ML_Long_Form

Eleanor Colligan

3/16/2022

Table of Contents:

Introduction

```
library(tidyverse)
library(tibble)
library(dials)
library(skimr)
library(workflows)
library(dplyr)
library(tidytext)
library(rsample)
library(recipes)
library(naniar)
library(parsnip)
library(tune)
#set seed
set.seed(384209)
```

As a woman, I find woman's health very interesting. Furthermore, due to the fact we live in a historically (and currently) man-focused world, there is a huge gap in women's health research and medicine compared to men's health. One example of this is birth control, and how reliable forms of birth control rest on women to take and deal with troubling (and) at times very serious) side effects. Furthermore, cancer is an illness that virtually touches everyone; it seems that everyone I know has a family member or friend that has been diagnosed with cancer. Breast cancer, and this data set as a result, which is at the intersection of woman's health and cancer, combines both pressing issues. The data set I chose is from the UCI Machine Learning Respository, and

can be found at: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29. During class, I feel like we did a lot more practice with regression than with classification, or perhaps classification is just less intuitive to me given I've learned about regression in other statistics classes. Either way, I wanted to do my project on classification so I could understand it better. I tried four algorithms: K-Nearest-Neighbor, Logistic Regression, Random Forest, and Boosted Tree.

Research Question How accurately can I predict the diagnosis (Malignant or Benign) of a tumor based off of various measurements of that tumor?

Data Set

```
data <- read_csv('unprocessed/data.csv') %>%
  mutate(diagnosis = factor(diagnosis, levels = c("M", "B")))
```

```
#data check- making sure there's no missing variables.
data %>%
naniar::miss_var_summary()
```

```
## # A tibble: 33 x 3
##    variable         n_miss pct_miss
##    <chr>             <int>    <dbl>
##  1 ...33               568      100
##  2 id                    0        0
##  3 diagnosis             0        0
##  4 radius_mean           0        0
##  5 texture_mean          0        0
##  6 perimeter_mean        0        0
##  7 area_mean             0        0
##  8 smoothness_mean       0        0
##  9 compactness_mean      0        0
## 10 concavity_mean        0        0
## # i 23 more rows
```

Removing all the variables I don't plan on using from data_set (all variables unrelated to the mean of various measurements)

```
data<- data %>%
select(ends_with('mean') | 'diagnosis' | 'id')
```

Cleaning variables names:

```
data<- data %>%
janitor::clean_names()
```

Looking at the outcome variable:

```
data %>%
select(diagnosis)
```

```
## # A tibble: 568 x 1
##    diagnosis
##    <fct>
##  1 M
##  2 M
##  3 M
##  4 M
##  5 M
##  6 M
##  7 M
##  8 M
##  9 M
## 10 M
## # i 558 more rows
```

Everything looks good to me with the diagnosis variable. The options are either B (for benign) or M (malignant). There are no missing rows of data.

Splitting the data:

```
data_split <- data %>%
  initial_split(prop=.8, strata= diagnosis)
train <- data_split %>% training()
test <- data_split %>% testing()
```

I'm splitting it by diagnosis because diagnosis is my outcome variable, and in past labs we've always made the strata the outcome variable.

Resampling data:

```
data_resamples <-
  train %>%
  vfold_cv(v=5, repeats = 3, strata = diagnosis)

keep_pred<- control_resamples(save_pred = TRUE, save_workflow = TRUE)
```

Current plans for recipe:

```
train<- tibble(train)
recipe <- recipe(diagnosis ~ id + radius_mean +  texture_mean +  perimeter_mean + area_mean + smoothness
  step_dummy(all_nominal_predictors()) %>%
  step_normalize(all_predictors())
```

Checking the recipe:

```
recipe %>%
 prep() %>%
 bake(new_data= train)
```

```
## # A tibble: 453 x 12
##        id radius_mean texture_mean perimeter_mean area_mean smoothness_mean
##     <dbl>       <dbl>        <dbl>          <dbl>     <dbl>           <dbl>
##  1 -0.180      -0.284       -0.815         -0.247    -0.369           0.836
```

```
##  2 -0.180      -1.31        -1.57          -1.30     -1.07           0.465
##  3 -0.244      -0.298       -0.191         -0.373    -0.358         -0.451
##  4  0.464      -1.68        -0.555         -1.65     -1.28          -0.729
##  5 -0.244      -0.579       -1.06          -0.562    -0.571          0.516
##  6 -0.244      -0.662        0.541         -0.700    -0.632         -0.702
##  7 -0.244      -0.123       -0.670         -0.182    -0.221         -1.40
##  8 -0.244      -0.610       -0.233         -0.659    -0.604         -0.976
##  9  0.464      -0.731       -0.115         -0.757    -0.686         -0.0566
## 10 -0.244      -1.56        -1.72          -1.55     -1.21           0.109
## # i 443 more rows
## # i 6 more variables: compactness_mean <dbl>, concavity_mean <dbl>,
## #   concave_points_mean <dbl>, symmetry_mean <dbl>,
## #   fractal_dimension_mean <dbl>, diagnosis <fct>
```

There are ten columns, which is the correct number of predictors.

KNN MODEL

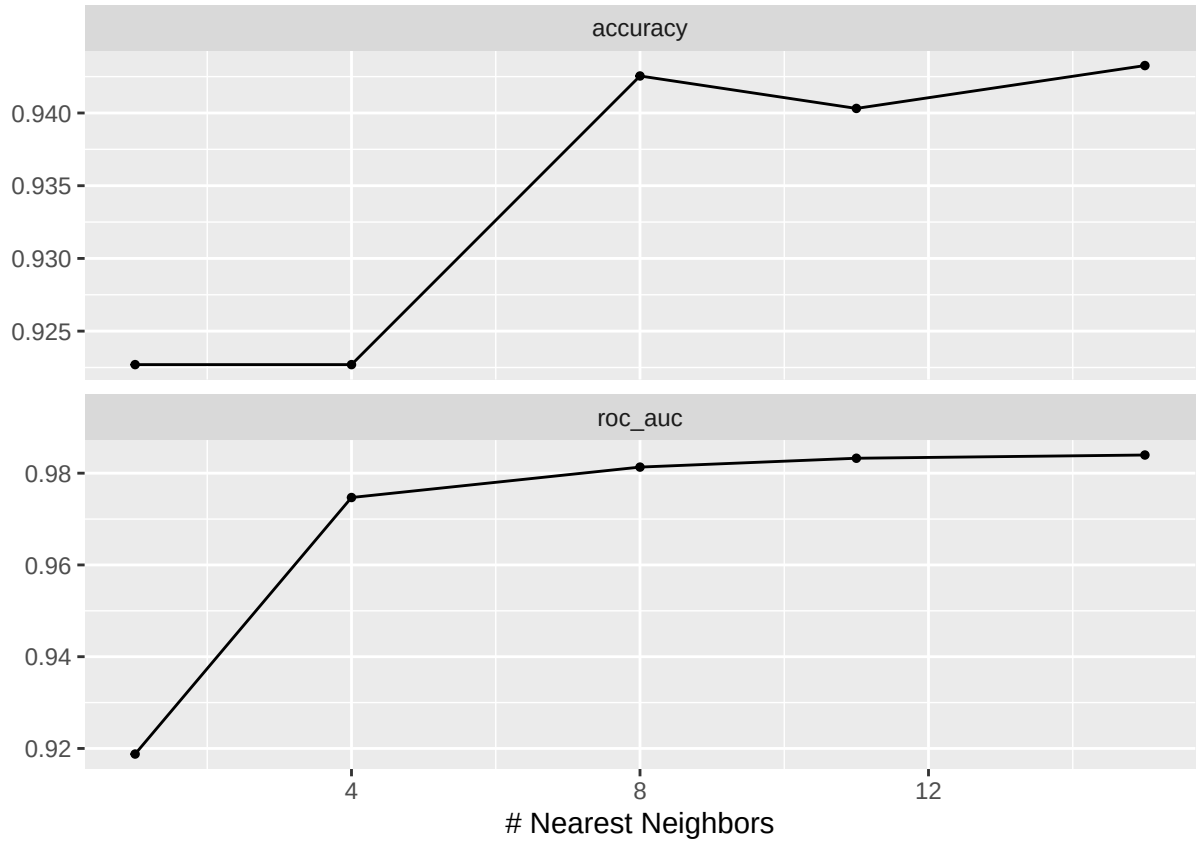
First, I'm going to try the k-nearest-neighbor model:

```
knn_model <- nearest_neighbor(mode = "classification",
                        neighbors = tune()) %>%
  set_engine("kknn")


# workflow
knn_wflow <-
  workflow() %>%
  add_recipe(recipe) %>%
  add_model(kknn_model)

#tuning params
kknn_params <- parameters(knn_wflow)
kknn_params %>%
  update(neighbors= neighbors(range= c(5,25)))
```

```
## Collection of 1 parameters for tuning
##
##  identifier      type    object
##   neighbors neighbors nparam[+]
```

```
grid_info <- grid_regular(kknn_params, levels = 5)

#fit to resamples
knn_res <- knn_wflow %>%
  tune_grid(resamples= data_resamples,
            control= keep_pred,
            grid= grid_info
  )

save(knn_res, file= 'processed/knn_res.rda')
```

```
load('processed/knn_res.rda')
knn_res %>%
  autoplot()
```




```
knn_res %>%
  collect_metrics() %>%
  filter() %>%
  arrange(mean)
```

```
## # A tibble: 10 x 7
##    neighbors .metric  .estimator  mean     n std_err .config
##        <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1         1 roc_auc  binary     0.919    15 0.00527 Preprocessor1_Model1
##  2         1 accuracy binary     0.923    15 0.00484 Preprocessor1_Model1
##  3         4 accuracy binary     0.923    15 0.00484 Preprocessor1_Model2
##  4        11 accuracy binary     0.940    15 0.00565 Preprocessor1_Model4
##  5         8 accuracy binary     0.943    15 0.00576 Preprocessor1_Model3
##  6        15 accuracy binary     0.943    15 0.00532 Preprocessor1_Model5
##  7         4 roc_auc  binary     0.975    15 0.00477 Preprocessor1_Model2
##  8         8 roc_auc  binary     0.981    15 0.00325 Preprocessor1_Model3
##  9        11 roc_auc  binary     0.983    15 0.00301 Preprocessor1_Model4
## 10        15 roc_auc  binary     0.984    15 0.00303 Preprocessor1_Model5
```

The ROC AUC from my nearest neighbors model from 4 upwards is really good, with values all above .97. It looks like the best AUC score is when the number of nearest neighbors is at 15, with a value of over .98.

My accuracy is also really good from 4 neighbors upwards, though not as good as my ROC AUC scores, with values all .92.

Finalizing KNN Model

```
#tuning params
select_best(knn_res)
```

```
## # A tibble: 1 x 2
##   neighbors .config
##       <int> <chr>
## 1        15 Preprocessor1_Model5
```

```
kn_params <- tibble(neighbors=15)

#finalmodelspec
final_wflow <-
  knn_res %>%
  extract_workflow()%>%
  finalize_workflow(parameters=kn_params)

final_model_fit<-
  final_wflow %>%
  fit(train)
```

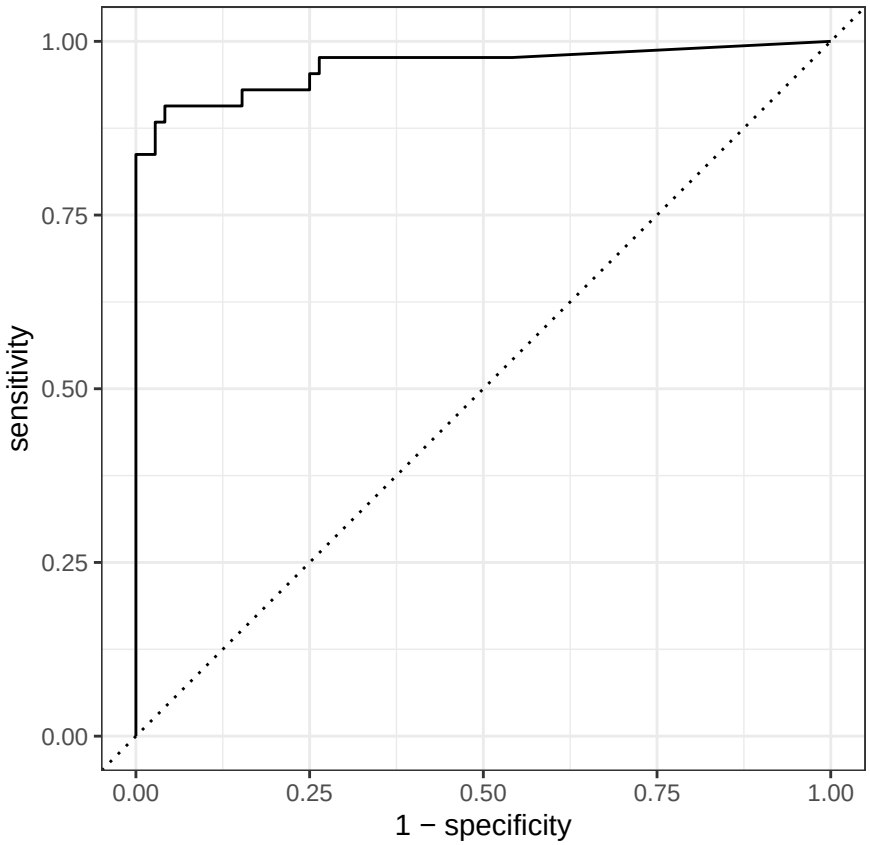
Using 15 neighbors.

Looking at Predictions & Metrics

```
data_knn_pred <- bind_cols(predict(final_model_fit, test %>% select(-diagnosis)), test %>% select(diagn

data_knn_prob_pred <- bind_cols(predict(final_model_fit, test %>% select(-diagnosis), type = "prob"), t

#accuracy
yardstick::accuracy(data_knn_pred, diagnosis, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.939
```

```
#roc auc
roc_curve_knn <- yardstick::roc_curve(data_knn_prob_pred, diagnosis, .pred_M)
autoplot(roc_curve_knn)
```




```
yardstick::roc_auc(data_knn_prob_pred, diagnosis, .pred_M)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.964
```

My KNN model has an ROC AUC of 96.4% and an accuracy 93.9%. Next, I'm gonna try using ga logistic regression model.

LOGISTIC REGRESSION MODEL

Next, I'm going to try the logistic regression model:

```
log_model <- logistic_reg() %>%
  set_engine("glm")


# workflow
log_wf <- workflow() %>%
  add_model(log_model) %>%
  add_recipe(recipe)

# this engine has no tuning parameters, as I see from it's documentation. This means I will have no tun
```

```
#fit to resamples
log_res <- log_wf %>%
  tune_grid(resamples= data_resamples,
            control= keep_pred,
            grid= grid_info
  )
```

```
## > A | warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## There were issues with some computations   A: x1There were issues with some computations   A: x5There
```

```
save(log_res, file= 'processed/log_res.rda')
```

```
load('processed/log_res.rda')
log_res %>%
  collect_metrics() %>%
  filter() %>%
  arrange(mean)
```

```
## # A tibble: 2 x 6
##   .metric  .estimator  mean     n std_err .config
##   <chr>    <chr>      <dbl> <int>   <dbl> <chr>
## 1 accuracy binary     0.932    15 0.00532 Preprocessor1_Model1
## 2 roc_auc  binary     0.984    15 0.00241 Preprocessor1_Model1
```

My accuracy and ROC AUC both look really good to me.

Since I don't have any tuning parameters, I'll use this model with my testing data for my final fit:

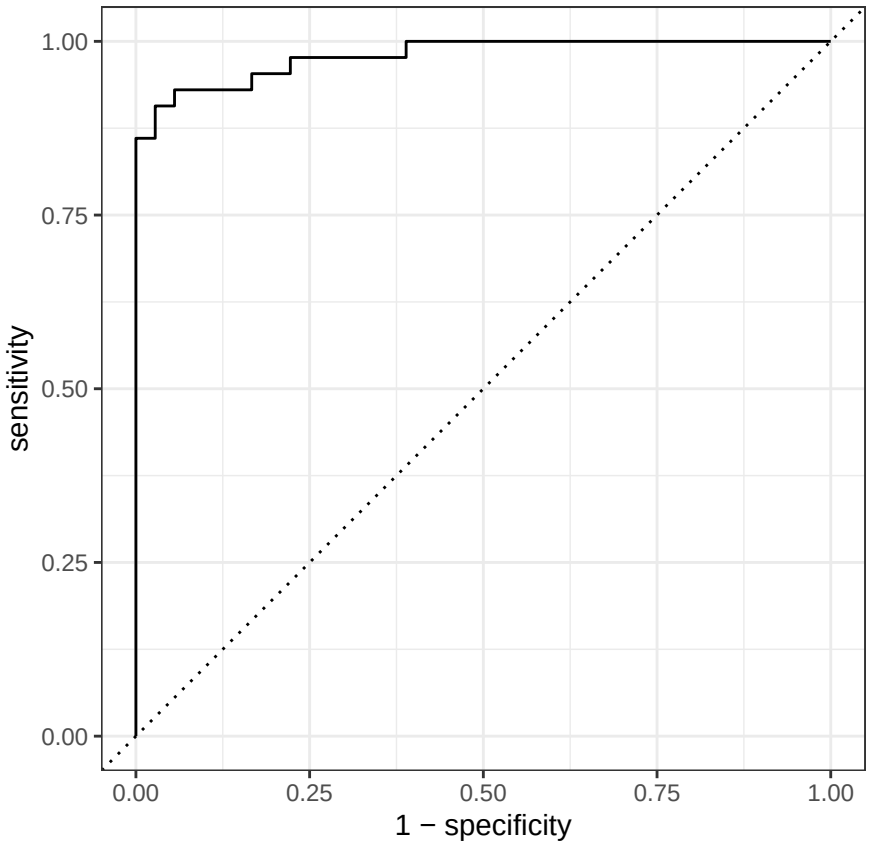
```
#finalmodelspec
final_wflow <-
  log_res %>%
  extract_workflow()

final_model_fit<-
  final_wflow %>%
  fit(train)
```

Looking at Predictions & Metrics

```
data_log_pred <- bind_cols(predict(final_model_fit, test %>% select(-diagnosis)), test %>% select(diagn

data_log_prob_pred <- bind_cols(predict(final_model_fit, test %>% select(-diagnosis), type = "prob"), t

#accuracy
yardstick::accuracy(data_log_pred, diagnosis, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.948
```

```r
#roc auc
roc_curve_log <- yardstick::roc_curve(data_log_prob_pred, diagnosis, .pred_M)
autoplot(roc_curve_log)
```




```r
yardstick::roc_auc(data_log_prob_pred, diagnosis, .pred_M)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.979
```

My accuracy has actually gone up (though only by one percent) compared to my model using the training set data, while my ROC AUC has gone down, by by under .5%.

Compared to my KNN model, my logistic model is better with regards to both accuracy and ROC AUC.

For my tree models, I'm going to use random forest and boosted.

Boosted

creating model & workflow, then turning the parameters and fitting them to the resamples.

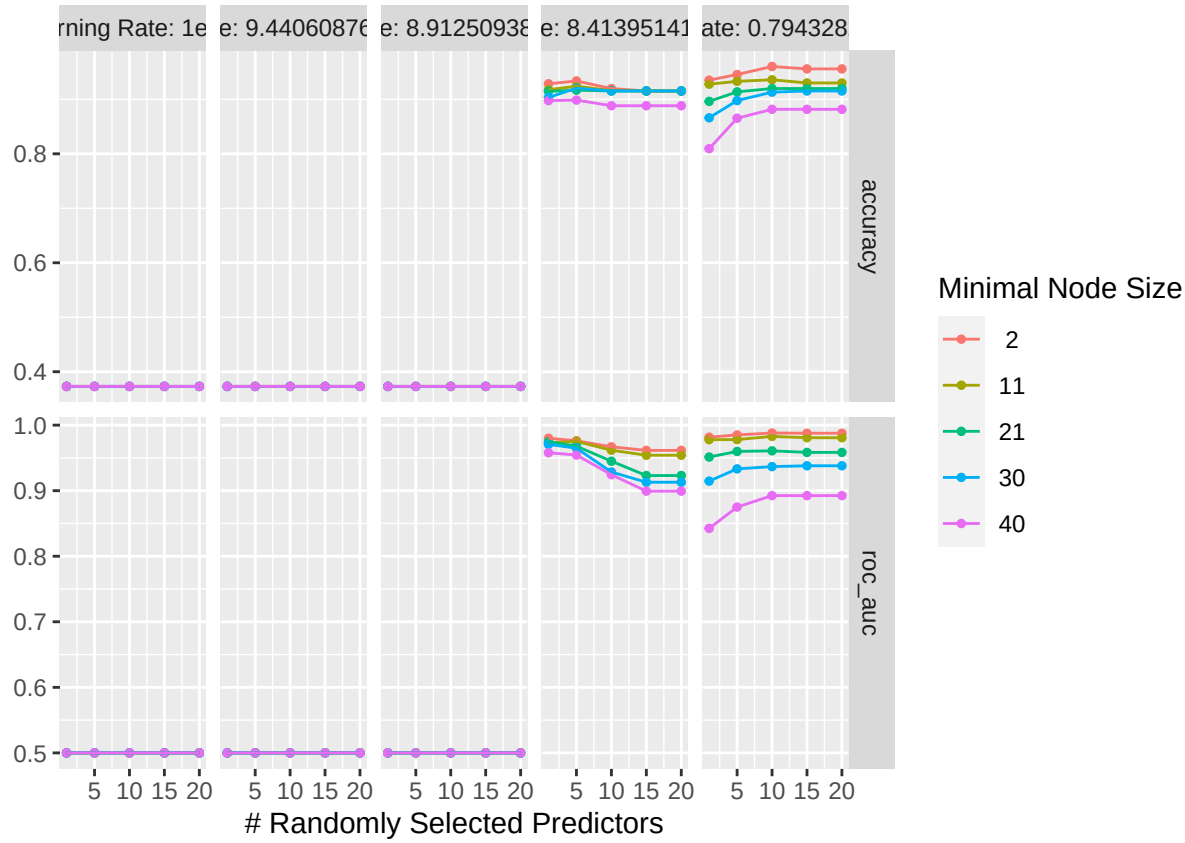
```
boost_model <- boost_tree(mode = "classification",
                          mtry = tune(),
                        min_n = tune(),
                        learn_rate= tune()) %>%
                     set_engine("xgboost")

# workflow
boost_wflow <-
  workflow() %>%
  add_recipe(recipe) %>%
  add_model(boost_model)

#tuning params
boost_params <- parameters(boost_wflow)
boost_params <- boost_params %>%
  update(learn_rate= learn_rate(range=c(-20,-.1)),
         mtry=mtry(range=c(1,20)))
grid_info <- grid_regular(boost_params, levels = 5)

#fit to resamples
boost_res <- boost_wflow %>%
  tune_grid(resamples= data_resamples,
            control= keep_pred,
            grid= grid_info)
```

saving my boost resamples and grid:

```
save(boost_res, file= 'processed/boost_res.rda')
```

looking at metrics:

```
load('processed/boost_res.rda')
boost_res %>%
  autoplot()
```




```
boost_res %>%
  collect_metrics() %>%
  filter() %>%
  arrange(mean)
```

```
## # A tibble: 250 x 9
##     mtry min_n learn_rate .metric  .estimator  mean     n  std_err .config
##    <int> <int>      <dbl> <chr>    <chr>      <dbl> <int>    <dbl> <chr>
##  1     1     2      1e-20 accuracy binary     0.373    15 0.000304 Preprocessor~
##  2     5     2      1e-20 accuracy binary     0.373    15 0.000304 Preprocessor~
##  3    10     2      1e-20 accuracy binary     0.373    15 0.000304 Preprocessor~
##  4    15     2      1e-20 accuracy binary     0.373    15 0.000304 Preprocessor~
##  5    20     2      1e-20 accuracy binary     0.373    15 0.000304 Preprocessor~
##  6     1    11      1e-20 accuracy binary     0.373    15 0.000304 Preprocessor~
##  7     5    11      1e-20 accuracy binary     0.373    15 0.000304 Preprocessor~
##  8    10    11      1e-20 accuracy binary     0.373    15 0.000304 Preprocessor~
##  9    15    11      1e-20 accuracy binary     0.373    15 0.000304 Preprocessor~
## 10    20    11      1e-20 accuracy binary     0.373    15 0.000304 Preprocessor~
## # i 240 more rows
```

Since I have so many options for my predictors, the visualization of the metrics isn't very helpful. I'm going to filter it down so only mean values of above .95 are shown.

```
boost_res %>%
  collect_metrics() %>%
  filter(mean > .95)
```

```
## # A tibble: 34 x 9
##     mtry min_n learn_rate .metric .estimator  mean     n std_err .config
##    <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     1     2 0.00000841 roc_auc binary     0.980    15 0.00360 Preprocessor1_~
##  2     5     2 0.00000841 roc_auc binary     0.976    15 0.00382 Preprocessor1_~
##  3    10     2 0.00000841 roc_auc binary     0.967    15 0.00523 Preprocessor1_~
##  4    15     2 0.00000841 roc_auc binary     0.961    15 0.00512 Preprocessor1_~
##  5    20     2 0.00000841 roc_auc binary     0.961    15 0.00512 Preprocessor1_~
##  6     1    11 0.00000841 roc_auc binary     0.973    15 0.00472 Preprocessor1_~
##  7     5    11 0.00000841 roc_auc binary     0.975    15 0.00384 Preprocessor1_~
##  8    10    11 0.00000841 roc_auc binary     0.962    15 0.00566 Preprocessor1_~
##  9    15    11 0.00000841 roc_auc binary     0.954    15 0.00692 Preprocessor1_~
## 10    20    11 0.00000841 roc_auc binary     0.954    15 0.00692 Preprocessor1_~
## # i 24 more rows
```

selecting the best boost model:

```
select_best(boost_res)
```

```
## # A tibble: 1 x 4
##    mtry min_n learn_rate .config
##   <int> <int>      <dbl> <chr>
## 1    10     2      0.794 Preprocessor1_Model103
```

Fitting Final Boosted Model

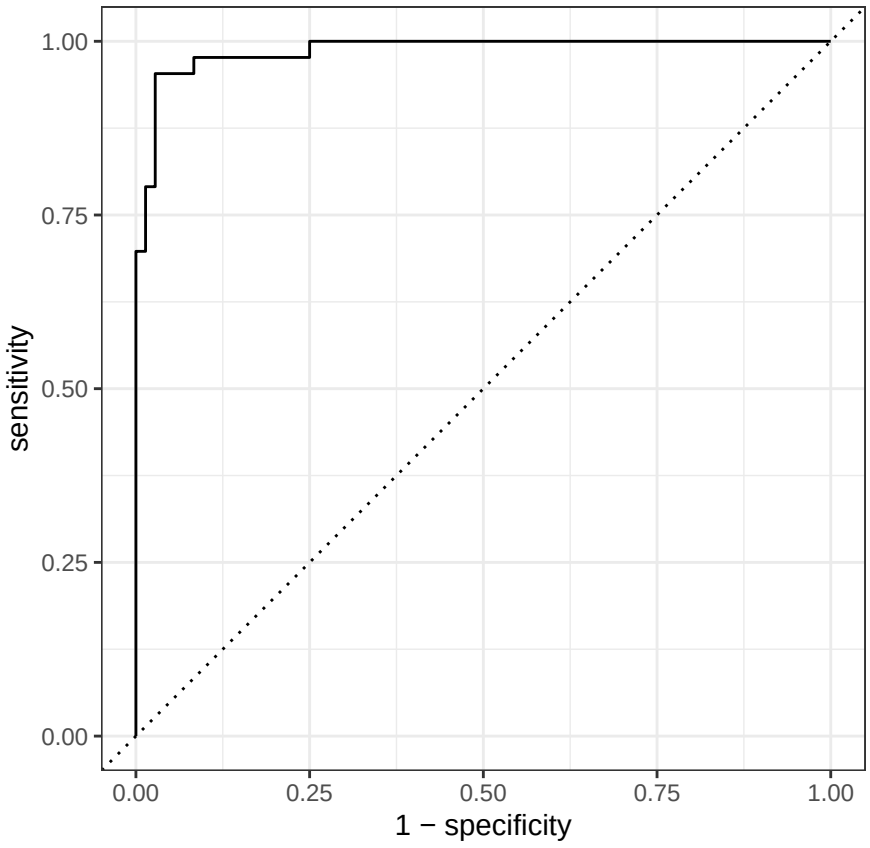
```
boost_params <- tibble(mtry=10, min_n=2, learn_rate=0.7943282)

#finalmodelspec
final_wflow <-
  boost_res %>%
  extract_workflow()%>%
  finalize_workflow(parameters=boost_params)

final_model_fit<-
  final_wflow %>%
  fit(train)
```

looking at the accuracy and roc auc of my boost model:

```
data_boost_pred <- bind_cols(predict(final_model_fit, test %>% select(-diagnosis)), test %>% select(dia

data_boost_prob_pred <- bind_cols(predict(final_model_fit, test %>% select(-diagnosis), type = "prob"),

#accuracy
yardstick::accuracy(data_boost_pred, diagnosis, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.957
```

```
#roc auc
roc_curve_boost <- yardstick::roc_curve(data_boost_prob_pred, diagnosis, .pred_M)
autoplot(roc_curve_boost)
```




```
yardstick::roc_auc(data_boost_prob_pred, diagnosis, .pred_M)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.986
```

Boosted Model Conclusion

So far, my boosted model is the most accurate and has the best ROC AUC score out of all my models.

Random Forest

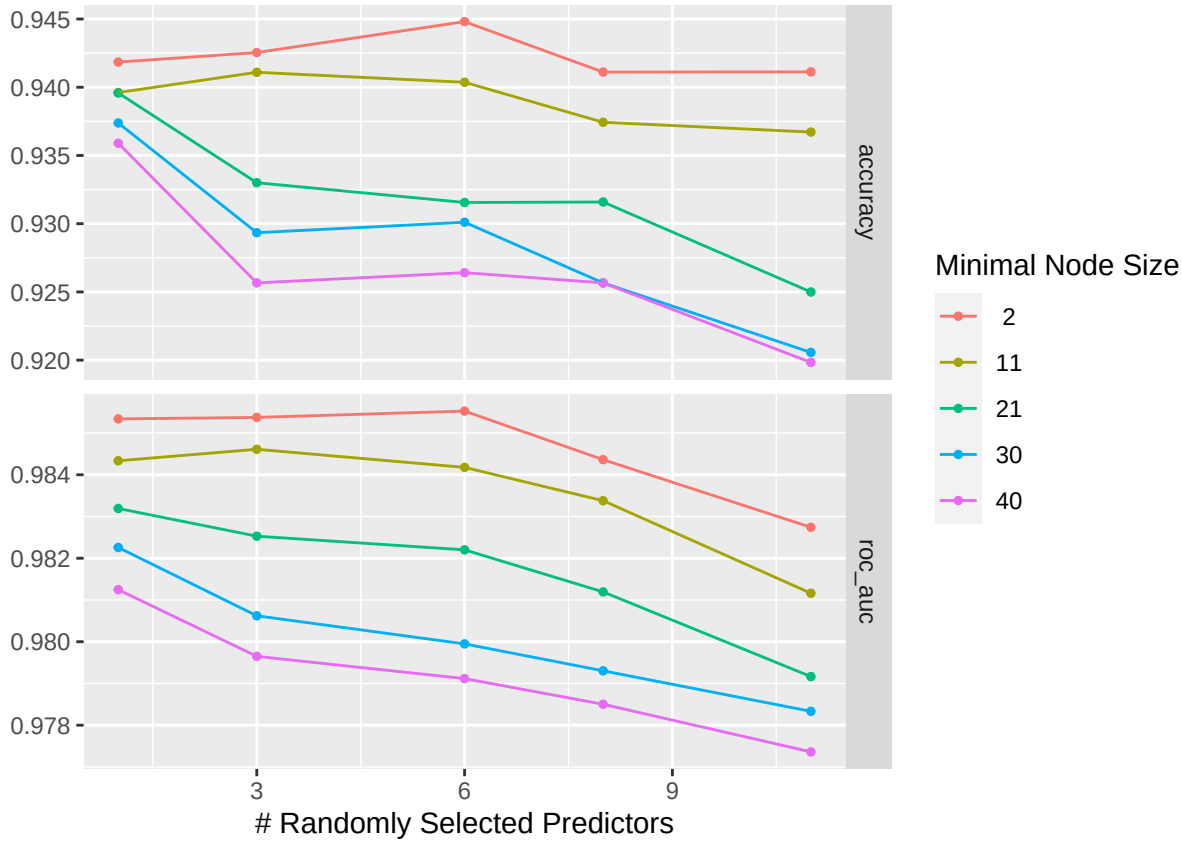
```
rf_model <- rand_forest(mode = "classification",
                        min_n = tune(),
                        mtry = tune(),
                    trees= 2000) %>%
  set_engine("ranger")

rf_wflow <- workflow() %>%
  add_recipe(recipe) %>%
  add_model(rf_model)

rf_params <- parameters(rf_wflow)

rf_params <- rf_params %>%
  update(mtry=mtry(range=c(1,11))
  )
grid_info <- grid_regular(rf_params, levels=5)

rf_res <- rf_wflow %>%
  tune_grid(resamples= data_resamples,
            control= keep_pred,
            grid= grid_info)
```

saving my random forest resamples and grid:

```
save(rf_res, file= 'processed/rf_res.rda')
```

collecting metrics:

```
load('processed/rf_res.rda')
rf_res %>%
  autoplot()
```




```
rf_res %>%
  collect_metrics() %>%
  filter() %>%
  arrange(mean)
```

```
## # A tibble: 50 x 8
##     mtry min_n .metric  .estimator  mean     n std_err .config
##    <int> <int> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1    11    40 accuracy binary     0.920    15 0.00654 Preprocessor1_Model25
##  2    11    30 accuracy binary     0.921    15 0.00719 Preprocessor1_Model20
##  3    11    21 accuracy binary     0.925    15 0.00772 Preprocessor1_Model15
##  4     8    30 accuracy binary     0.926    15 0.00685 Preprocessor1_Model19
##  5     3    40 accuracy binary     0.926    15 0.00669 Preprocessor1_Model22
##  6     8    40 accuracy binary     0.926    15 0.00685 Preprocessor1_Model24
##  7     6    40 accuracy binary     0.926    15 0.00651 Preprocessor1_Model23
##  8     3    30 accuracy binary     0.929    15 0.00654 Preprocessor1_Model17
##  9     6    30 accuracy binary     0.930    15 0.00676 Preprocessor1_Model18
## 10     6    21 accuracy binary     0.932    15 0.00674 Preprocessor1_Model13
## # i 40 more rows
```

Interestingly, it looks like for all minimal node sizes, the roc auc declines from when the randomly selected number of predictors goes below one, meaning the roc auc is best when the number of predictors we use is one. The minimum node size of 2 for both the roc auc and the accuracy is best.

```
load('processed/rf_res.rda')
select_best(rf_res)
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     6     2 Preprocessor1_Model03
```

tuning my mtry and min_n parameters:

```
#tuning params
select_best(rf_res)
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1     6     2 Preprocessor1_Model03
```

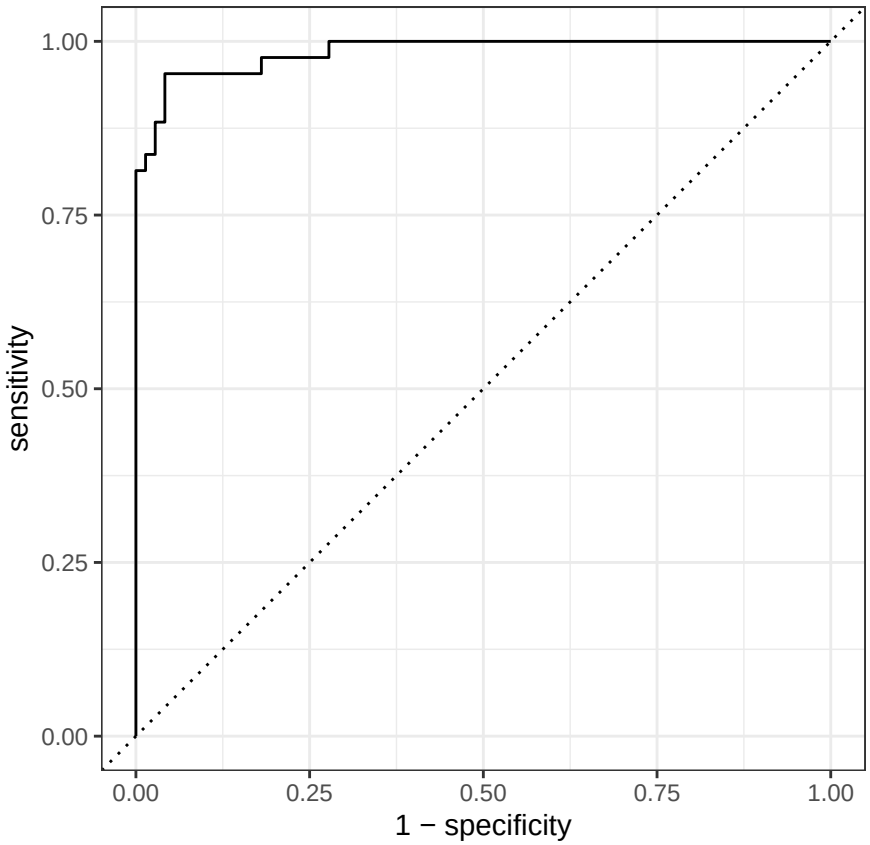
```
rf_params <- tibble(mtry=2, min_n=2)

#finalmodelspec
final_wflow <-
  rf_res %>%
  extract_workflow()%>%
  finalize_workflow(parameters=rf_params)

final_model_fit<-
  final_wflow %>%
  fit(train)
```

Looking at Predictions & Metrics

```
data_rf_pred <- bind_cols(predict(final_model_fit, test %>% select(-diagnosis)), test %>% select(diagnos

data_rf_prob_pred <- bind_cols(predict(final_model_fit, test %>% select(-diagnosis), type = "prob"), tes

#accuracy
yardstick::accuracy(data_rf_pred, diagnosis, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.948
```

```
#roc auc
roc_curve_rf <- yardstick::roc_curve(data_rf_prob_pred, diagnosis, .pred_M)
autoplot(roc_curve_rf)
```




```
yardstick::roc_auc(data_rf_prob_pred, diagnosis, .pred_M)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.985
```

My roc auc is 98.5% which is a little below that of my boosted tree model. My accuracy is 94.8%, which is also a bit below that of my boosted model.

Final Model: Boosted Forests

Since my boosted forests was the most successful out of all my models (K-Nearest Neighbors, Logistic Regression and Random Forest) I am going to try tuning it even more to see if I can get the accuracy above 95.7%.

creating workflow and fitting my parameters to my resamples:

```
final_model <- boost_tree(mode = "classification",
                          mtry = tune(),
                          min_n = 2,
                          learn_rate= tune()) %>%
                    set_engine("xgboost")
```

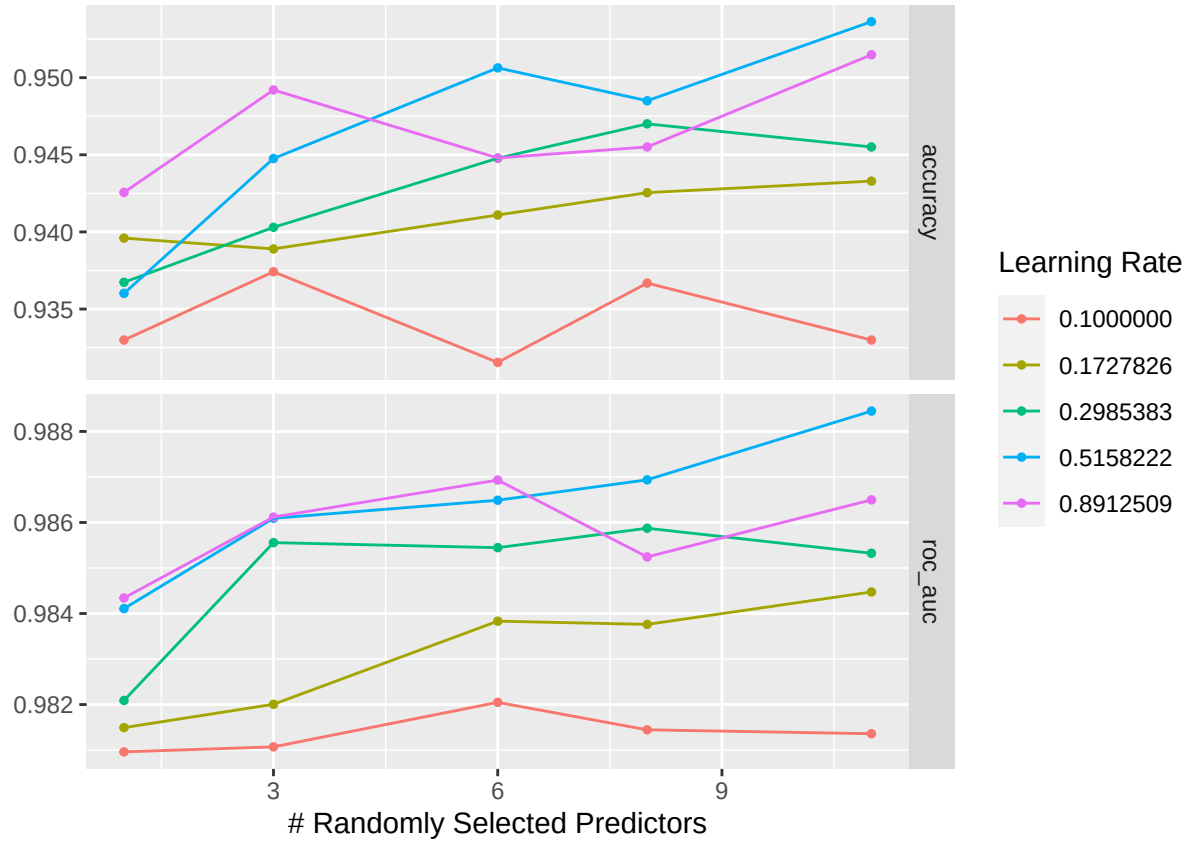
```
# workflow
final_wflow <-
  workflow() %>%
  add_recipe(recipe) %>%
  add_model(final_model)

#tuning params
final_params <- parameters(final_wflow)
final_params <- final_params %>%
  update(learn_rate= learn_rate(range=c(-1,-.05)),
         mtry=mtry(range=c(1,11)))
grid_info <- grid_regular(final_params, levels = 5)

#fit to resamples
final_res <- final_wflow %>%
  tune_grid(resamples= data_resamples,
            control= keep_pred,
            grid= grid_info)
```

saving resamples:

```
save(final_res, file='processed/final_res.rda')
```

looking at metrics:

```
load('processed/final_res.rda')
final_res %>%
  autoplot()
```




```
final_res %>%
  collect_metrics() %>%
  filter(mean > .95)
```

```
## # A tibble: 28 x 8
##     mtry learn_rate .metric .estimator  mean     n std_err .config
##    <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1     1      0.1   roc_auc binary     0.981    15 0.00339 Preprocessor1_Model01
##  2     3      0.1   roc_auc binary     0.981    15 0.00355 Preprocessor1_Model02
##  3     6      0.1   roc_auc binary     0.982    15 0.00396 Preprocessor1_Model03
##  4     8      0.1   roc_auc binary     0.981    15 0.00352 Preprocessor1_Model04
##  5    11      0.1   roc_auc binary     0.981    15 0.00392 Preprocessor1_Model05
##  6     1      0.173 roc_auc binary     0.981    15 0.00357 Preprocessor1_Model06
##  7     3      0.173 roc_auc binary     0.982    15 0.00368 Preprocessor1_Model07
##  8     6      0.173 roc_auc binary     0.984    15 0.00334 Preprocessor1_Model08
##  9     8      0.173 roc_auc binary     0.984    15 0.00353 Preprocessor1_Model09
## 10    11      0.173 roc_auc binary     0.984    15 0.00328 Preprocessor1_Model10
## # i 18 more rows
```

It looks like there isn't any accuracy above .95, but maybe with the testing set it will prove to be greater. selecting the best:

```
select_best(final_res)
```

```
## # A tibble: 1 x 3
```

```
##    mtry learn_rate .config
##   <int>      <dbl> <chr>
## 1    11      0.516 Preprocessor1_Model20
```

fitting final parameters and model:

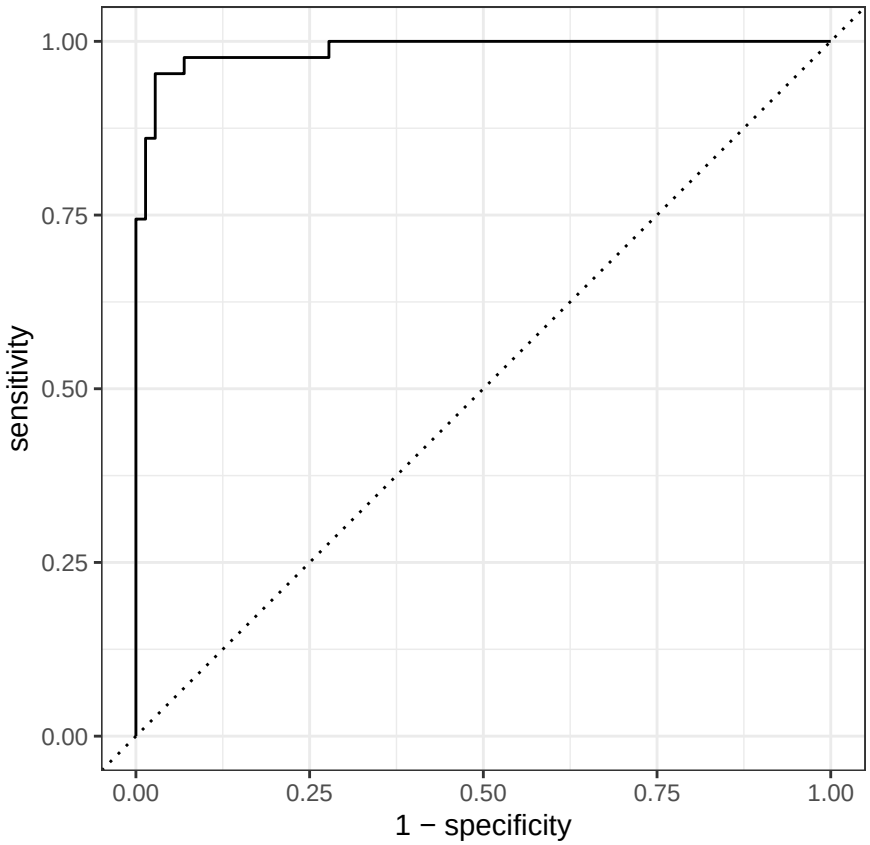
```
final_params <- tibble(mtry=11, min_n=2, learn_rate= 0.5158222)

#finalmodelspec
final_wflow <-
  final_res %>%
  extract_workflow()%>%
  finalize_workflow(parameters=final_params)

final_model_fit<-
  final_wflow %>%
  fit(train)
```

looking at the accuracy and roc auc of my final model:

```
data_final_pred <- bind_cols(predict(final_model_fit, test %>% select(-diagnosis)), test %>% select(dia

data_final_prob_pred <- bind_cols(predict(final_model_fit, test %>% select(-diagnosis), type = "prob"),

#accuracy
yardstick::accuracy(data_final_pred, diagnosis, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.965
```

```
#roc auc
roc_curve_final <- yardstick::roc_curve(data_final_prob_pred, diagnosis, .pred_M)
autoplot(roc_curve_final)
```




```
yardstick::roc_auc(data_final_prob_pred, diagnosis, .pred_M)
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.988
```

I know from testing different values for mtry that the accuracy value stays the same for mtry=11 and values above 11. This makes sense because the number of predictors I have in my model is 11. For my final model, I got an ROC AUC score of 98.77% and an accuracy score of 96.5%.

Conclusion

For my machine learning project, I decided to a classification project instead of regression, because I have taken classes on linear regression before but never on classification. My research question was: *How accurately can I predict the diagnosis (Malignant or Benign) of a tumor based off of various measurements of that tumor?* To answer that question, I used 4 different types of algorithims to mode: K-Nearest-Neighbor, Logistic Regression, Random Forest, and Boosted Tree. After creating models using all four, I found that my model using the Boosted Tree algorithim was the most accurate, with an accuracy of 94.7%. My random forest also had an accuracy of 94.7%, but my boosted trees had a higher ROC AUC. I wanted to try tuning the parameters more, and seeing if I could get the accuracy even higher. After tuning the parameters of my boosted trees model more, I ended up with an ROC AUC score of 98.77% and an accuracy score of 96.5%. So, to answer my research question, I can accurately predict the diagnosis of a tumor based off of various measurements of that tumor at an accuracy of up to 96.5%. A further research question I have is how each

individual variable impacts the accuracy of my model; however, I'm unsure how to determine that at this time, but would like to learn.

Data Issues

This data, which was donated to the UCI Machine Learning Repository in 1995, is a bit outdated. I think it would be more beneficial to have more measurement variables from patients from the 2000s onwards. In addition to this, I noticed that there is no variable indicator for if the patient had cancer previously, which does impact the likelihood of going into remission for cancer. Though it is not related to tumor dimensions, I think this would have a big impact on the accuracy, and potentionally bring it even closer to 100%. This would definitely be a good variable to include, which could be dummy coded as 0 (no, has not previously been diagnosed with cancer) or 1 (yes, has been previously diagnosed with cancer.)