Random Forests

Andrew Noecker

4/27/2021

Introduction

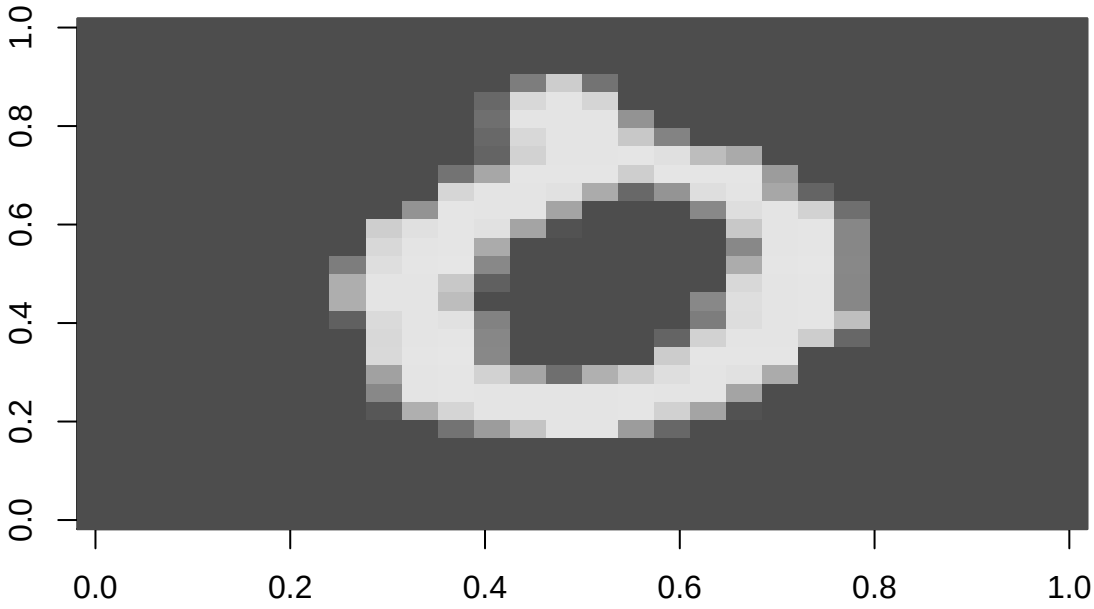
Today we will be using a small subset of the MNIST dataset, by selecting 1000 digits for both training and testing:

```
mnist <- read_mnist("~/Mscs 341 S22/Class/Data")
set.seed(2022)
index <- sample(nrow(mnist$train$images), 1000)
digit.train.tbl <- as_tibble (mnist$train$images[index,]) %>%
  mutate(digit = factor(mnist$train$labels[index]))

index <- sample(nrow(mnist$test$images), 1000)
digit.test.tbl <- as_tibble (mnist$test$images[index,]) %>%
  mutate(digit = factor(mnist$test$labels[index]))
```

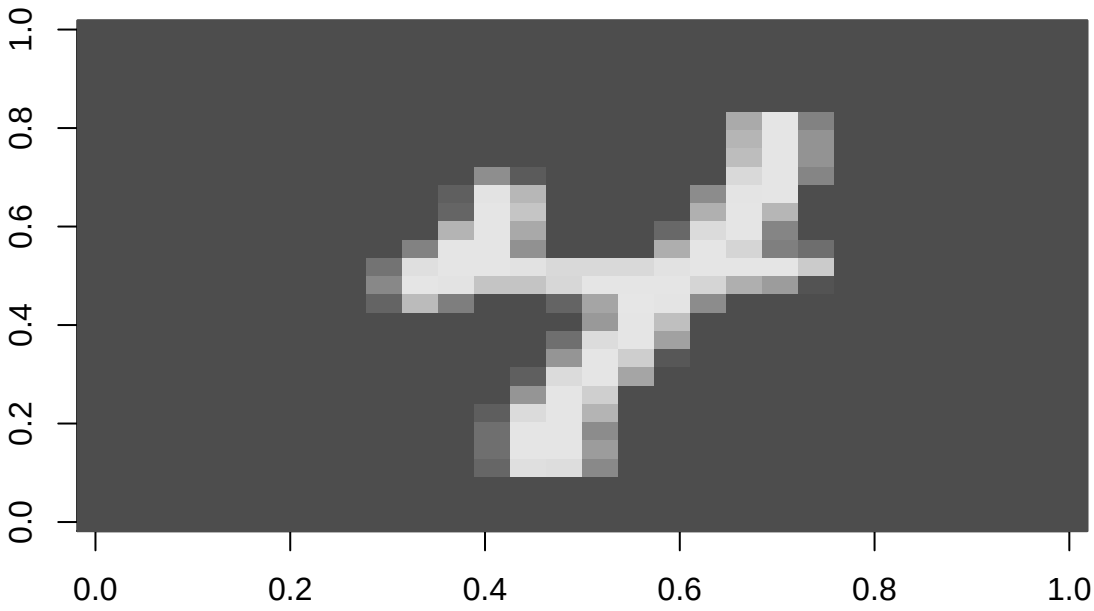
And as before let's make a couple of functions to plot particular digits from our dataset

```
plotImage <- function(dat,size=28){
  imag <- matrix(dat,nrow=size)[,28:1]
  image(imag,col=grey.colors(256), xlab = "", ylab="")
}

plot_row <- function(tbl) {
  ntbl <- tbl %>%
    select(V1:V784)
  plotImage(as.matrix(ntbl))
}
```

```
# Plot the 100th digit from training
plot_row(digit.train.tbl[100,])
```




```
# Plot the 13th digit from testing
plot_row(digit.test.tbl[12,])
```



As in our last homework, we are interested in doing multi-digit classification.

Maximal classification trees

1. a. Fill out the details of the function `create_rtree` which creates a maximal tree for doing multi-digit classification. The function receives a training dataset and returns a fitted regression tree. Create a regression tree called `digit.rtree.model` using the function `create_rtree` on your training dataset.

```
create_rtree <- function (train.tbl) {
  rtree.model <- decision_tree(cost_complexity = 0)%>%
    set_mode("classification")%>%
    set_engine("rpart")
  rtree.recipe <- recipe(digit ~ ., data = train.tbl)

  model.wflow <- workflow()%>%
    add_model(rtree.model)%>%
    add_recipe(rtree.recipe)

  fit(model.wflow, train.tbl)
}

digit.rtree.model <- create_rtree(digit.train.tbl)
```

   b. Calculate the accuracy and confusion matrix of `digit.rtree.model` using your testing dataset

```
augment(digit.rtree.model, digit.test.tbl)%>%
  accuracy(digit, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.663
```

```
augment(digit.rtree.model, digit.test.tbl)%>%
  conf_mat(digit, .pred_class)
```

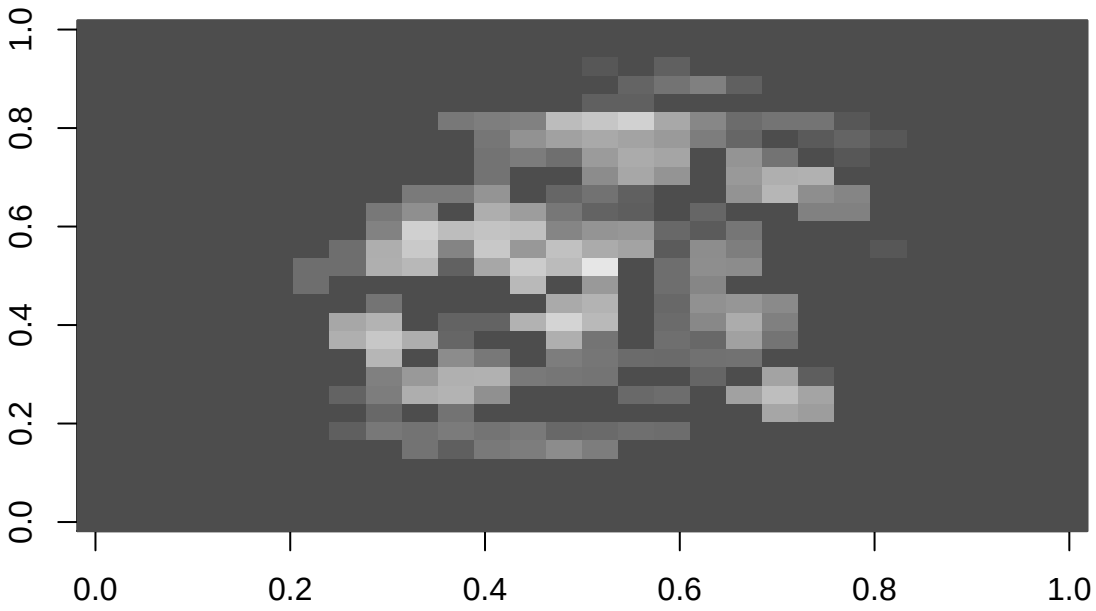
```
##            Truth
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0  73   0   3   6   4   6   4   1   6   1
##          1   0 114   3   2   0   0   0   3   3   0
##          2   5   5  70   8   7   1  14   5   7   2
##          3   0   2   2  47   1   6   0   0   2   1
##          4   4   0   1   0  42   1   2   2   9  11
##          5   2   1   1  22  11  58   3   1   5  19
##          6   2   1   7   2  18   7  62   4   9   2
##          7   3   0   5   2   1   2   4  77   0   1
##          8   0   1   1   4   5   5  10   0  49   5
##          9   0   0   5   1  13   0   1   9   2  71
```

We can define a function that will allow us to plot the pixel importance for a particular model.

```
create_image_vip <- function(model.fit) {
  # Creates the importance image
  imp.tbl <- model.fit %>%
    extract_fit_engine() %>%
    vip::vi() %>%
    mutate(col=as.double(str_remove(Variable,"V")))
  mat <- rep(0, 28*28)
  mat[imp.tbl$col] <- imp.tbl$Importance
  mat
}
```

Let's use this function to show the important pixels for our model:

```
create_image_vip(digit.rtree.model) %>%
  plotImage()
```



Using bootstrapping

Bootstrapping allows us to create new training datasets by sampling with replacement from our training dataset. Let's create 3 bootstrap samples from our original training dataset:

```
set.seed(12345)
bootstrap.split <- bootstraps(digit.train.tbl, times=3)
bootstrap.split
```

```
## # Bootstrap sampling
## # A tibble: 3 x 2
##   splits             id
##   <list>             <chr>
## 1 <split [1000/346]> Bootstrap1
## 2 <split [1000/372]> Bootstrap2
## 3 <split [1000/371]> Bootstrap3
```

Notice how `bootstrap.split` is a tibble. Also note that we can access the 2nd bootstrap by using the command:

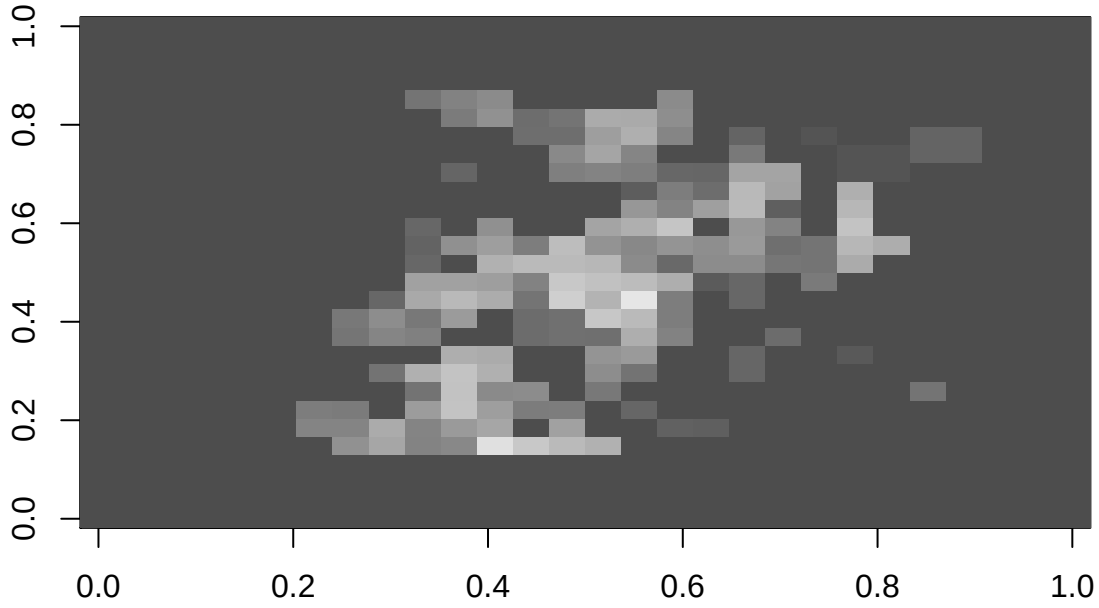
```
analysis(bootstrap.split$splits[[2]])
```

```
## # A tibble: 1,000 x 785
##       V1    V2    V3    V4    V5    V6    V7    V8    V9   V10   V11   V12   V13
##    <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
##  1     0     0     0     0     0     0     0     0     0     0     0     0     0
```

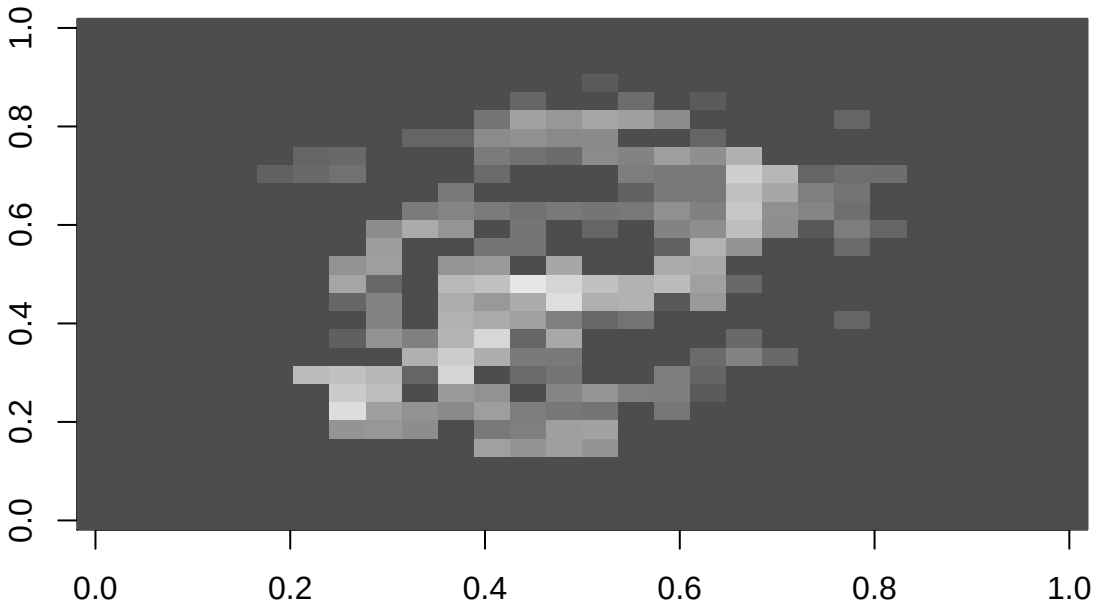
```
##  2     0     0     0     0     0     0     0     0     0     0     0     0     0
##  3     0     0     0     0     0     0     0     0     0     0     0     0     0
##  4     0     0     0     0     0     0     0     0     0     0     0     0     0
##  5     0     0     0     0     0     0     0     0     0     0     0     0     0
##  6     0     0     0     0     0     0     0     0     0     0     0     0     0
##  7     0     0     0     0     0     0     0     0     0     0     0     0     0
##  8     0     0     0     0     0     0     0     0     0     0     0     0     0
##  9     0     0     0     0     0     0     0     0     0     0     0     0     0
## 10     0     0     0     0     0     0     0     0     0     0     0     0     0
## # ... with 990 more rows, and 772 more variables: V14 <int>, V15 <int>,
## #   V16 <int>, V17 <int>, V18 <int>, V19 <int>, V20 <int>, V21 <int>,
## #   V22 <int>, V23 <int>, V24 <int>, V25 <int>, V26 <int>, V27 <int>,
## #   V28 <int>, V29 <int>, V30 <int>, V31 <int>, V32 <int>, V33 <int>,
## #   V34 <int>, V35 <int>, V36 <int>, V37 <int>, V38 <int>, V39 <int>,
## #   V40 <int>, V41 <int>, V42 <int>, V43 <int>, V44 <int>, V45 <int>,
## #   V46 <int>, V47 <int>, V48 <int>, V49 <int>, V50 <int>, V51 <int>, ...
```

2. Fit classification trees to each bootstrapped training dataset and plot the variable importance as an image for each model. Are there pixels that are common to the three models?

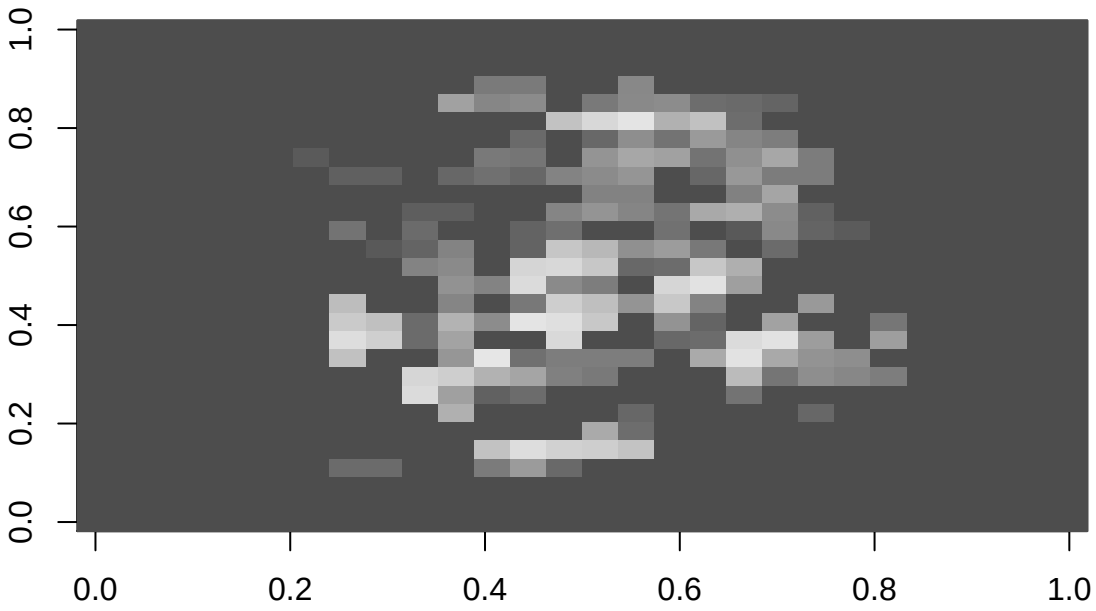
```
digit.rtree.model1 <- create_rtree(analysis(bootstrap.split$splits[[1]]))
digit.rtree.model2 <- create_rtree(analysis(bootstrap.split$splits[[2]]))
digit.rtree.model3 <- create_rtree(analysis(bootstrap.split$splits[[3]]))

create_image_vip(digit.rtree.model1) %>%
  plotImage()
```



```
create_image_vip(digit.rtree.model2) %>%
  plotImage()
```




```
create_image_vip(digit.rtree.model3) %>%
  plotImage()
```



Overall, we don't see a major difference between the 3 images of pixel importance. For the first bootstrap model, we see the important pixels are a little more slanted from bottom left to top right. The second model is more centered in the middle, but the most important (lighter) pixels also seem to go from bottom left to top right. The third one is also much more centered, with the lighter pixels are a little more all over the center with less of a direction.

Remembering bagging

In bagging we combine a fixed amount of trees which are trained on bootstraps of our training dataset. Let's create a bagging model by leveraging the function `use_ranger` which generates code that we can copy/paste to create our model.

```
library(usemodels)
use_ranger(digit~., data=digit.train.tbl, tune=FALSE)
```

```
## ranger_recipe <-
##   recipe(formula = digit ~ ., data = digit.train.tbl)
##
## ranger_spec <-
##   rand_forest(trees = 1000) %>%
##   set_mode("classification") %>%
##   set_engine("ranger")
##
## ranger_workflow <-
##   workflow() %>%
```

```
##   add_recipe(ranger_recipe) %>%
##   add_model(ranger_spec)
```

We will modify the generated code as follows:

- We will be using `trees=100` so that our code runs faster
- We will be using `mtry=784` so that make sure that we are using all of our variables for each of our classification trees.
- We will add the parameter `importance="impurity"`, so that we can determine the importance of the variables in our model.

```
ranger_recipe <-
  recipe(formula = digit ~ ., data = digit.train.tbl)

ranger_spec <-
  rand_forest(trees = 100, mtry=784) %>%
  set_mode("classification") %>%
  set_engine("ranger", importance = "impurity")

ranger_workflow <-
  workflow() %>%
  add_recipe(ranger_recipe) %>%
  add_model(ranger_spec)
```

Finally let's fit our model on the training dataset and calculate our accuracy on the testing dataset. Notice how our bagging model improves significantly over individual maximal classification trees

```
digit.bag.model <- fit(ranger_workflow, digit.train.tbl)

augment(digit.bag.model, digit.test.tbl) %>%
  accuracy(truth=digit, estimate= .pred_class)
```

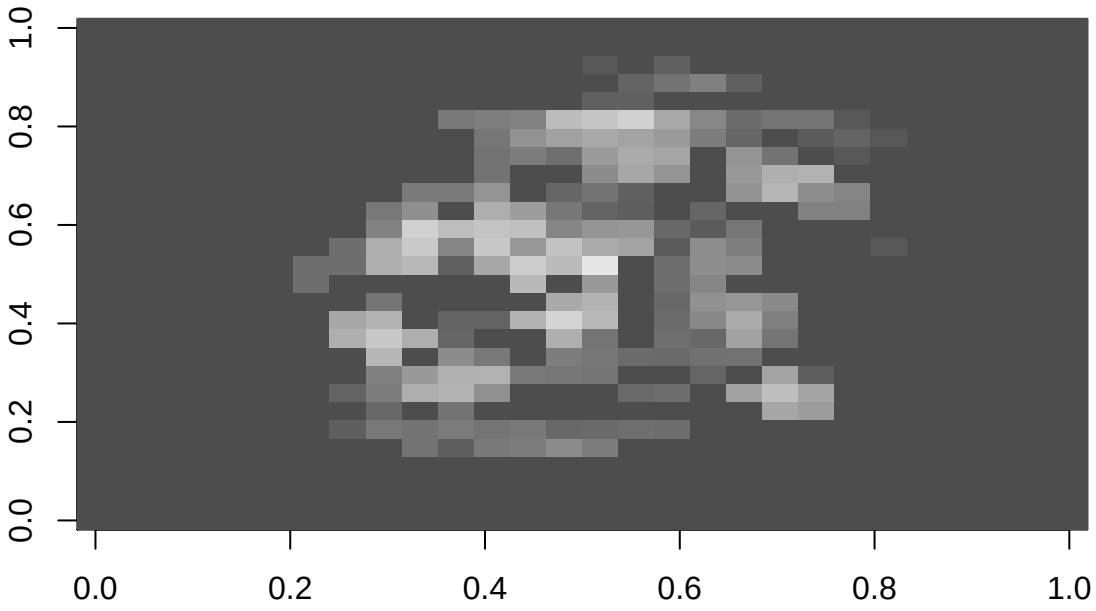
```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.877
```

```
augment(digit.bag.model, digit.test.tbl) %>%
  conf_mat(truth=digit, estimate= .pred_class)
```

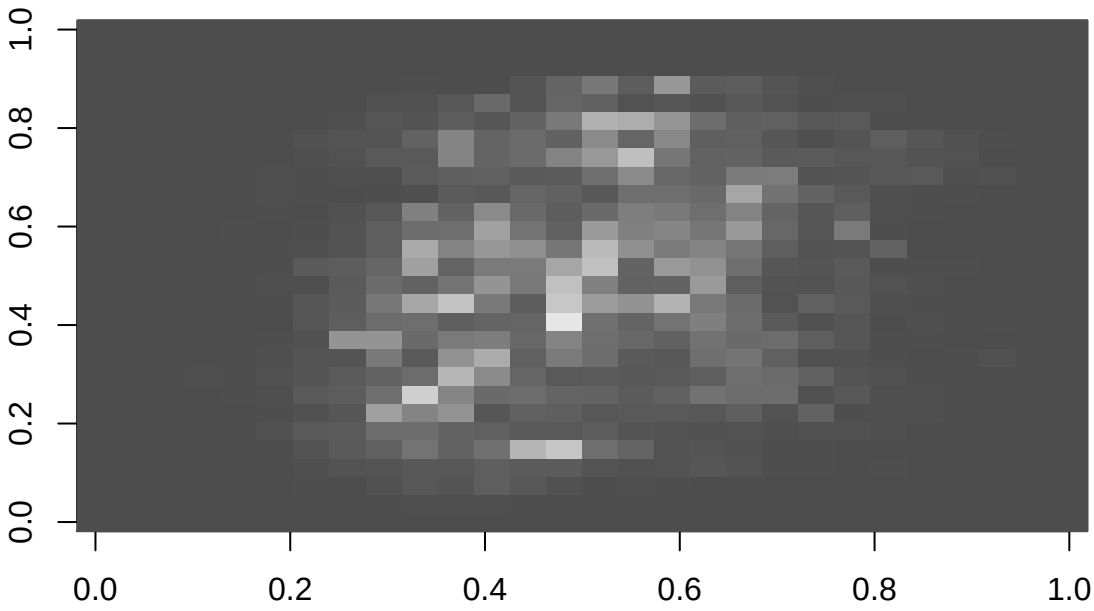
```
##           Truth
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0  85   0   2   4   0   1   2   1   1   1
##          1   0 122   0   0   0   0   0   2   3   0
##          2   1   2  85   3   0   0   1   4   5   1
##          3   0   0   0  72   1   2   0   2   4   0
##          4   0   0   1   0  91   3   3   1   2   4
##          5   0   0   0   8   0  75   2   0   3   2
##          6   3   0   1   0   1   2  90   0   1   0
##          7   0   0   5   5   1   0   1  89   0   5
##          8   0   0   1   2   0   3   1   1  71   3
##          9   0   0   3   0   8   0   0   2   2  97
```

Notice that the variable importance in a bag is the sum of the variable importance across all splits for all trees. We can plot our variable importance for maximal trees and for our bagging model below. Notice how in bagging more pixels end up being used, however some pixels in the middle of the image are used repetitively

```
create_image_vip(digit.rtree.model) %>%
  plotImage()
```




```
create_image_vip(digit.bag.model) %>%
  plotImage()
```



Forests

Let's remember that the standard deviation of $\frac{(X_1+\cdots+X_n)}{n}$ is $\frac{\sigma}{\sqrt{n}}$ if $X_1,\ldots,X_n$ are *independent*. In practice the outcome of a maximal tree can be quite correlated with the outcome of a different maximal tree since some variables are dominant when we do splits over our training data (think for examples the pixels in the center of our digit images)

In *random forests* we try to decorrelate each of these trees by selecting a random fraction of the variables at each level of the tree, to force different trees to not be related to each other. In practice the number of variables can be selected by using the parameter `mtry` and usually is:

- $\frac{n}{3}$ for regression trees
- $\sqrt{n}$ for classification trees.

This can be implemented using the following code

```
ranger_recipe <-
  recipe(formula = digit ~ ., data = digit.train.tbl)

ranger_spec <-
  rand_forest(trees = 100, mtry=28) %>%
  set_mode("classification") %>%
  set_engine("ranger",importance = "impurity")

ranger_workflow <-
  workflow() %>%
```

```
  add_recipe(ranger_recipe) %>%
  add_model(ranger_spec)

digit.forest.model <- fit(ranger_workflow, digit.train.tbl)
```

Notice that we can calculate as before our accuracy and confusion matrix and we get an improvement over our bagging approach:

```
augment(digit.forest.model, digit.test.tbl) %>%
  accuracy(truth=digit, estimate= .pred_class)
```

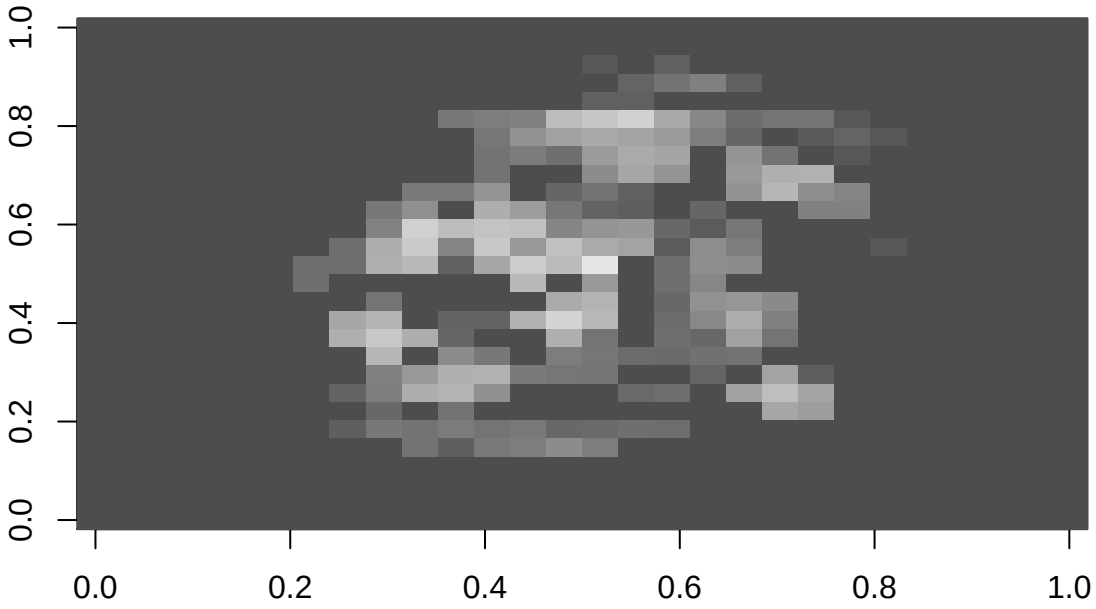
```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.893
```

```
augment(digit.forest.model, digit.test.tbl) %>%
  conf_mat(truth=digit, estimate= .pred_class)
```

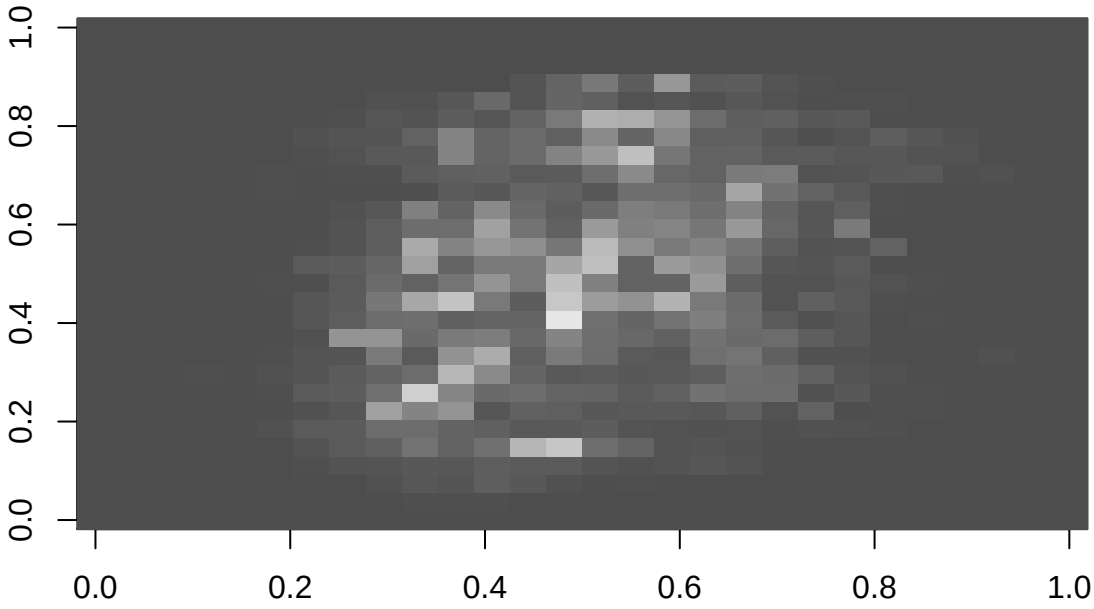
```
##            Truth
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0  88   0   0   5   0   1   2   0   2   1
##          1   0 123   0   0   0   0   1   2   2   0
##          2   1   1  89   3   0   0   0   5   3   0
##          3   0   0   0  78   0   2   0   1   3   0
##          4   0   0   1   1  91   3   3   0   1   7
##          5   0   0   0   4   1  75   3   0   4   2
##          6   0   0   0   0   0   3  90   0   2   0
##          7   0   0   4   1   1   0   0  91   0   7
##          8   0   0   1   2   0   2   1   0  75   3
##          9   0   0   3   0   9   0   0   3   0  93
```

Finally, let's plot our variable importance for our three methods:

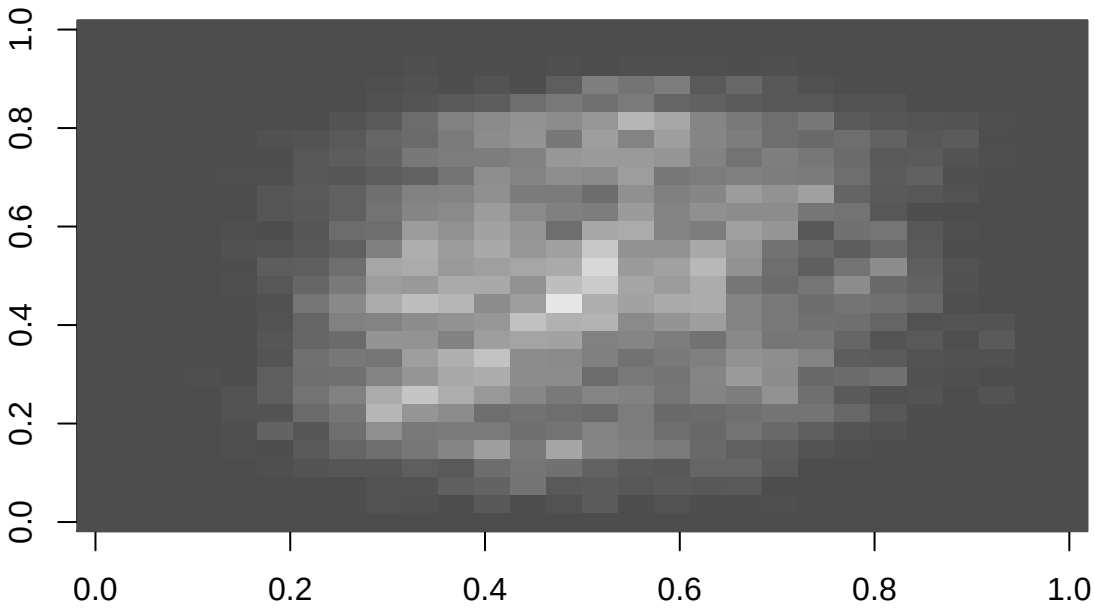
```
create_image_vip(digit.rtree.model) %>%
  plotImage()
```




```
create_image_vip(digit.bag.model) %>%
  plotImage()
```




```
create_image_vip(digit.forest.model) %>%
  plotImage()
```



Notice that when compared to our bagging model, our random forest starts using more pixels in our image.

Tissue classification

In the next set of exercises we will be issue the `tissue_gene_expression` dataset.

```
library(dslabs)
data(tissue_gene_expression)

tissue.gene.tbl <- tissue_gene_expression$x %>%
  as_tibble() %>%
  mutate(tissue = as.factor(tissue_gene_expression$y))

set.seed(4272022)
tissue.split <- initial_split(tissue.gene.tbl, prop=0.5)
tissue.train.tbl <- training(tissue.split)
table(tissue.train.tbl$tissue)
```

```
##
##  cerebellum       colon endometrium hippocampus      kidney       liver
##          19          17           6          19          15          14
##    placenta
##           4
```

```
tissue.test.tbl <- testing(tissue.split)
```

3. Create an optimal classification tree to predict tissue type by selecting the optimal `cp` using 10-fold cross-validation. Calculate the accuracy and confusion matrix using your testing dataset. What do you observe happening for placenta?

```
set.seed(4272022)
tissue.folds <- vfold_cv(tissue.train.tbl, v = 10)


tissue.model <-
  decision_tree(cost_complexity=tune()) %>%
  set_mode("classification") %>%
  set_engine("rpart")

  tissue.recipe <- recipe(tissue ~ ., data=tissue.train.tbl)


  tissue.wflow <- workflow() %>%
    add_recipe(tissue.recipe) %>%
    add_model(tissue.model)

  tissue.grid <-
    grid_regular(cost_complexity(), levels = 4)

  tissue.res <-
    tune_grid(
      tissue.wflow,
      resamples = tissue.folds,
      grid = tissue.grid)

  best.cp <- select_best(tissue.res, desc(cost_complexity), metric = "accuracy")
  tissue.final.wf <- finalize_workflow(tissue.wflow, best.cp)
  tissue.final.fit <- fit(tissue.final.wf, tissue.train.tbl)

  augment(tissue.final.fit, tissue.test.tbl)%>%
    accuracy(tissue, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.926
```

```
  augment(tissue.final.fit, tissue.test.tbl)%>%
    conf_mat(tissue, .pred_class)
```

```
##              Truth
## Prediction    cerebellum colon endometrium hippocampus kidney liver placenta
##   cerebellum          19     0           1           0      0     0        0
##   colon                0    17           0           0      0     0        0
##   endometrium          0     0           7           0      3     0        2
##   hippocampus          0     0           0          12      0     0        0
##   kidney               0     0           1           0     21     0        0
##   liver                0     0           0           0      0    12        0
##   placenta             0     0           0           0      0     0        0
```

We see an accuracy of 92.6%, pretty good overall. We also see that none of the placentas are correctly classified (0 in that diagonal slot of the confusion matrix). Granted, there are only 2 placentas in our testing

dataset, but none were correctly classified.

4. From the previous point we notice that missclassified all of the placentas. Note that the number of placentas in our dataset is six (four of them in training) , and that, by default, `rpart` requires 20 observations before splitting a node. Hence it is not possible to have a leaf where placentas are the majority when using our default `min_n`. Rerun the analysis in 3 by setting up `min_n=1`. Does the accuracy increase? Look at the confusion matrix again. Calculate the top-5 most important features by using `vip`. Check in `genecards` a couple of the genes that you obtain and argue if they make biological sense.

```
set.seed(4272022)
tissue.folds <- vfold_cv(tissue.train.tbl, v = 10)


tissue.model <-
  decision_tree(cost_complexity=tune(), min_n = 1) %>%
  set_mode("classification") %>%
  set_engine("rpart")

  tissue.recipe <- recipe(tissue ~ ., data=tissue.train.tbl)


  tissue.wflow <- workflow() %>%
    add_recipe(tissue.recipe) %>%
    add_model(tissue.model)

  tissue.grid <-
    grid_regular(cost_complexity(), levels = 4)

  tissue.res <-
    tune_grid(
      tissue.wflow,
      resamples = tissue.folds,
      grid = tissue.grid)

  best.cp <- select_best(tissue.res, desc(cost_complexity), metric = "accuracy")
  tissue.final.wf <- finalize_workflow(tissue.wflow, best.cp)
  tissue.final.fit <- fit(tissue.final.wf, tissue.train.tbl)

  augment(tissue.final.fit, tissue.test.tbl)%>%
    accuracy(tissue, .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.926
```

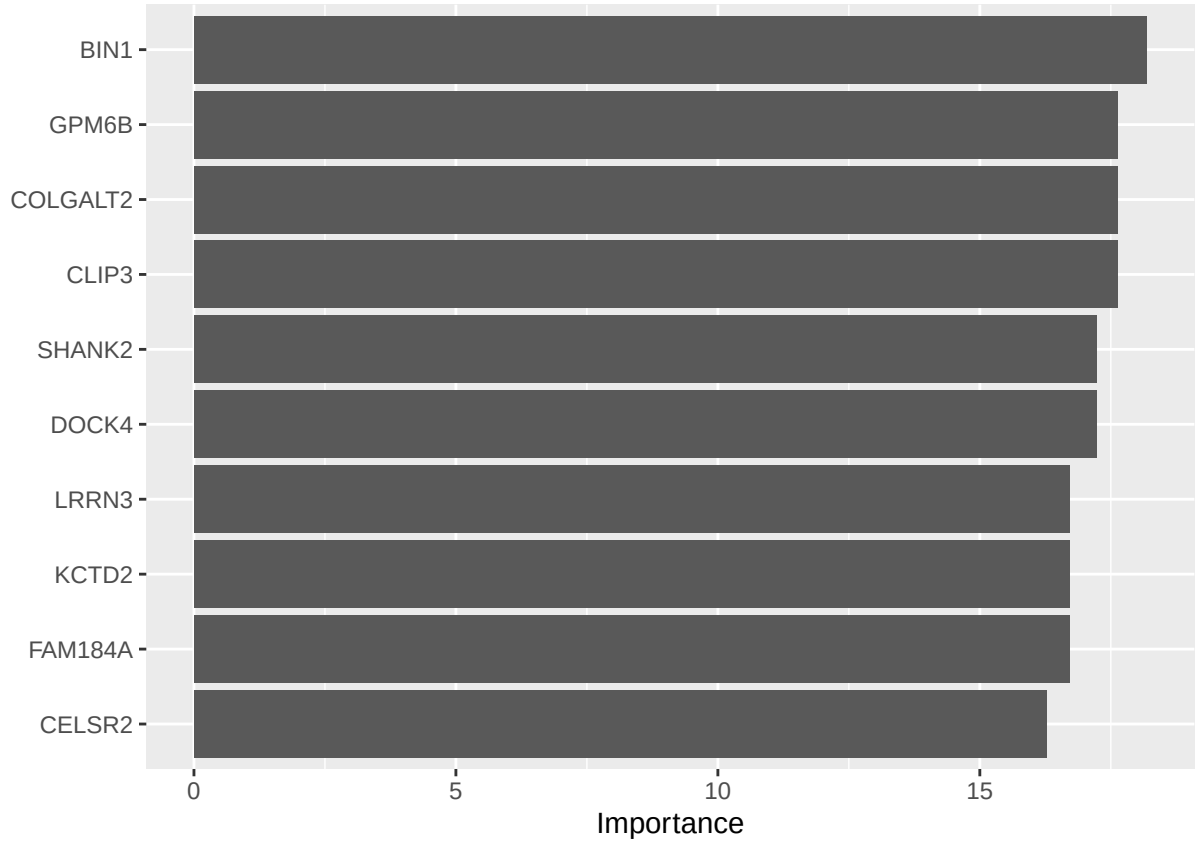
```
  augment(tissue.final.fit, tissue.test.tbl)%>%
    conf_mat(tissue, .pred_class)
```

```
##              Truth
## Prediction    cerebellum colon endometrium hippocampus kidney liver placenta
##   cerebellum          19     0           1           0      0     0        0
##   colon                0    17           0           0      0     0        0
##   endometrium          0     0           5           0      0     0        0
##   hippocampus          0     0           0          12      0     0        0
```

```
##    kidney              0     0           1           0     21     0        0
##    liver               0     0           0           0      0    12        0
##    placenta            0     0           2           0      3     0        2
```

The accuracy stays, the same here, but we now correctly classify the 2 placentas in our testing dataset.

```
#IMPORTANCE
tissue.final.fit %>%
  extract_fit_engine() %>%
  vip::vip()
```



We see BIN1, GPM6B, COGALT2, CLIP3, and SHANK2 are the 5 most important genes. BIN1 is a tumor suppressor, GPM6B encodes a protein that is present in most brain regions. COLGALT2 is predicted to be involved in collagen fibril organizing. Because most of these are involved with proteins and other tissues it makes sense they show up on the importance scale.

5. Repeat 4, by using a bagging model with `100` trees (notice `min_n=1`).

```
ranger_recipe <-
  recipe(formula = tissue ~ ., data = tissue.train.tbl)

ranger_spec <-
  rand_forest(trees = 100, mtry=500, min_n = 1) %>%
  set_mode("classification") %>%
  set_engine("ranger", importance = "impurity")

ranger_workflow <-
  workflow() %>%
  add_recipe(ranger_recipe) %>%
```

```
  add_model(ranger_spec)

tissue.bag.fit <- fit(ranger_workflow, tissue.train.tbl)

augment(tissue.bag.fit, tissue.test.tbl)%>%
  accuracy(tissue, .pred_class)
```

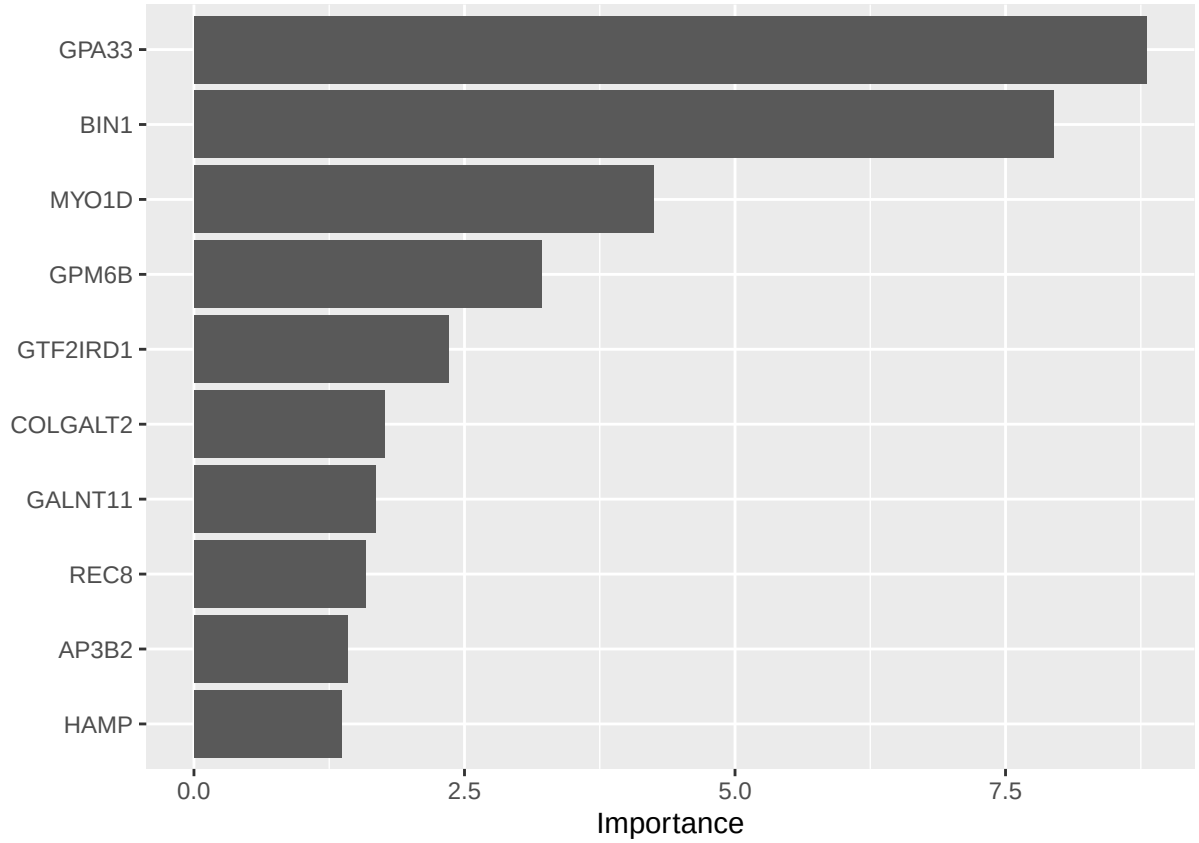
```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.989
```

```
augment(tissue.bag.fit, tissue.test.tbl)%>%
  conf_mat(tissue, .pred_class)
```

```
##              Truth
## Prediction    cerebellum colon endometrium hippocampus kidney liver placenta
##   cerebellum          19     0           0           0      0     0         0
##   colon                0    17           0           0      0     0         0
##   endometrium          0     0           8           0      0     0         0
##   hippocampus          0     0           0          12      0     0         0
##   kidney               0     0           1           0     24     0         0
##   liver                0     0           0           0      0    12         0
##   placenta             0     0           0           0      0     0         2
```

We see an accuracy of 97.9%, which is REALLY good.

```
tissue.bag.fit %>%
  extract_fit_engine() %>%
  vip::vip()
```



Here we see BIN1 and GPA33 as far and away most important.

6. Repeat 5, using a random forest with 100 trees.

```
set.seed(4272022)
ranger_recipe <-
  recipe(formula = tissue ~ ., data = tissue.train.tbl)

ranger_spec <-
  rand_forest(trees = 100, mtry=23, min_n = 1) %>%
  set_mode("classification") %>%
  set_engine("ranger", importance = "impurity")

ranger_workflow <-
  workflow() %>%
  add_recipe(ranger_recipe) %>%
  add_model(ranger_spec)

tissue.forest.fit <- fit(ranger_workflow, tissue.train.tbl)

augment(tissue.forest.fit, tissue.test.tbl)%>%
  accuracy(tissue, .pred_class)
```

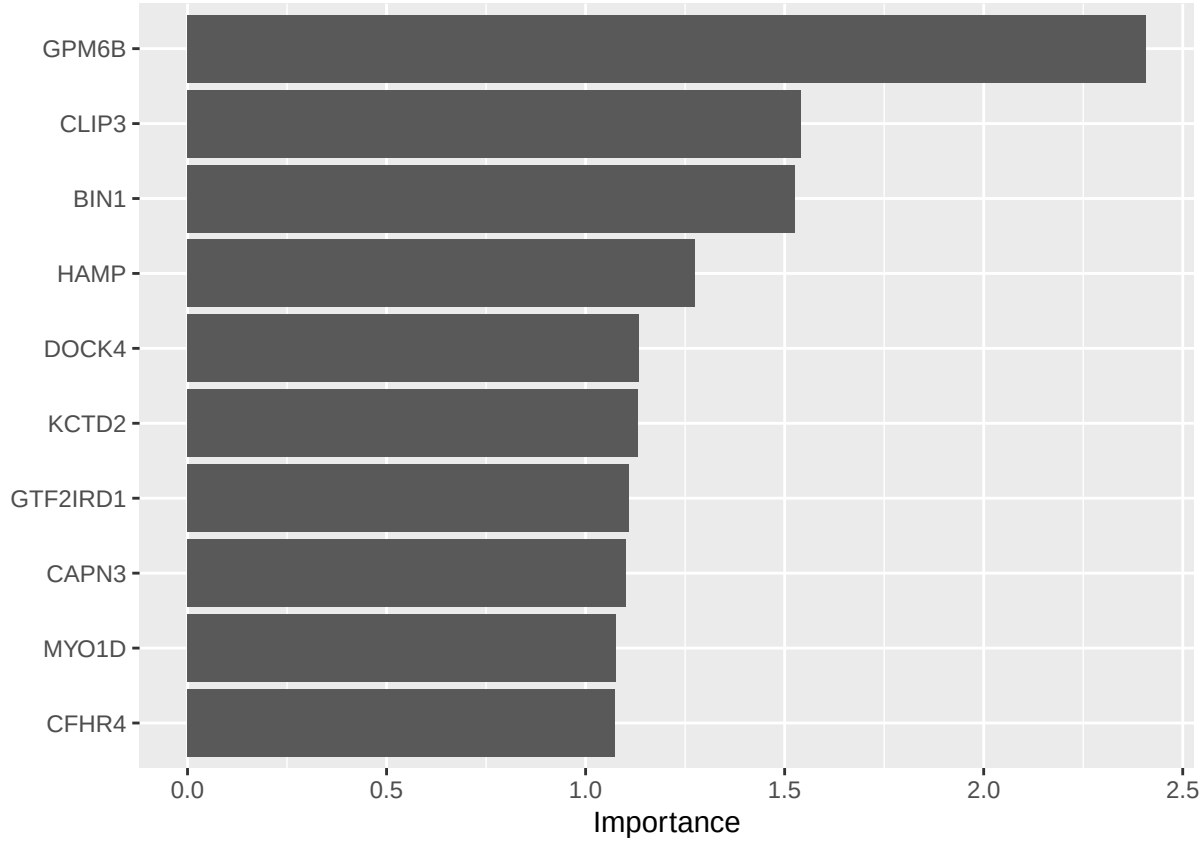
```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass         1
```

```
augment(tissue.forest.fit, tissue.test.tbl)%>%
  conf_mat(tissue, .pred_class)
```

```
##              Truth
## Prediction    cerebellum colon endometrium hippocampus kidney liver placenta
##   cerebellum          19     0           0           0      0     0         0
##   colon                0    17           0           0      0     0         0
##   endometrium          0     0           9           0      0     0         0
##   hippocampus          0     0           0          12      0     0         0
##   kidney               0     0           0           0     24     0         0
##   liver                0     0           0           0      0    12         0
##   placenta             0     0           0           0      0     0         2
```

We see an accuracy of 1, meaning that every observation was predicted correctly!

```
tissue.forest.fit %>%
  extract_fit_engine() %>%
  vip::vip()
```

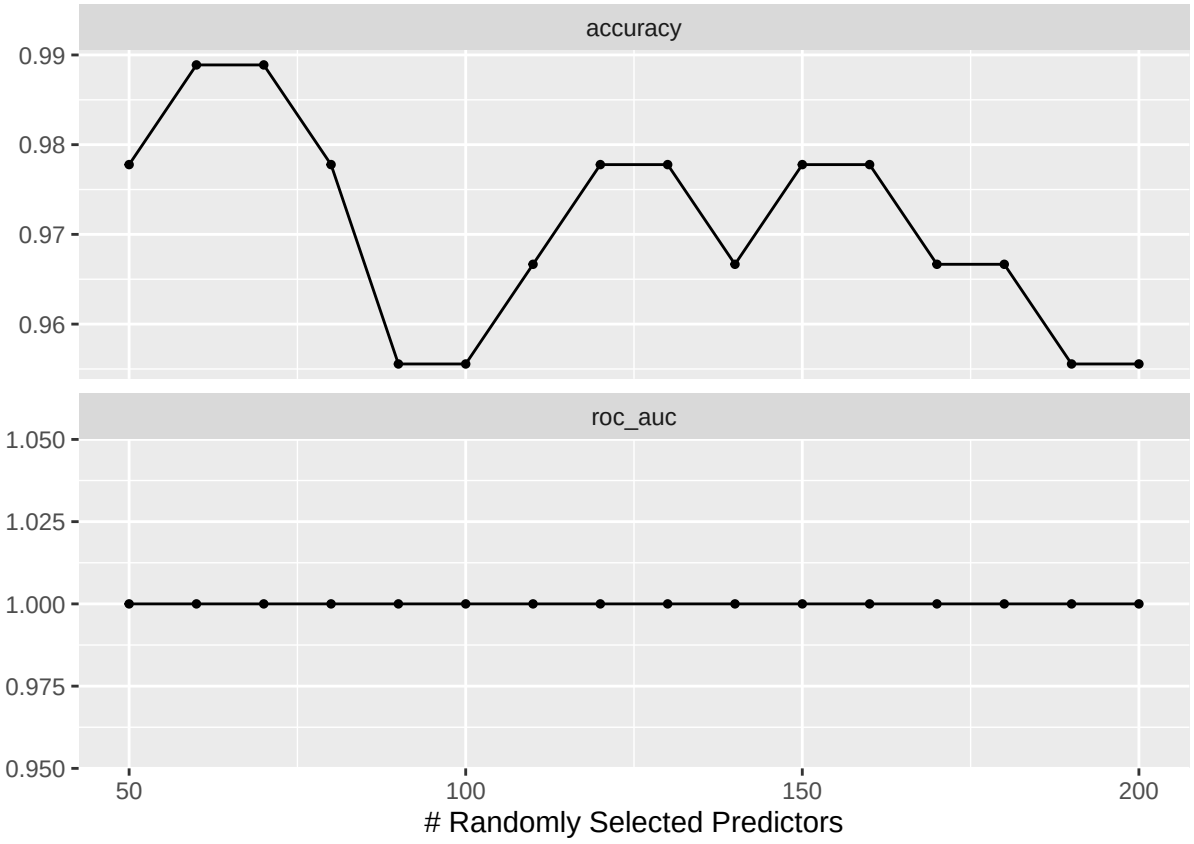


We see that GPM6B, CLIP3, and BIN1 are still the most important, similar to our prior models for the most part.

7. Repeat 6 but this time the optimal `mtry` using 10-fold cross validation. For values of `mtry` use at least 10 values from 50 to 200. Compare the top-5 most important variables with what you got in 5 and in 4.

```
set.seed(4272022)
tissue.folds <- vfold_cv(tissue.train.tbl, v = 10)
```

```
ranger_recipe <-
  recipe(formula = tissue ~ ., data = tissue.train.tbl)

ranger_spec <-
  rand_forest(trees = 100, mtry=tune(), min_n = 1) %>%
  set_mode("classification") %>%
  set_engine("ranger", importance = "impurity")

ranger_workflow <-
  workflow() %>%
  add_recipe(ranger_recipe) %>%
  add_model(ranger_spec)

tissue.grid <- tibble(mtry = seq(50, 200, by = 10))

tissue.res <-
    tune_grid(
      ranger_workflow,
      resamples = tissue.folds,
      grid = tissue.grid)

autoplot(tissue.res)
```




```
best.mtry <- select_best(tissue.res, metric = "accuracy")
tissue.forest.final.wflow <- finalize_workflow(ranger_workflow, best.mtry)
tissue.forest.final.fit <- fit(tissue.forest.final.wflow, tissue.train.tbl)
```

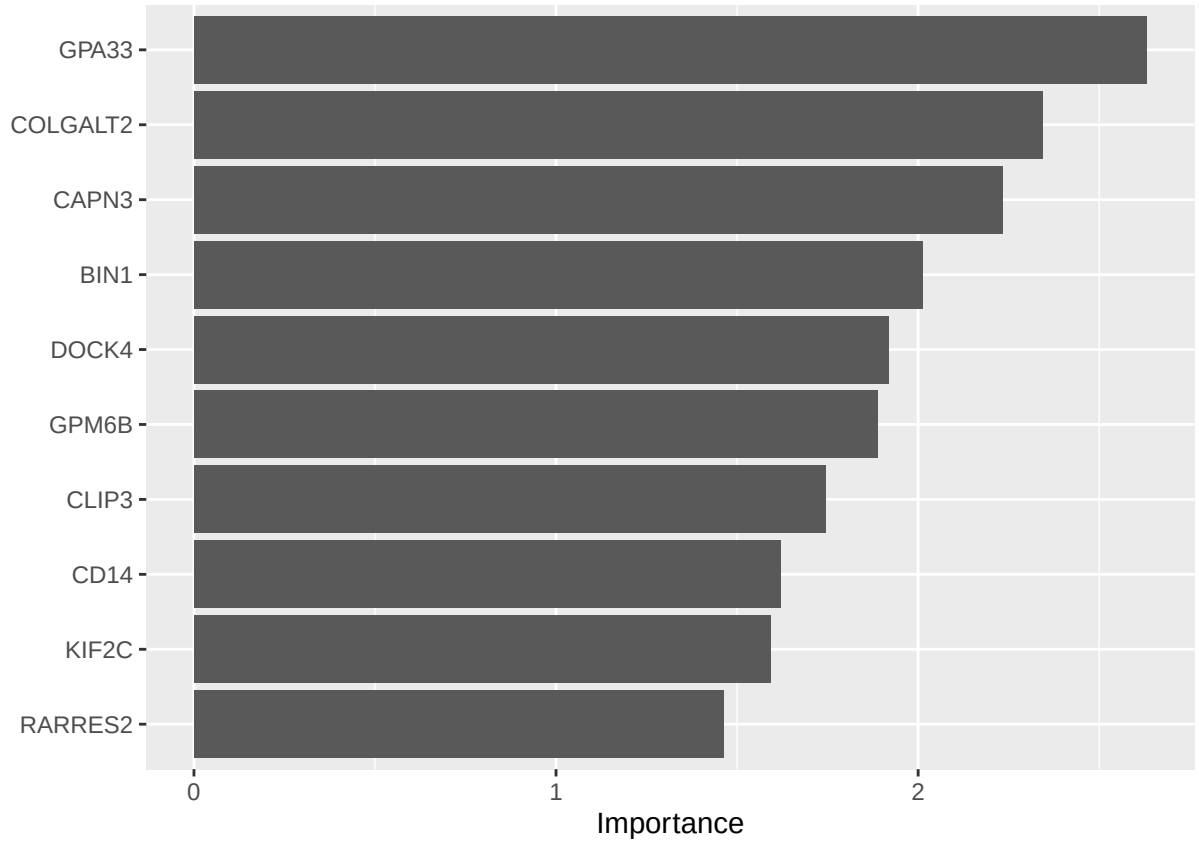
```
augment(tissue.forest.final.fit, tissue.test.tbl)%>%
  accuracy(tissue, .pred_class)
```

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

```
augment(tissue.forest.final.fit, tissue.test.tbl)%>%
  conf_mat(tissue, .pred_class)
```

```
##              Truth
## Prediction    cerebellum colon endometrium hippocampus kidney liver placenta
##   cerebellum          19     0           0           0      0     0         0
##   colon                0    17           0           0      0     0         0
##   endometrium          0     0           9           0      0     0         0
##   hippocampus          0     0           0          12      0     0         0
##   kidney               0     0           0           0     24     0         0
##   liver                0     0           0           0      0    12         0
##   placenta             0     0           0           0      0     0         2
```

```
tissue.forest.final.fit %>%
  extract_fit_engine() %>%
  vip::vip()
```



We once again see an accuracy of 100%, which occurred when we set mtry to 60. Our most important predictors were GPA33, COLGALT2, CAPN3, and BIN1. Most of those were carry overs from previous

models, although I don't remember seeing CAPN3 in the top 5, so it's interesting it showed up this time.