points_mean
##   Min.   :0.05263   Min.   :0.01938   Min.   :0.00000   Min.   :0.00000
##   1st Qu.:0.08637   1st Qu.:0.06492   1st Qu.:0.02956   1st Qu.:0.02031
##   Median :0.09587   Median :0.09263   Median :0.06154   Median :0.03350
##   Mean   :0.09636   Mean   :0.10434   Mean   :0.08880   Mean   :0.04892
##   3rd Qu.:0.10530   3rd Qu.:0.13040   3rd Qu.:0.13070   3rd Qu.:0.07400
##   Max.   :0.16340   Max.   :0.34540   Max.   :0.42680   Max.   :0.20120
```

```
pairs(df_wdbc[,c(3:10)])
```



```
boxplot(df_wdbc[,c(3:10)], cex.axis = 0.7 , las  = 2)
```

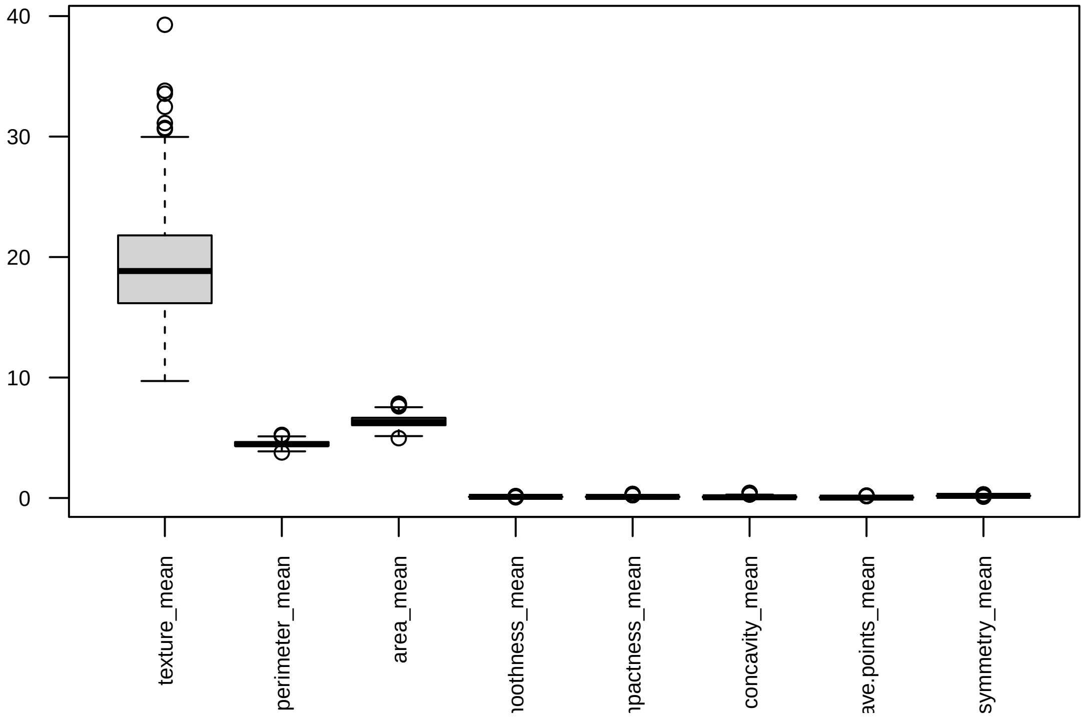


Here, all I am doing is using some summaries to observe the numerical value as well as the trend in feature variables. For the sake of space, I did not display all the features but it is fairly easy to change the parameters of each plot to observe the trends in the respective variables. According the the pairsplot, there appears to be random scatter among the variables present, but there seems to be a strong linear relationship between the radius_mean and perimeter_mean variables. Moreover, if we observe the boxplot, there is a value that has much larger values than the others (area_mean). If we observe the summary() function output, this agrees with this observation. As a result, we may need to either scale or drop this variable entirely as it may skew the data all together. If we do scale the data, it will simply just be taking the log of the entire column.

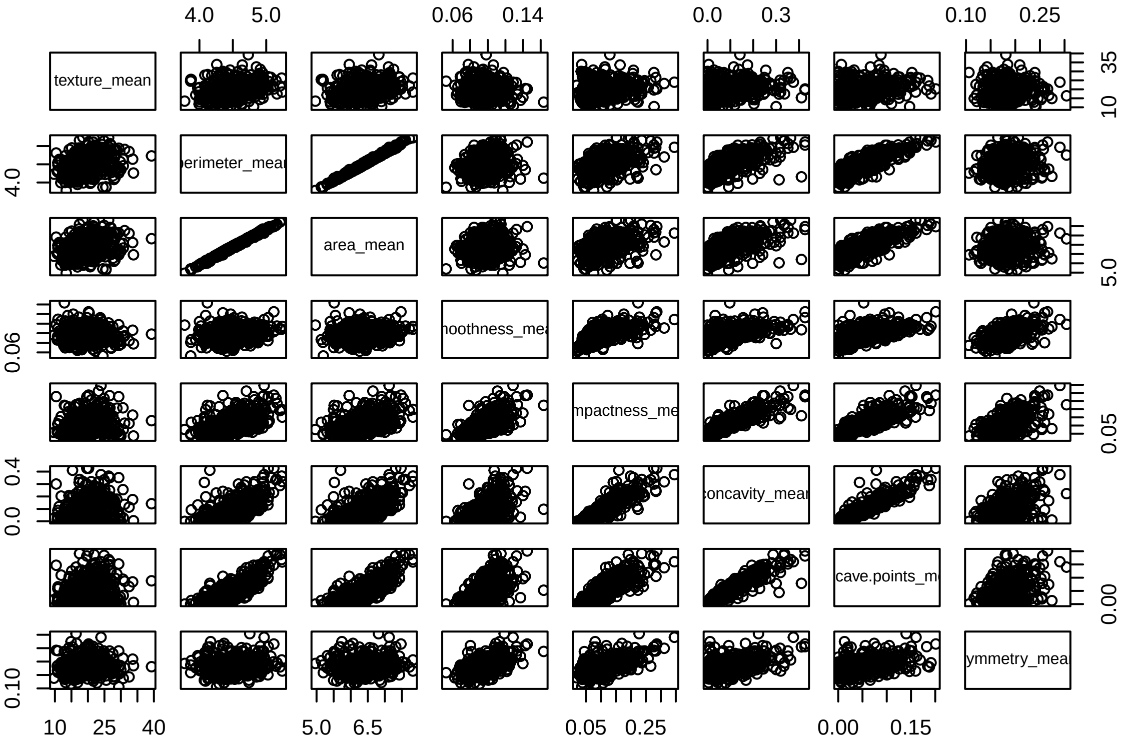
Step 3: preprocessing step(s) [scaling, feature engineering/variable selection {based on predictors only}, lumping or dropping categories from predictors, one-hot encoding, etc.]

```
#let us scale the values far apart from eachother

df_scaled <- df_wdbc %>%
  mutate(area_mean = log(area_mean), area_worst = log(area_worst),
         perimeter_mean = log(perimeter_mean),
         perimeter_worst = log(perimeter_worst))

#change the binary values

df_new <- df_scaled %>%
  mutate(diagnosis, diagnosis = if_else(diagnosis == "M", 1, 0))

#drop unecessary columns
```

```
df_new <- df_new %>%
  select(2:32)

boxplot(df_new[,c(3:10)], cex.axis = 0.7 , las  = 2)
```



```
pairs(df_new[,c(3:10)])
```



Next, I will create the recipe to be used in the model.

```
#convert to factor
df_new[, 'diagnosis'] <- as.factor(df_new[, 'diagnosis'])
#First split the model using dplyr for pipelines

set.seed(400076920)
split <-
  df_new %>%
  na.omit %>%
  initial_split()
train_data <- training(split)
test_data <- testing(split)




#recipe cooking
df_rec <- recipe(diagnosis ~., data = train_data)
```

Later on when we tune the parameters, we will see we need the diagnosis to be a factor. All that was done in the above step was scale certain features/variables by taking the log to get values closer to the other variables or else the magnitude would have been much higher in those parameters. Notice now how much tighter the variables are for the boxplot (the same variables were displayed as in step 2). The pairsplot seems to have gotten better linear relationship with most of the variables displayed in step 2 but there appears to be somewhat of a curve relationship for the first 3 variables. Overall, the scaling implemented here seems to have a positive effect on the data. Thus, we will keep this change. I have also encoded the malignant cancer types as 1 and the benign as 0 as this will help the classification problem later (i.e. here we are dealing with the binary response variable). We also dropped the patient ID column as this has no effect on the model for classification purposes.

Step 4: model choice [What model classes did you pick? Why?]

The model choice I have selected is BART (Bayesian Additive Regression Trees). Recall from lecture that this method is a sum-of-trees model to try and approximate an unknown function. We have selected this option because it is a totally acceptable algorithm for classification and regression problems (in this case we are using it for a classification problem as we are trying to classify patients into malignant and benign cancer types). Moreover, it is deemed often as more robust to outliers and missing values in the data set. This is very useful in our scenario just to provide a safety net to missing potential outliers in the data set even though we have scaled variables. The reason I have selected this over CART (classification and regression trees) as it can provide a more flexible approach to the problem (just in case the model is not linear). The computational cost is slightly more in BART than CART but that is the price of being more flexible.

Step 5: model tuning [What hyperparameters did you tune? How? What loss function did you use and why? What was the range of achieved/minimized loss functions?]

The hyperparameters I will be tuning in this BART model include: number of trees, prior_terminal_node_coef (this sets the precision of this normal distribution, which controls the amount of shrinkage applied to the terminal node coefficients) and prior_terminal_node_exp. These were obtained by reading the R documentation found on https://www.rdocumentation.org/packages/parsnip/versions/1.0.2. I will do this by using the built in tune() function and using the 'roc_auc' metric as the rsq and rmse are only reserved for regression problems in the tune_grid() library to determine performance. The value that is deemed the best will be the values of number of trees and tree depth that I will implement. Next, the loss function that I will use is the logistic distribution as the response variable is binary by nature (i.e. only takes 0 and 1 values). As outline here: https://peterroelants.github.io/posts/cross-entropy-logistic/, logistic distribution is a powerful tool for binary classification.

In the next step, I will tune the parameters as discussed above and find the best model (i.e. will state the optimal tree values and the range of the loss function - the probabilities range).

Step 6: determining and fitting the best model

```r
#first let us tune with grid search

#implement the bart model here that is to be tuned
df_temp <-
  parsnip::bart(
    trees = tune(),
    prior_terminal_node_coef = tune(),
    prior_terminal_node_expo = tune(),
  ) %>%
  set_engine("dbarts") %>%
  set_mode("classification") #as our problem is a classification problem,
#use regression if you want regression

#Next, let us obtain the parameter objects
params <-
  workflow() %>%
  add_model(df_temp) %>%
  add_recipe(df_rec) %>%
  extract_parameter_set_dials() %>%
  finalize(train_data)

#Here is where we will be actually tuning the parameters based on a random grid
```

```
set.seed(400076920) #initialize
num_of_folds <- vfold_cv(train_data)

#tune the actual model on the grid, create a workflow and then essentially
#use the folds created above to score and the one with the best score is deemed
#the best
reg_tune_grid <-
  workflow() %>%
  add_recipe(df_rec) %>%
  add_model(df_temp) %>%
  tune_grid(
    num_of_folds,
    grid = 5,
    param_info = params,
    metrics = metric_set(roc_auc, sens, spec) #if reg, use rsq or rmse
  )

#Now, select the bext hyperparameters after all that stuff of tuning
optimal_params <-
  #I used roc_auc to save some time, could use both prob would take
  #double the time to run
  select_best(reg_tune_grid, metric = 'roc_auc') %>%
  select(-.config)
#the optimal params were ntrees = 306, prior_terminal_node_coef = 0.826,
#prior_terminal_node_expo = 2.99

#estimate and get the score after using the final split!
#this may take some time to compute
final_flow <-
  workflow() %>%
  add_model(df_temp) %>%
  add_recipe(df_rec) %>%
  finalize_workflow(optimal_params)
set.seed(400076920)#initialize
fit <- final_flow %>%
  last_fit(split)

#metrics- remember we only care about the classification ones and not the
#rsq and rmse reserved for regression in the tuning documentation
collect_metrics(fit)
```

```
## # A tibble: 2 x 4
##   .metric  .estimator .estimate .config
##   <chr>    <chr>          <dbl> <chr>
## 1 accuracy binary         0.972 Preprocessor1_Model1
## 2 roc_auc  binary         0.998 Preprocessor1_Model1
```

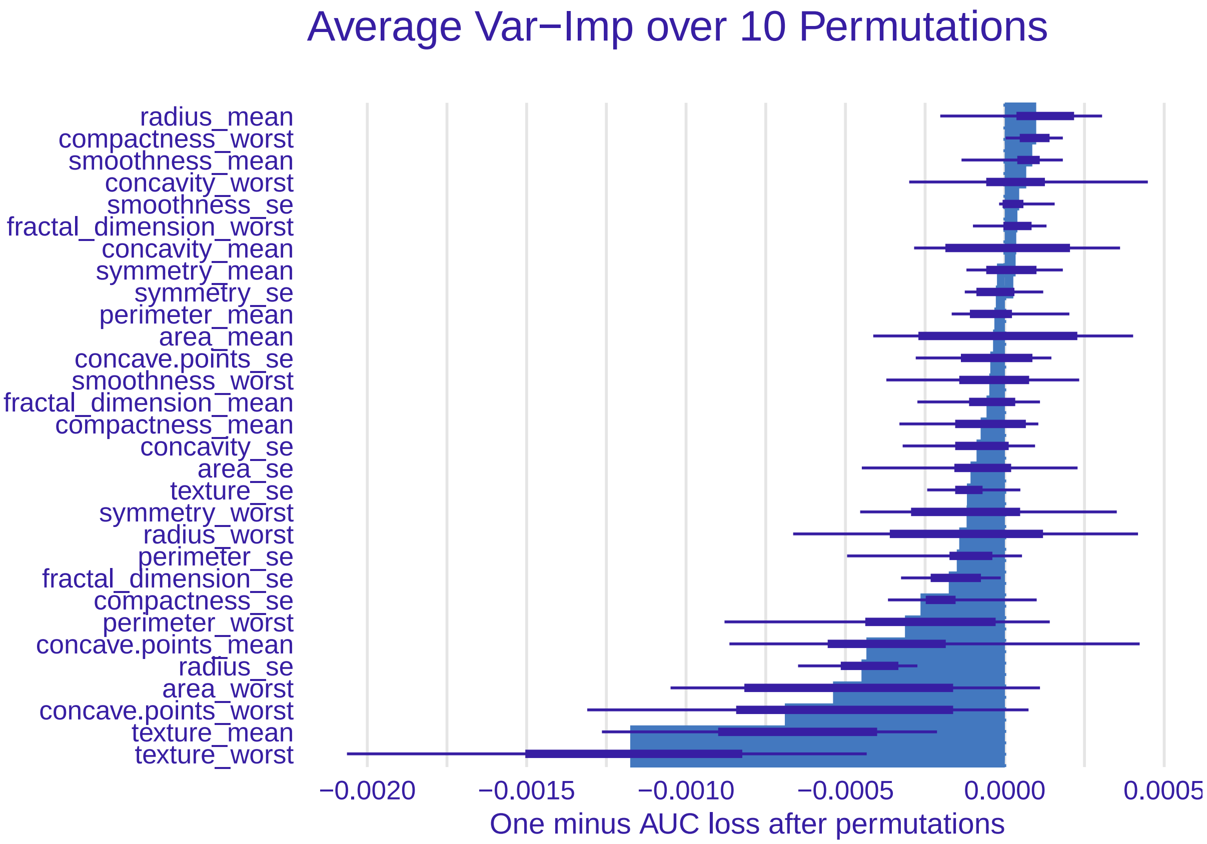
As we can see, the ‘roc_auc’ is 0.9984486 and the accuracy is 0.9720280, which is great performance.

Step 7: evaluate and explain the results of the model [partial dependence plots, variable importance, etc.]

As previously mentioned, the roc_auc and accuracy scores are very close to 1 which indicates our performance is fairly high (ideal is 1 for both metrics!). These metrics were used as only regression models for rsq and rmse are used. In our case, we have a classification problem and that is why we use these metrics. Next, let us observe the variable importance according the the optimal tuned parameters. In order to determine how much each variable explains the model, we will use the Dalex package and use similar techniques outlined here: https://rdrr.io/cran/DALEXtra/man/explain_tidymodels.html and here: https://www.r-bloggers.com/2022/12/the-effect-of-childhood-education-on-wealth-modeling-with-bayesian-additive-regression-trees-bart/.

```
#to the TA grading, if you try to run this section it may take some time
#as it needs to permuate through all those options

#First, let us define a workflow to fit a bart model given our optimal parameters
library(DALEXtra)

#this will be useful for the training of the data set and determining importance
temp_imp_data <-
  df_rec %>%
  prep() %>%
  bake(new_data = NULL)

#next, we are going to use the optimal params and use it to create a model as
#follows
optimal_specs<-
  parsnip::bart(
    trees = 306,
    prior_terminal_node_coef = 0.826,
    prior_terminal_node_expo = 2.99,
  ) %>%
  set_engine("dbarts") %>%
  set_mode("classification")

#now, as discussed in lecture we are going to use the dalex package to create
#a explainer object that essentially explains the importance of the vars
explainer <-
  explain_tidymodels(
    optimal_specs %>%
      fit(diagnosis~., data = train_data),
    data = train_data %>% select(-diagnosis),
    y = as.numeric(train_data$diagnosis),
    #need to run it as numeric here or wont work
    verbose = FALSE
  )

#the final step is calculating the variable-importance measure and using this to
#plot the graph and discuss the differences

set.seed(400076920)
imp_plot_df <-
  model_parts(
    explainer = explainer,
    B = 10, #can change this to whatever number of permutations you want
    type = "difference",
    label = ""
  )
```

```
plot(imp_plot_df) +
  ggtitle("Average Var-Imp over 10 Permutations", "")
```



To interpret the model above, we observe that one minus the auc loss after permutations is essentially the deviation away from the ideal model. It is important to keep in mind that while there are some negative values, look at the magnitude. They are relatively very small so even removing the variables might have little to no effect on them. As a result, we can declare that almost all (an argument can be made to remove some of the variables at the bottom to improve performance) are relatively important. However, no one variable stands alone in providing a positive importance to the model. Thus, we keep all.

Question 2:

(a) Derive the optimal weight values for both the MSE (L2 norm) and binomial deviance loss functions.

Derive weights for the MSE

First, let us start with the given information from ESL II, In other words, recall that to find the optimal constants γ_{jm} in a specific region can be obtained using the following:

$$\hat{\gamma_{jm}} = argmin_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

.

The above equation can be found on page 357 denoted by (10.30). For squared error loss, (i.e. the mean square error in this first part of the question), the regression tree is selected as the best one that predicts the current residuals $y_i - f_{m-1}(x_i)$, whereby the $\hat{\gamma_{jm}}$ essentially acts as the mean of those calculated residuals in each region (or leaf in our case).

Derivation:

Recall that the objective for the MSE is to minimize the sum of squared errors between the predicted and true values. During the boosting process, at each step the next step is given by the sum of the prediction from the previous boosting step (i.e. f_{m-1}). Thus, let us first start with the loss function of MSE given in ESL of:

$$L = \frac{1}{n} \sum_{i=0} (y_i - f_{m-1}(x_i))^2$$

Here, y_i acts as the observed value and γ_i acts as the predicted value.

Next, as we are required to minimize the sum of squared errors, we must take the derivative with respect to γ.

Basically, we want to fit a tree at each step 'm' of the boosting algorithm. In that senze, we take the $-\partial L/\partial f(x)$ to obtain our value of γ_{jm} where j is the index of the leaf and m is the m boosting step.

So, let us start with the result of step 2c of the algorithm 10.3.

$$\hat{\gamma_{jm}} = argmin_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

Sub in the loss function for MSE.

$$= argmin_{\gamma_{jm}} \sum_{x_i \in R_{jm}} (y_i - (f_{m-1}(x_i) + \gamma_{jm}))^2$$

Next, expand the sum of squared.

$$= argmin_{\gamma_{jm}} \sum_{x_i \in R_{jm}} (y_i^2 - 2y_i(f_{m-1}(x_i) + \gamma_{jm}) + (f_{m-1}(x_i) + \gamma_{jm})^2)$$

differentiating with respecting γ and simplifying,

$$= \sum_{x_i \in R_{jm}} (y_i - f_{m-1}(x_i) - \gamma_{jm})$$

$$= \sum_{x_i \in R_{jm}} (y_i - f_{m-1}(x_i) - \gamma_{jm}) = 0$$

$$= \sum_{x_i \in R_{jm}} y_i - \sum_{x_i \in R_{jm}} f_{m-1}(x_i) - \sum_{x_i \in R_{jm}} \gamma_{jm} = 0$$

$$\gamma_{jm} \sum_{x_i \in R_{jm}} 1 = \sum_{x_i \in R_{jm}} y_i - \sum_{x_i \in R_{jm}} f_{m-1}(x_i)$$

But note that the term $\sum_{x_i \in R_{jm}} 1$ is just the magnitude of jm (i.e. $|jm|$).

Thus, the optimal weights for the MSE γ_{jm} is

$$\gamma_{jm} = \frac{1}{|jm|} \sum_{x_i \in R_{jm}} y_i - f_{m-1}(x_i)$$

Derive weights for binomial deviance loss function

Using a similar approach as above, we can derive the weights with a binomial deviance loss function.

Recall the binomial deviance loss function (as a form of the log-loss family) in an equivalent form to ESL pg. 346 as:

$$l(y, p(x)) = ylog(p) + (1-y)log(1-p)$$

Now, we sub this loss function into the optimal weights equation previously defined.

$$\hat{\gamma_{jm}} = argmin_{\gamma_{jm}} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \gamma_{jm})$$

$$= -argmin_{\gamma_{jm}} \sum_{x_i \in R_{jm}} y_i log(p + (1-y_i)log(1-p)$$

$$= -argmin_{\gamma_{jm}} \sum_{x_i \in R_{jm}} y_i log(p + log(1-p) - y_i log(1-p)$$

$$-argmin_{\gamma_{jm}} \sum_{x_i \in R_{jm}} y_i log(\frac{p}{1-p}) - log(1-p)$$

But, remember that the terms inside the summation are simply,

$$log(1-p) = log(1 - \frac{e^{f_{m-1}(x_i)+\gamma_{jm}}}{1+e^{f_{m-1}(x_i)+\gamma_{jm}}}) = -log(1 + e^{f_{m-1}(x_i)+\gamma_{jm}})$$

$$log(\frac{p}{1-p}) = log(p) - log(1-p) = log(e^{f_{m-1}(x_i)+\gamma_{jm}}) = f_{m-1}(x_i) + \gamma_{jm}$$

Now, if we substitute this back into the argmin summation above,

$$= \frac{\partial}{\partial \gamma_{jm}} \sum_{x_i \in R_{jm}} -y_i(f_{m-1}(x_i) + \gamma_{jm}) + (-log(1 + e^{f_{m-1}(x_i))+\gamma_{jm}})$$

if we take the derivative with respect to γ_{jm} for each leaf (i.e. L_i) inside this summation, we get

$$= \frac{\partial}{\partial \gamma_{jm}} (-y_i \gamma_{jm} + \frac{e^{f_{m-1}(x_i)+\gamma_{jm}}}{1+e^{f_{m-1}(x_i)+\gamma_{jm}}})$$

Simplifying the term and setting it to 0:

$$0 = -y_i + \frac{e^{f_{m-1}(x_i)+\gamma_{jm}}}{1+e^{f_{m-1}(x_i)+\gamma_{jm}}}$$

Solving for γ_{jm} :

$$y_i = \frac{e^{f_{m-1}(x_i)+\gamma_{jm}}}{1+e^{f_{m-1}(x_i)+\gamma_{jm}}}$$

$$y_i(1 + e^{f_{m-1}(x_i)+\gamma_{jm}}) = e^{f_{m-1}(x_i)+\gamma_{jm}}$$

$$y_i + y_i(e^{f_{m-1}(x_i)+\gamma_{jm}}) = e^{f_{m-1}(x_i)+\gamma_{jm}}$$

$$y_i = e^{f_{m-1}(x_i)+\gamma_{jm}} - y_i(e^{f_{m-1}(x_i)+\gamma_{jm}})$$
$$y_i = e^{f_{m-1}(x_i)+\gamma_{jm}}(1 - y_i)$$

Taking the log,

$$log(\frac{y}{1-y_i}) = f_{m-1}(x_i) + \gamma_{jm}$$
$$\gamma_{jm} = p - f_{m-1}(x_i)$$

So, we interpret this result as the difference between p and the predicted probability of the model using the binomial deviance loss function **per** leaf (as we solved for each value in the summation).

The only sources I used for this problem was ESL II to obtain the loss functions and the papers mentioned by the assignment.

(b) Do the same for Newton boosting (Chen and Guestrin 2016), where we use a second-order rather than a first-order approximation to the loss function.

In the (Chen and Guestrin 2016) paper, we observe that the newtons boosting optimal weights can be obtained using second order approximations!

Derive the Newton boosting weight assuming a second-order approximation for the MSE Loss

As suggested by (Chen and Guestrin(2016)) : https://www.youtube.com/watch?v=StWY5QWMXCw, we can use a second order taylor approximation to get the newtons boosting estimate.

Basically, we need to use second-order approximations of the two loss functions to get the optimal weights for newton's boosting.

Recall the loss function for the MSE,

$$L = \frac{1}{n}\sum_{i=0}(y_i - f_{m-1}(x_i))^2$$

A second order approximation would be,

$$L \approx L(y_i, f_{m-1}(x_i)) + L'(y, f_{m-1}(x_i))\gamma + 1/2 * L''(y_i, f_{m-1}(x_i))\gamma^2$$

Taking the derivative with respect to γ

$$\frac{\partial L}{\partial \gamma} \approx L'(y, f_{m-1}(x_i)) + L''(y_i, f_{m-1}(x_i))\gamma$$

Solving for γ,

$$\gamma = \frac{L'(y, f_{m-1}(x_i))}{L''(y_i, f_{m-1}(x_i))}$$

Plugging in the appropriate values,

$$\gamma = \frac{-L'(y, f_{m-1}(x_i))}{L''(y_i, f_{m-1}(x_i))}$$

Notice, that the second derivative of the MSE loss function is simply a constant. Thus, the optimal value is

$$\gamma_{jm} = \frac{1}{|jm|}\sum_{i=0}(y_i - f_{m-1}(x_i))^2$$

.

which is simply the average of residuals at each leaf!

Derive the Newton Boosting Weight using binomial loss function

So, let us find a second order taylor approximation of the likelihood.

Using the equivalent form of 10.18 on page 346 of ESL II, we observe that the likelihood is:

$$L = y_i log(p) + (1 - y_i) log(1 - p)$$

$$L \approx L(y_i, p) + L'(y, p)\gamma + 1/2 * L''(y_i, p)\gamma^2$$

Ignoring the first term, we take the derivative with respect to γ.

$$\frac{\partial L}{\partial \gamma} \approx L'(y, p) + L''(y_i, p)\gamma$$

Solving for γ,

$$\gamma = -\frac{L'(y, p)}{L''(y_i, p)}$$

$$\gamma = -\frac{\frac{y_i - p}{p(1-p)}}{\frac{p^2(x_i) - 2y_i p + y_i}{p-1)^2 p^2}}$$

Simplifying this further, we arrive at the conclusion of:

$$\gamma_{jm} = \frac{p - y_i}{p(1 - p)}$$

which is essentially the residual (derived in 2a!) divided by p multiplied by (1-P).