Random Forests

Jaime Davila

4/27/2021

MSE is for regression, and misclassification is for classification.

misclassification error = 1 - accuracy for classification datasets.

MSE is mean squared error which is the average RSS (residual sum squared) MSE is for regression usually.

MSE = RMSEˆ2

"Root Mean Square Error (RMSE) is the standard deviation of the residuals (prediction errors). Residuals are a measure of how far from the regression line data points are; RMSE is a measure of how spread out these residuals are." residuals = leftover value. (what about classification? is it possible to do this? yeah?)

Tree: can be regression or classification tree.

*****is MSE the same as misclassification error? can RMSE be used for classification and prediction? what about accuracy? _____ regression trees:

'stump' contains all data. get average value from stump to calculate MSE.

MSE = variance. sum of predicted - actual. (average - xi)

rmse = standard deviation?

500 observations: pick some xi < t (t is threshhold) so mse is minimized?

don't really get what prof is explaining.

keep branching out downwards in the direction that minimizes mse?

cost-complexity:

variables: min_n, cost_complexity. (me: what about tree_depth?)

---

bootstrap:

divide up data into various chunks multiple times. These are 'bags'. get a regression tree for each chunk of data. **** Notice for each tree, cost complexity is 0 - get maximal trees. Prediction is average .pred of each tree. _____ classification trees:

how do 'average' the bags for classification? get the most common. how do you calculate that though? oh I think I have an idea.

step_dummy: used in ridge classification because it turns classes into probabilities?

---

random forest:

when building a tree, instead of selecting from all possible variables, you just pick a smaller subset, at random, from the original variables.

you do this because you 'force' the variables/predictors to be independent.

Introduction

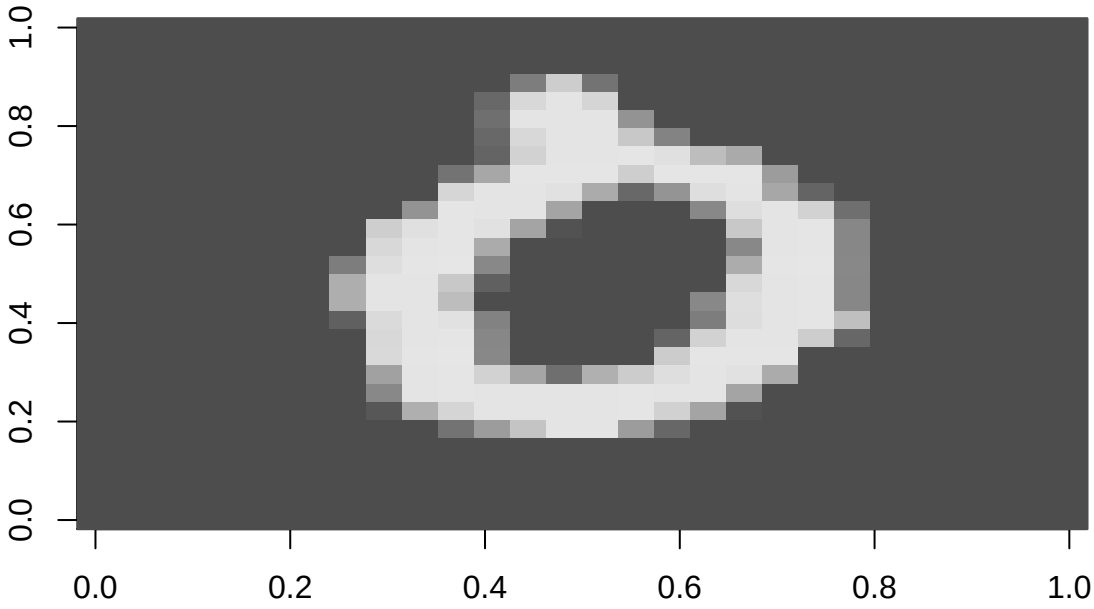
Today we will be using a small subset of the MNIST dataset, by selecting 1000 digits for both training and testing:

```
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]))
```

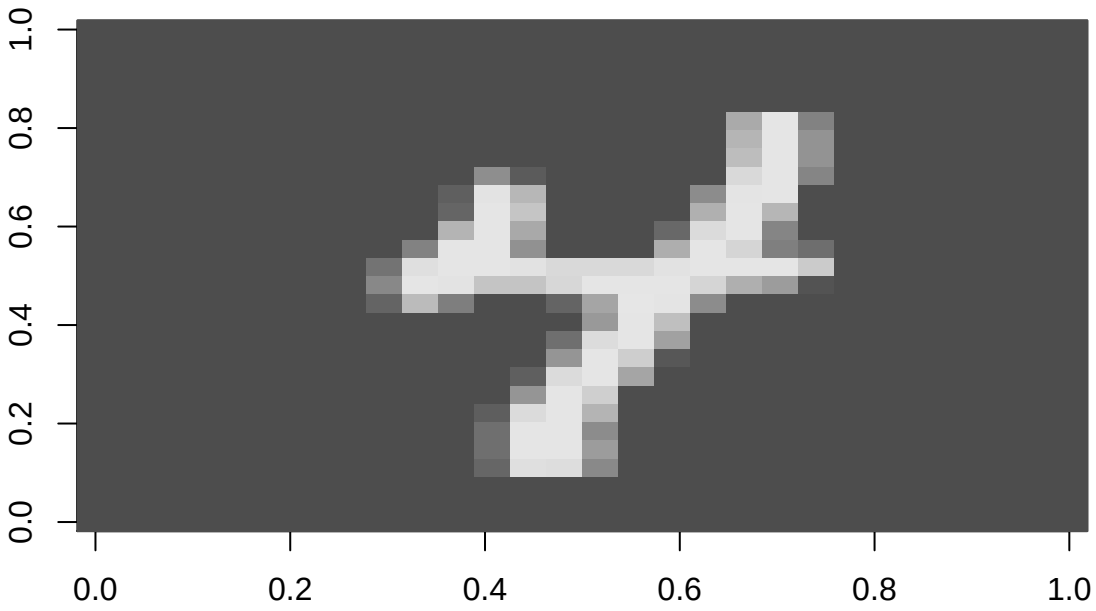
And as before let's make a couple of functions to plot particular digits from our dataset

```
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 (digit.train.tbl) {
  # Set up the model, recipe and workflow
  digit.model <-
  decision_tree(cost_complexity=0) %>%# maximal tree: cp = 0
  set_mode("classification") %>%
  set_engine("rpart")#

  digit.recipe <- recipe(digit ~., data=digit.train.tbl)
  digit.wflow <- workflow() %>%
    add_recipe(digit.recipe) %>%
    add_model(digit.model)
  # Fit the worfklow using the training dataset
  fit(digit.wflow, digit.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(truth = digit, estimate = .pred_class)
```

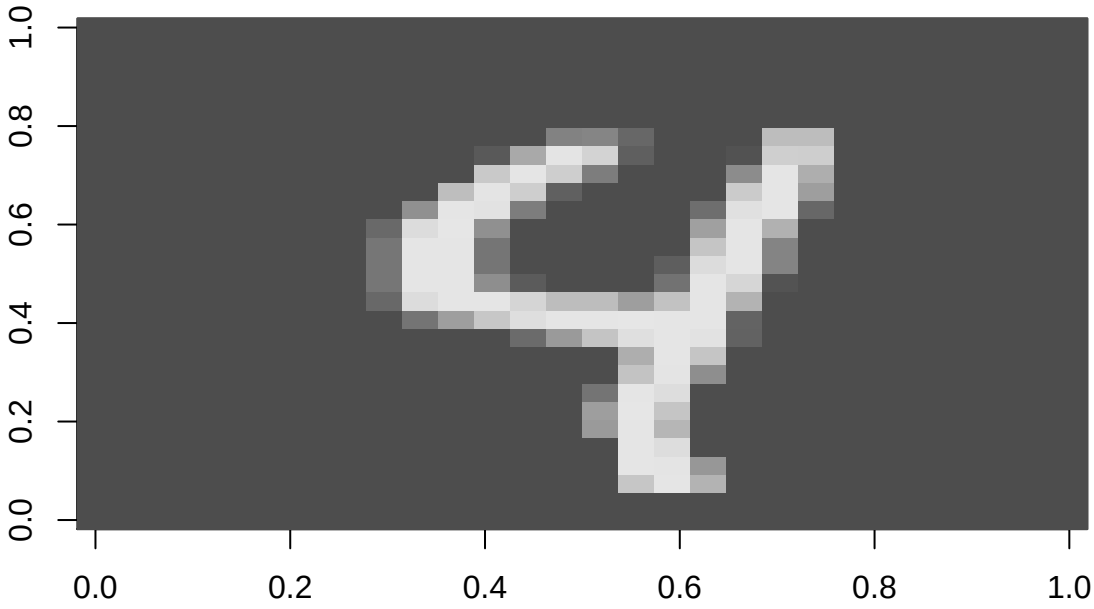
```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.663
```

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

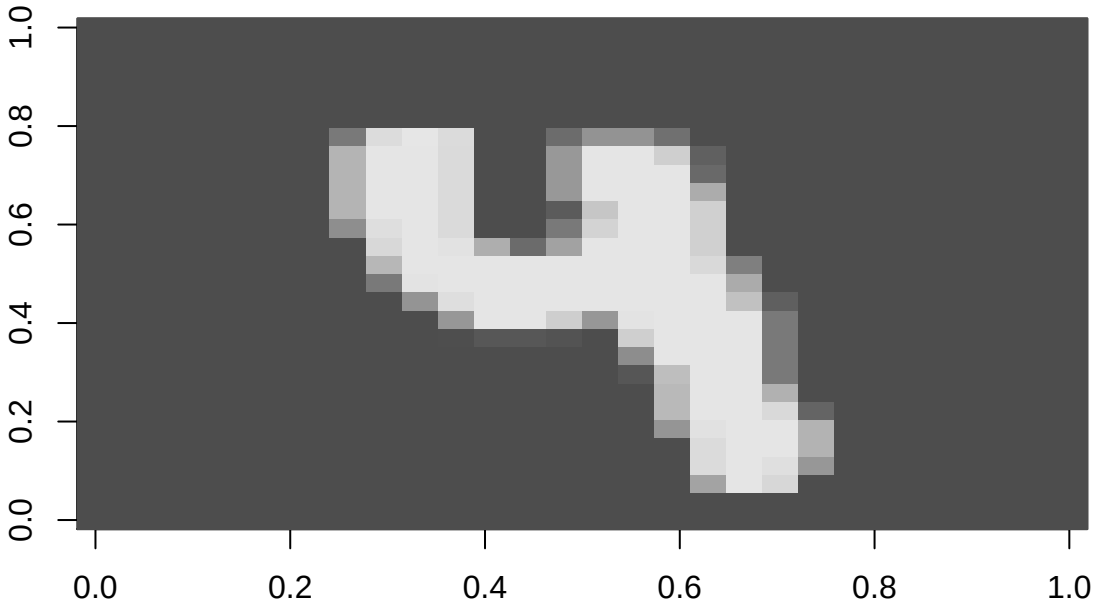
```
##            Truth
## Prediction   0   1   2   3   4   5   6   7   8   9
##          0  73   0   3   6   4   6   4   1   6   1
##          1   0 114   3   2   0   0   0   3   3   0
##          2   5   5  70   8   7   1  14   5   7   2
##          3   0   2   2  47   1   6   0   0   2   1
##          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
```

I got curious about why some 4's get classified as 9's so I decided to take a look:

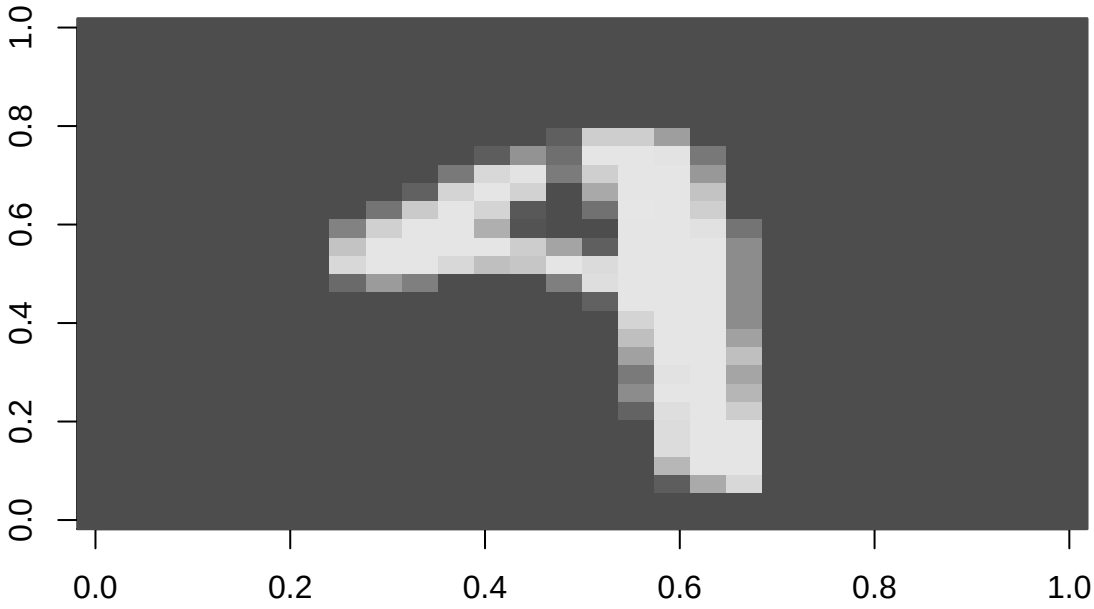
```
miss.4s.tbl <- augment(digit.rtree.model,digit.test.tbl) %>%
  filter(digit==4 & .pred_class==9)

plot_row(miss.4s.tbl[2,])
```



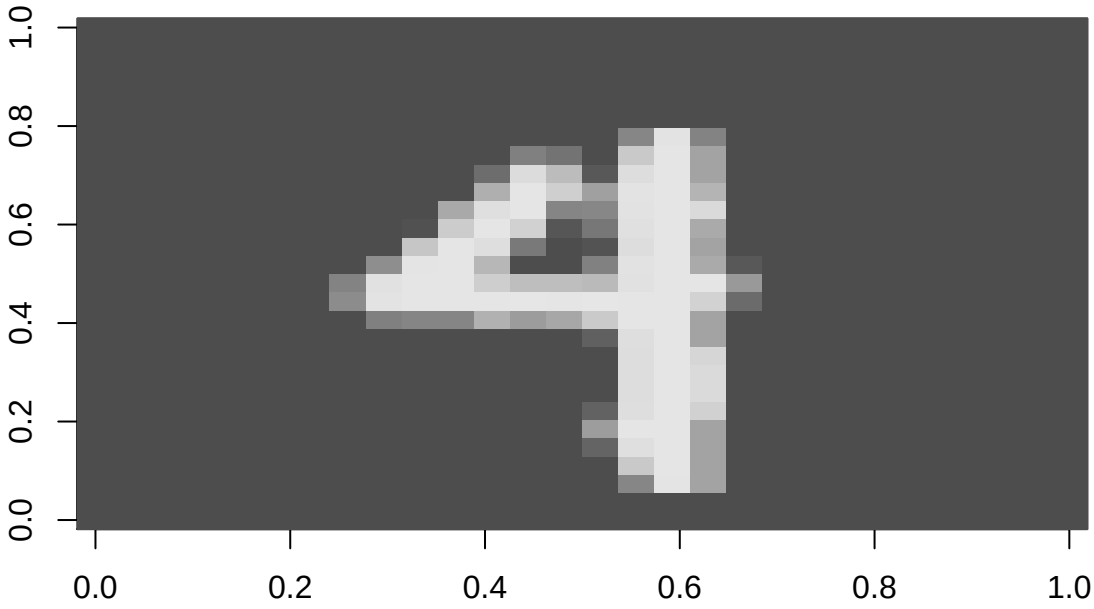

```
plot_row(miss.4s.tbl[6,])
```



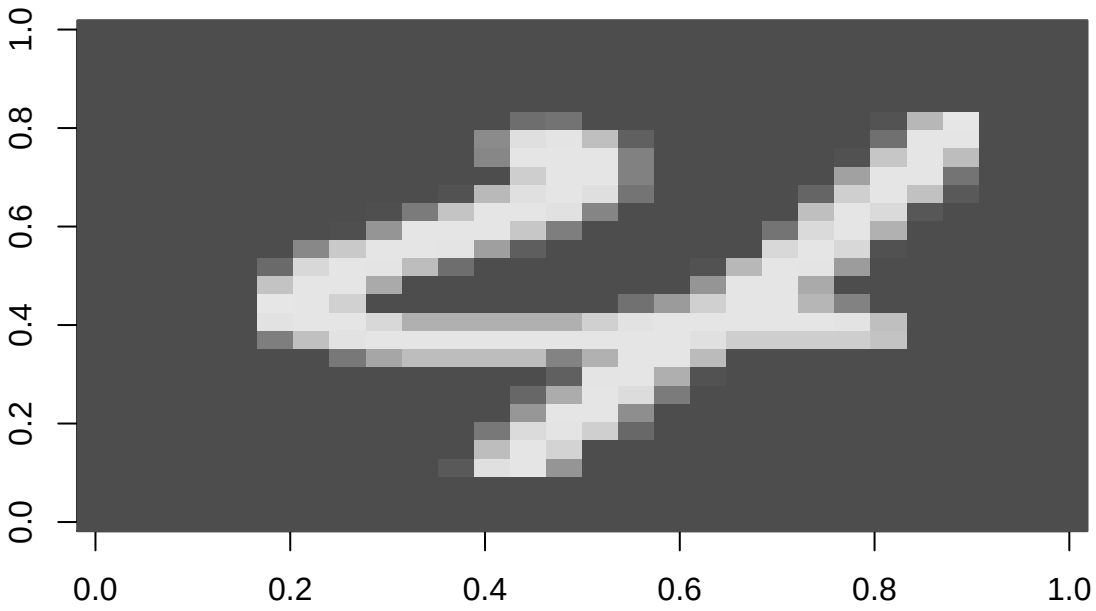

```
plot_row(miss.4s.tbl[9,])
```




```
plot_row(miss.4s.tbl[13,])
```




```
plot_row(miss.4s.tbl[8,])
```

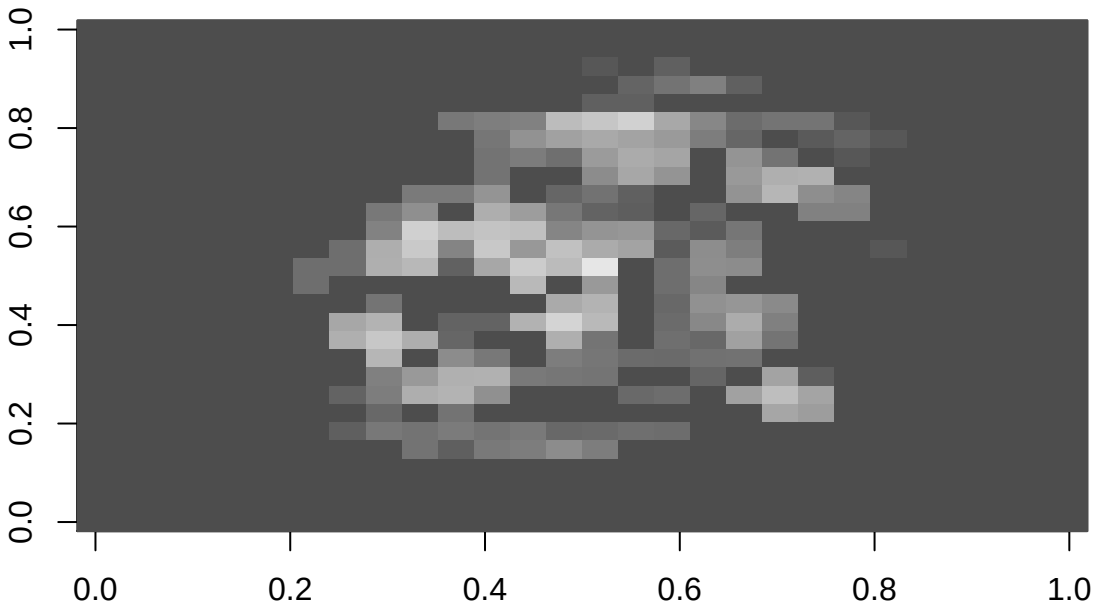


On a different note, we can define a function that will allow us to plot the pixel importance for a particular model.

```
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:

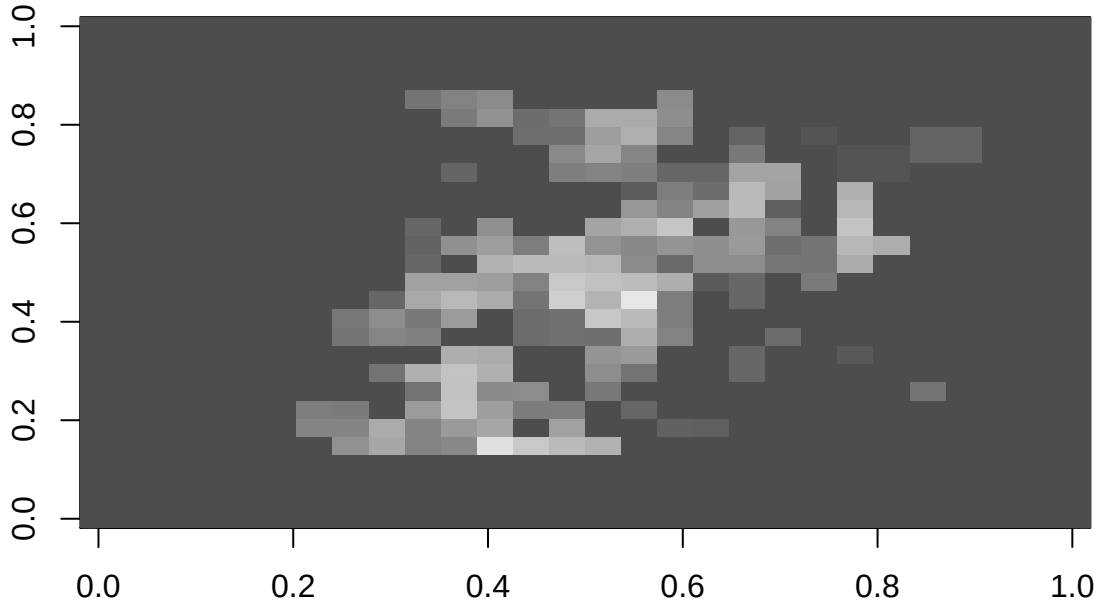
```
analysis(bootstrap.split$splits[[2]])
```

```
## # 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
```

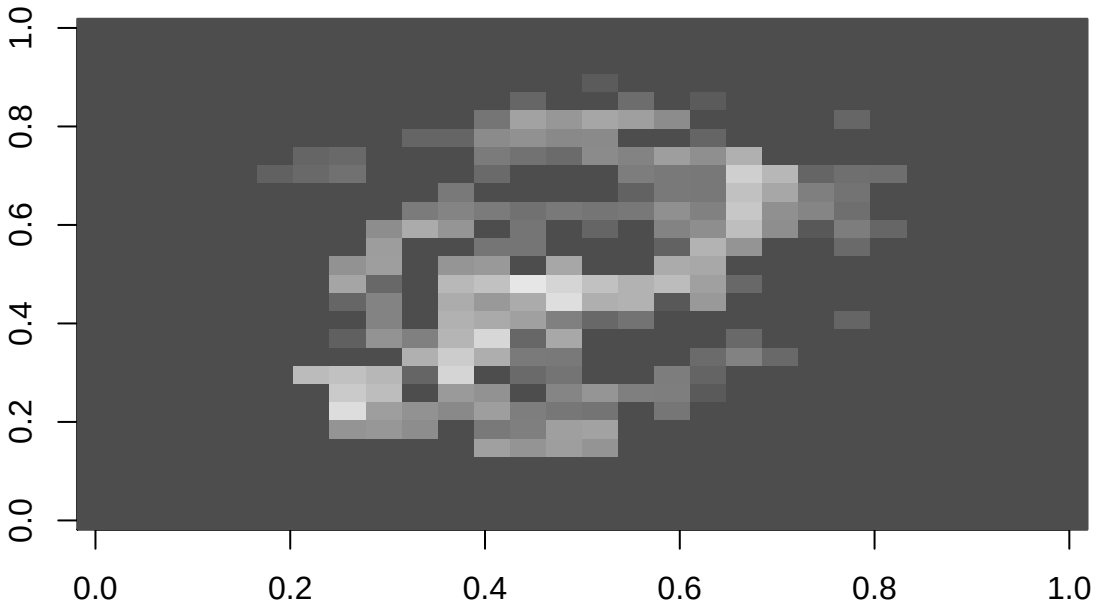
```
##  2     0     0     0     0     0     0     0     0     0     0     0     0     0
##  3     0     0     0     0     0     0     0     0     0     0     0     0     0
##  4     0     0     0     0     0     0     0     0     0     0     0     0     0
##  5     0     0     0     0     0     0     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?

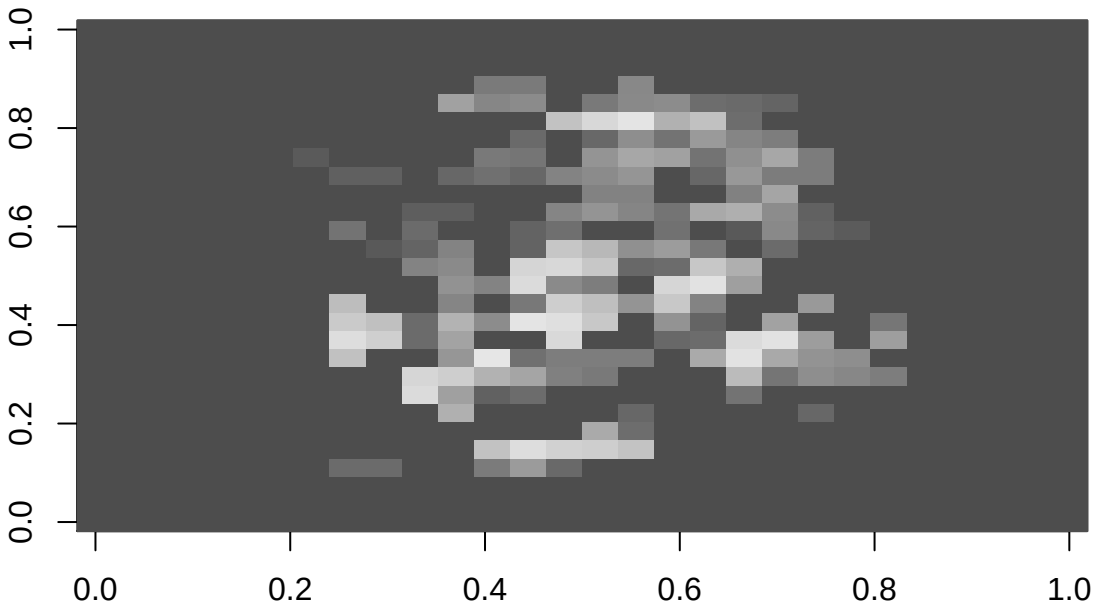
```
split1.model <- create_rtree(analysis(bootstrap.split$splits[[1]]))
split2.model <- create_rtree(analysis(bootstrap.split$splits[[2]]))
split3.model <- create_rtree(analysis(bootstrap.split$splits[[3]]))

create_image_vip(split1.model) %>%
  plotImage()
```



```
create_image_vip(split2.model) %>%
  plotImage()
```




```
create_image_vip(split3.model) %>%
  plotImage()
```



Um, it's hard to tell which pixels that are common to each mode. They all look like mayo stains smeared on something. Vaguely the pixels in the middle are used the most.

prof: do it 20 times:

y <- 20 bootstrap.split <- bootstraps(digit.train.tbl, times = y) for (i in 1:y) { bootstrap.split%splits[[i]] %>% analysis() %>% create_rtree() %>% create_image_vip() %>% plotImage() %> print() }

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 can actually create a model that can be used on testing dataset.

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) %>% #  mtry is number of predictors that each bag considers in eac
  set_mode("classification") %>%
  set_engine("ranger",importance = "impurity")  # keep track of importance of variables so the importan

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
```

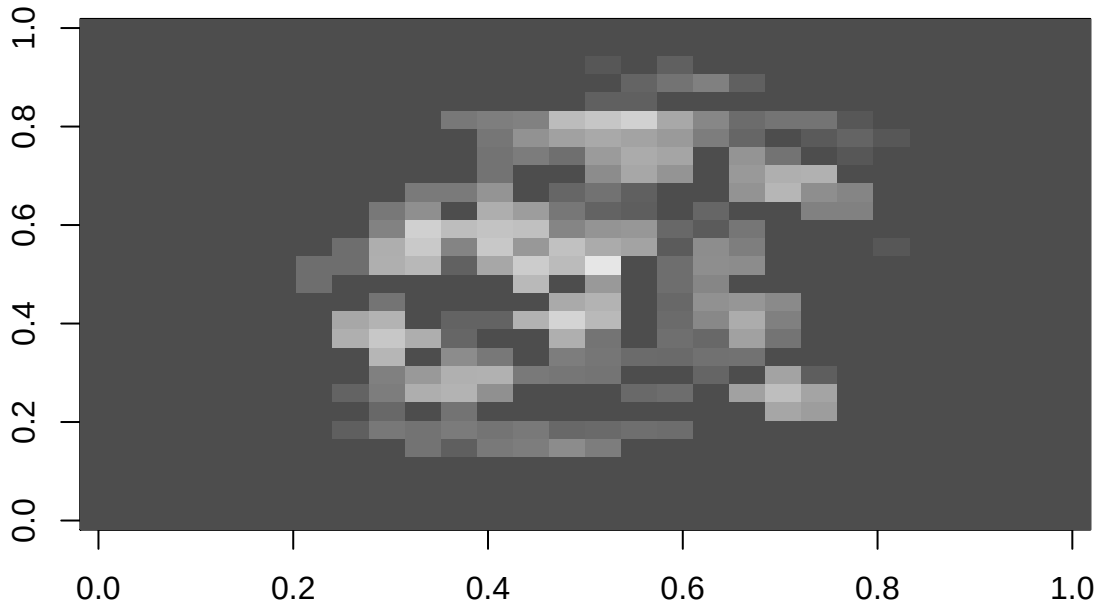
And let's look at the 4's that get classified as 9's in this model

```
miss.4s.tbl <- augment(digit.bag.model,digit.test.tbl) %>%
  filter(digit==4 & .pred_class==9)

for (i in 1:nrow(miss.4s.tbl)) {
  print(plot_row(miss.4s.tbl[i,]))
}
```

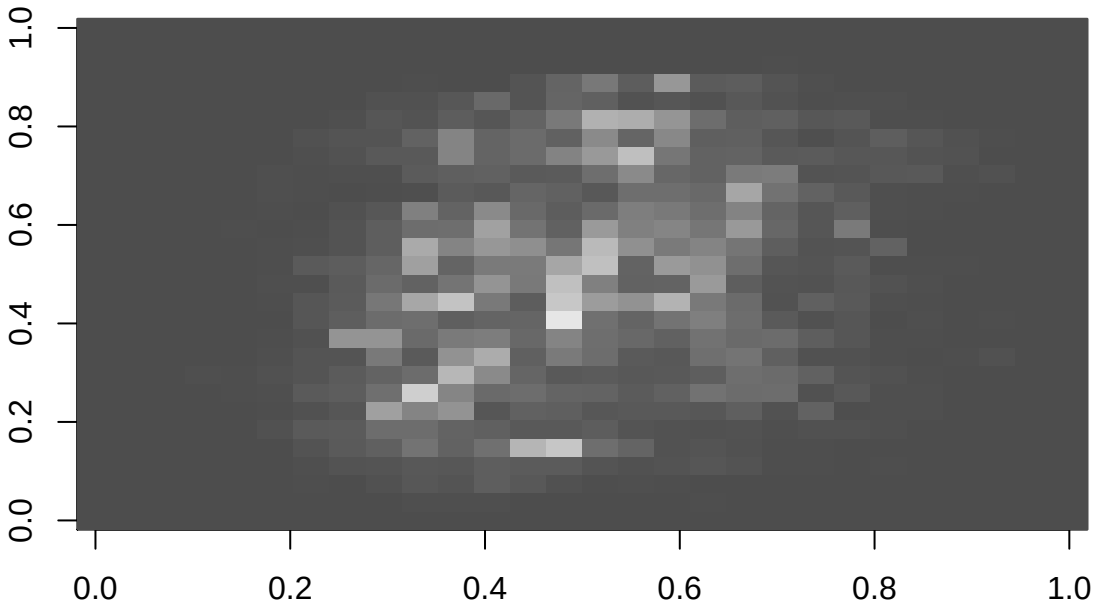
```
## NULL

## NULL

## NULL

## NULL

## NULL

## NULL

## NULL

## NULL
```

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()
```



---

could we use cross-validation in this? cross-validation is to maximize parameters.

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     0.898
```

```
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  87   0   0   3   0   1   3   0   1   1
##          1   0 123   0   0   0   0   1   1   1   1
##          2   1   1  91   3   0   0   0   8   4   0
##          3   0   0   0  79   0   1   0   0   5   0
##          4   0   0   1   1  93   4   3   0   1   6
##          5   0   0   0   4   0  77   3   0   2   1
##          6   1   0   0   0   0   2  89   0   0   0
##          7   0   0   3   2   1   0   1  87   0   6
##          8   0   0   1   2   1   1   0   1  77   3
##          9   0   0   2   0   7   0   0   5   1  95
```

And let's check the 4's that get classified as 9's in this last model

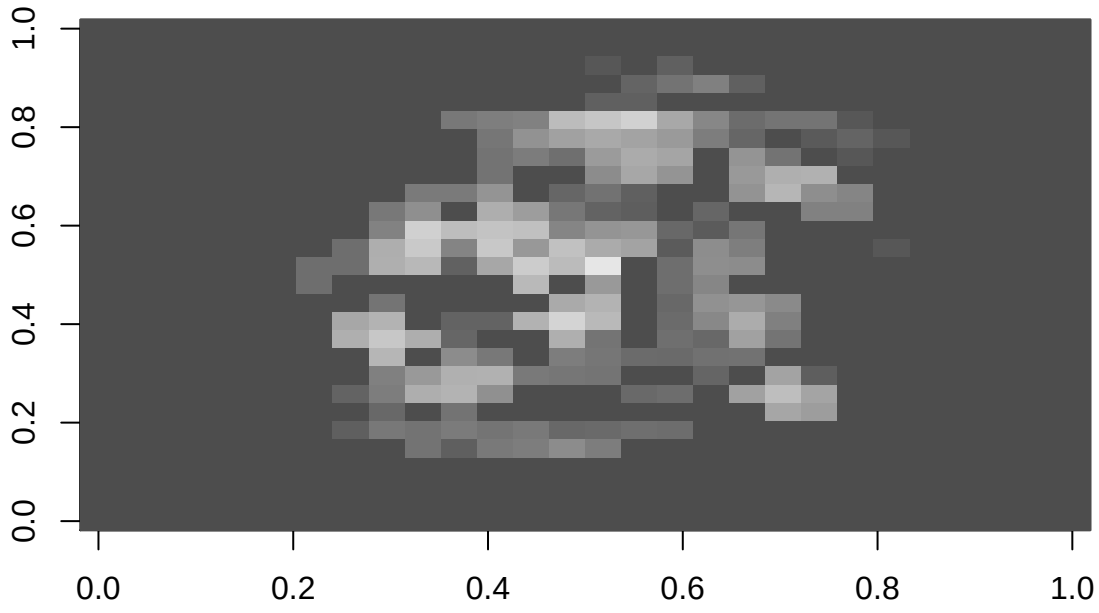
```
miss.4s.tbl <- augment(digit.forest.model,digit.test.tbl) %>%
  filter(digit==4 & .pred_class==9)

for (i in 1:nrow(miss.4s.tbl)) {
  print(plot_row(miss.4s.tbl[i,]))
}
```

```
## NULL

## NULL

## NULL

## NULL

## NULL

## NULL
```

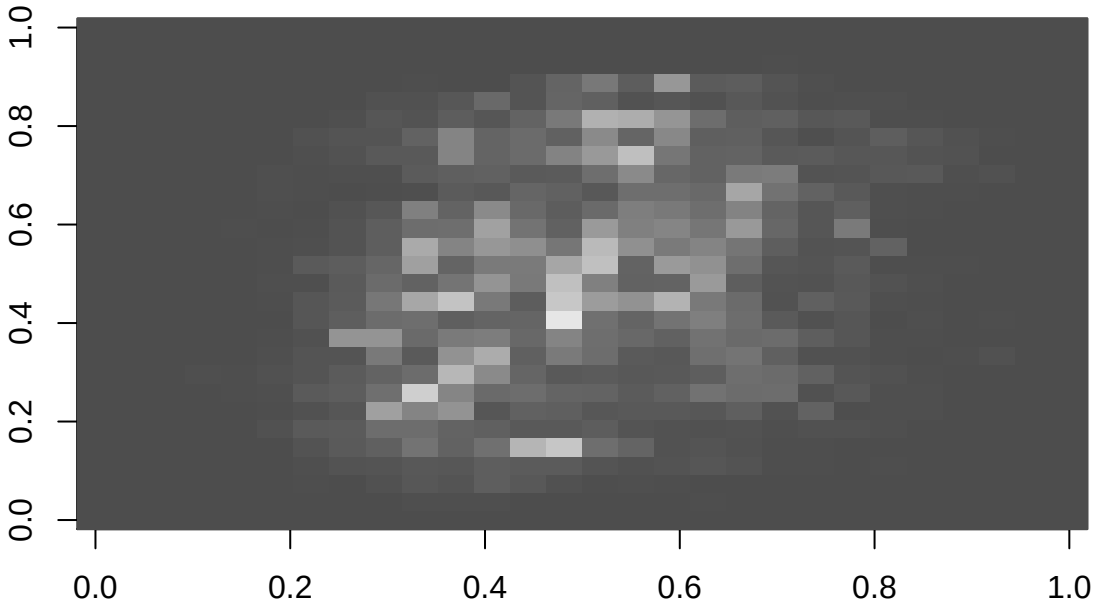
```
## NULL
```

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

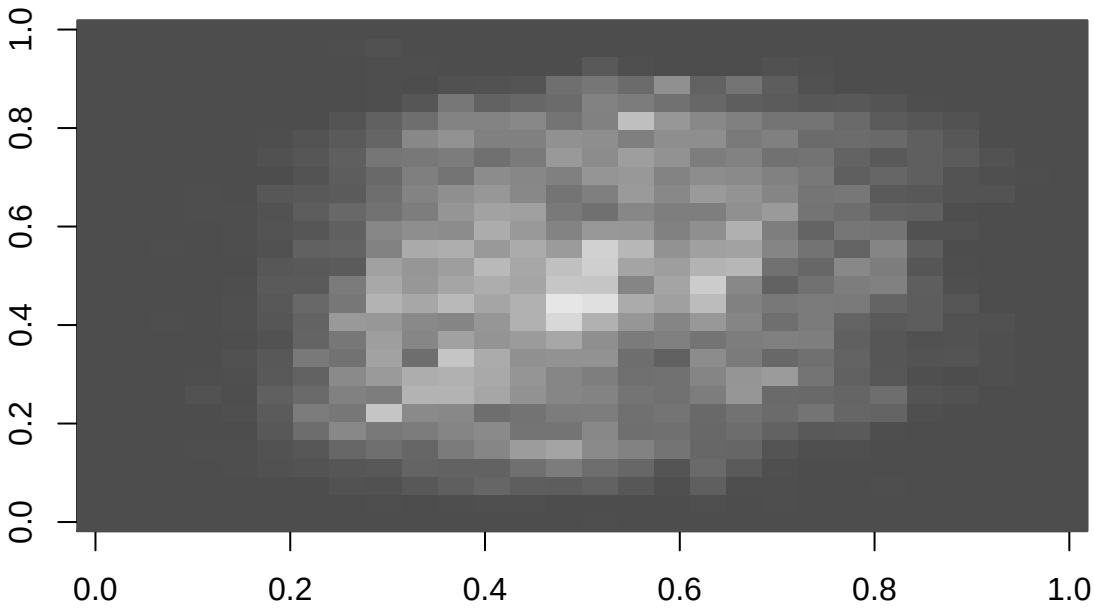
```
create_image_vip(digit.rtree.model) %>%
  plotImage()
```




```
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)
```

---

can we use cross-validation with the two models above (random tree and bagging ro whatever they are - double check). **** when is feedback day for project? this upcoming tuesday? thursday = last model we use. class. boosting or something. **** then presentation. prof says try to make the presentation 8 minutes long. I should download the mirror share or whatever for presentation. prof says we can present whatever we want. can make a presentation about it. we are presenting on teh tv, can prsent pdf or etc.

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?

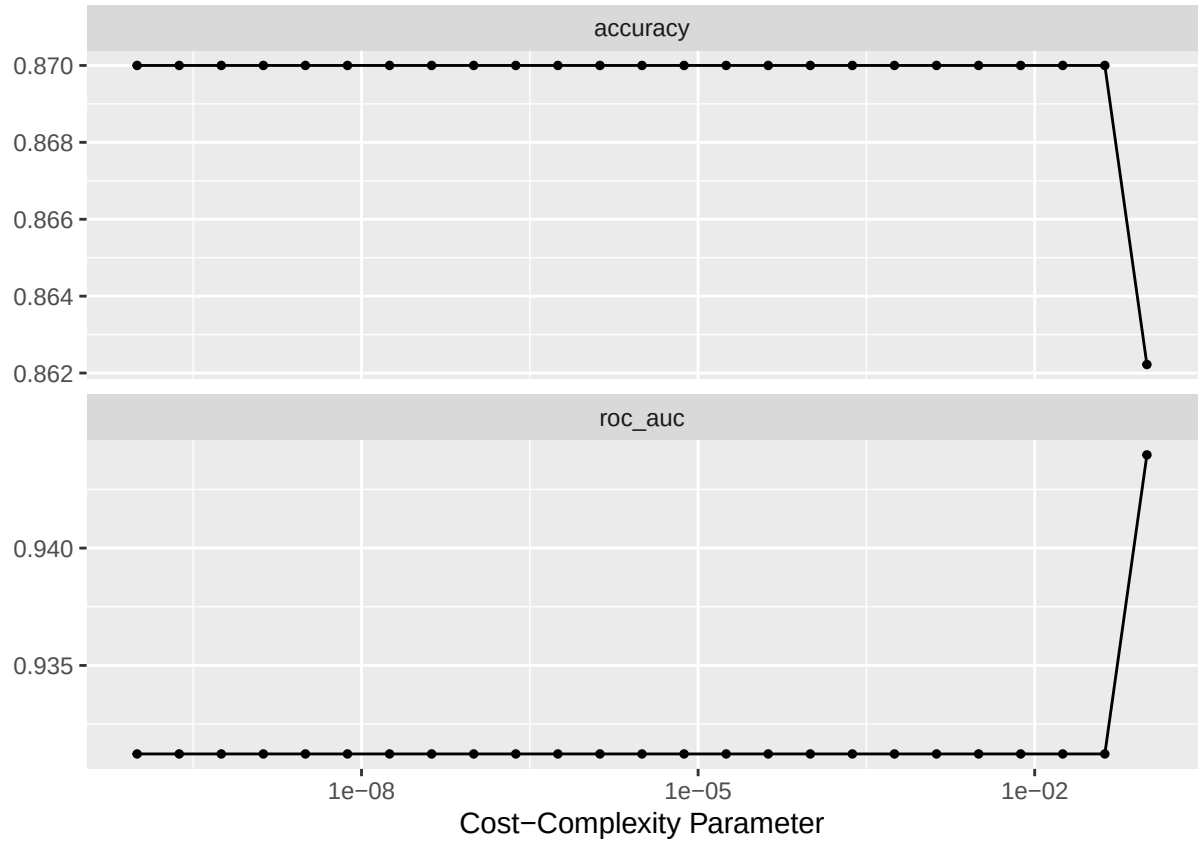
```
set.seed(12345)
tissueFolds <- vfold_cv(tissue.train.tbl, v = 10)
tissueGrid <- grid_regular(cost_complexity(), levels = 25)
```

```
tissueModel <- decision_tree(cost_complexity=tune()) %>%# maximal tree: cp = 0
  set_mode("classification") %>%
  set_engine("rpart")#
tissueRecipe <- recipe(tissue ~., data=tissue.train.tbl)
tissueWflow <- workflow() %>%
    add_recipe(tissueRecipe) %>%
    add_model(tissueModel)
#skip fitting for now
```

```
tissueRes <- tune_grid(tissueWflow,resamples = tissueFolds,grid = tissueGrid)
show_best(tissueRes)
```

```
## # A tibble: 5 x 7
##   cost_complexity .metric .estimator  mean     n std_err .config
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <fct>
## 1        1   e- 1 roc_auc hand_till  0.944    10  0.0115 Preprocessor1_Model25
## 2        1   e-10 roc_auc hand_till  0.931    10  0.0188 Preprocessor1_Model01
## 3        2.37e-10 roc_auc hand_till  0.931    10  0.0188 Preprocessor1_Model02
## 4        5.62e-10 roc_auc hand_till  0.931    10  0.0188 Preprocessor1_Model03
## 5        1.33e- 9 roc_auc hand_till  0.931    10  0.0188 Preprocessor1_Model04
```

```
autoplot(tissueRes)
```




```
(bestPenalty <- select_best(tissueRes,metric = "accuracy"))
```

```
## # A tibble: 1 x 2
##   cost_complexity .config
##             <dbl> <fct>
## 1    0.0000000001 Preprocessor1_Model01
```

```
tissue.final.wf <- finalize_workflow(tissueWflow,  bestPenalty)
tissue.final.fit <- fit(tissue.final.wf, tissue.train.tbl)

augment(tissue.final.fit, tissue.test.tbl) %>%
  accuracy(truth=tissue, estimate=.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(truth=tissue, estimate=.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
```

rmse is for regression accuracy is for classification

placenta misclassified as endometrium twice. kidney misclassified as endometrium twice. endometrium misclassified as kidney once. (5 total wrong)

accuracy on testing is 0.926

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.

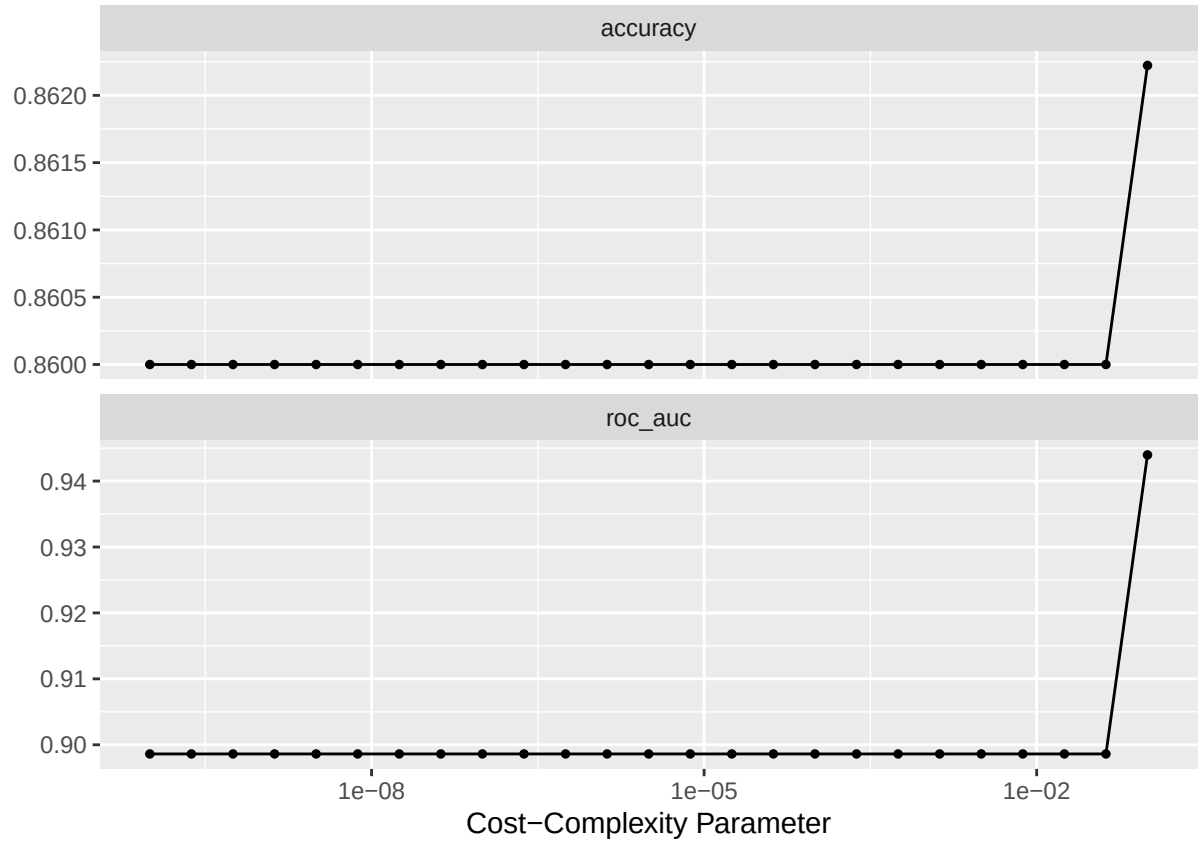
```
set.seed(12345)
tissueFolds <- vfold_cv(tissue.train.tbl, v = 10)
tissueGrid <- grid_regular(cost_complexity(), levels = 25)
```

```
tissueModel <- decision_tree(cost_complexity=tune(), min_n = 1) %>%# maximal tree: cp = 0
  set_mode("classification") %>%
  set_engine("rpart")#
tissueRecipe <- recipe(tissue ~., data=tissue.train.tbl)
tissueWflow <- workflow() %>%
    add_recipe(tissueRecipe) %>%
    add_model(tissueModel)
#skip fitting for now
```

```
tissueRes <- tune_grid(tissueWflow,resamples = tissueFolds,grid = tissueGrid)
show_best(tissueRes)
```

```
## # A tibble: 5 x 7
##   cost_complexity .metric .estimator  mean     n std_err .config
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <fct>
## 1        1   e- 1 roc_auc hand_till  0.944    10  0.0115 Preprocessor1_Model25
## 2        1   e-10 roc_auc hand_till  0.899    10  0.0224 Preprocessor1_Model01
## 3        2.37e-10 roc_auc hand_till  0.899    10  0.0224 Preprocessor1_Model02
## 4        5.62e-10 roc_auc hand_till  0.899    10  0.0224 Preprocessor1_Model03
## 5        1.33e- 9 roc_auc hand_till  0.899    10  0.0224 Preprocessor1_Model04
```

```
autoplot(tissueRes)
```




```
(bestPenalty <- select_best(tissueRes,metric = "accuracy"))
```

```
## # A tibble: 1 x 2
##   cost_complexity .config
##             <dbl> <fct>
## 1             0.1 Preprocessor1_Model25
```

```
tissue.final.wf <- finalize_workflow(tissueWflow,  bestPenalty)
tissue.final.fit <- fit(tissue.final.wf, tissue.train.tbl)

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

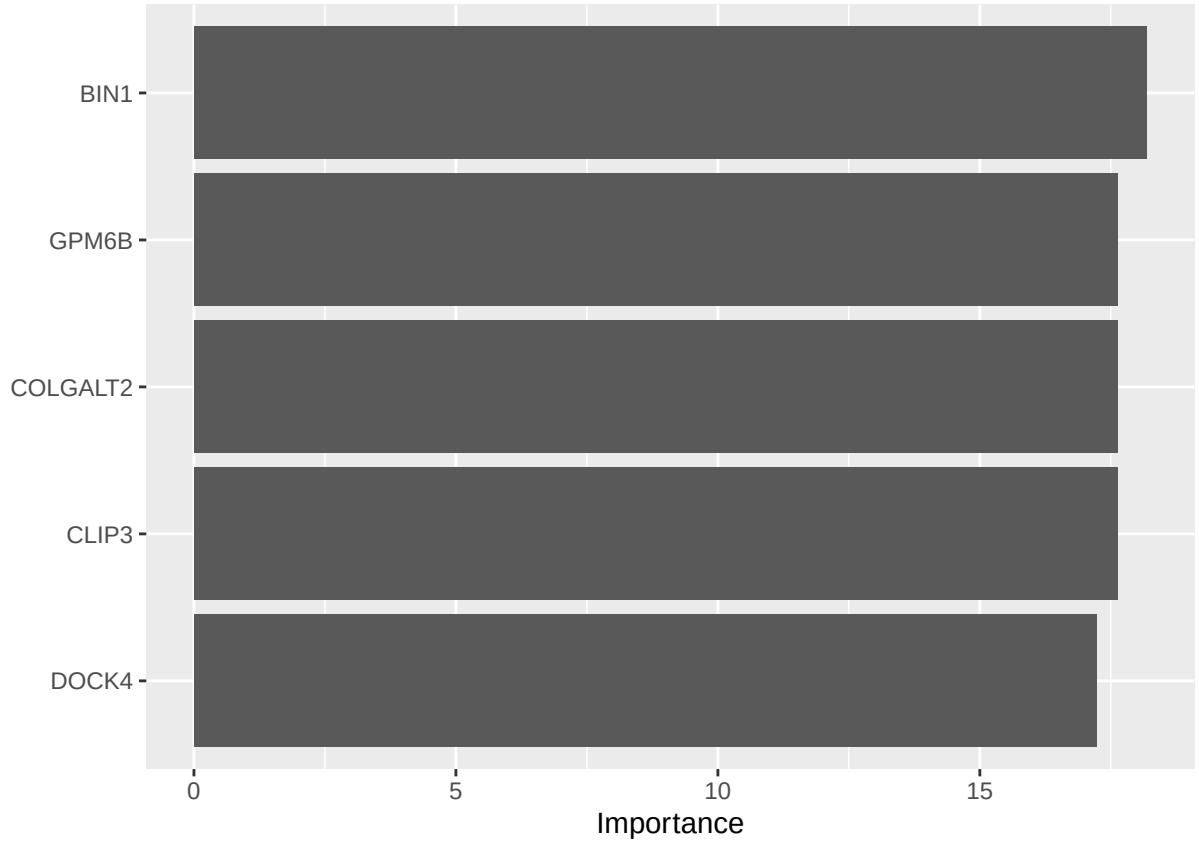
```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.884
```

```
augment(tissue.final.fit, tissue.test.tbl) %>%
  conf_mat(truth=tissue, estimate=.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           0           0      0     0         0
##   hippocampus          0     0           0          12      0     0         0
##   kidney               0     0           8           0     24     0         2
```

```
##   liver                0     0           0           0      0    12        0
##   placenta             0     0           0           0      0     0        0
```

```
extract_fit_parsnip(tissue.final.fit) %>%
  vip(num_features = 5L)
```



Top 5 genes, in order: BIN1, GPM6B, COLGALT2, CLIP3, DOCK4 (protein coding, bridging indicator 1, protein coding, protein coding, protein coding).

I looked at this website called genecards.org but I really do not know enough about biology to comment on whether the genes 'make biological sense'.

accuracy on testing 0.884. accuracy got worse.

placenta misclassified as kidney 2 times, endometrium misclassified as cerebellum once. endometrium misclassified as kidney 8 times. (11 total wrong)

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=501, min_n = 1) %>% #  mtry is number of predictors that each bag consi
  set_mode("classification") %>%
  set_engine("ranger",importance = "impurity")  # keep track of importance of variables so the importan

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

```
  add_model(ranger_spec)
```

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

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

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

```
augment(tissue.bag.model, tissue.test.tbl) %>%
  conf_mat(truth=tissue, estimate= .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           7           0      0     0         0
##   hippocampus          0     0           0          12      0     0         0
##   kidney               0     0           2           0     24     0         0
##   liver                0     0           0           0      0    12         0
##   placenta             0     0           0           0      0     0         2
```

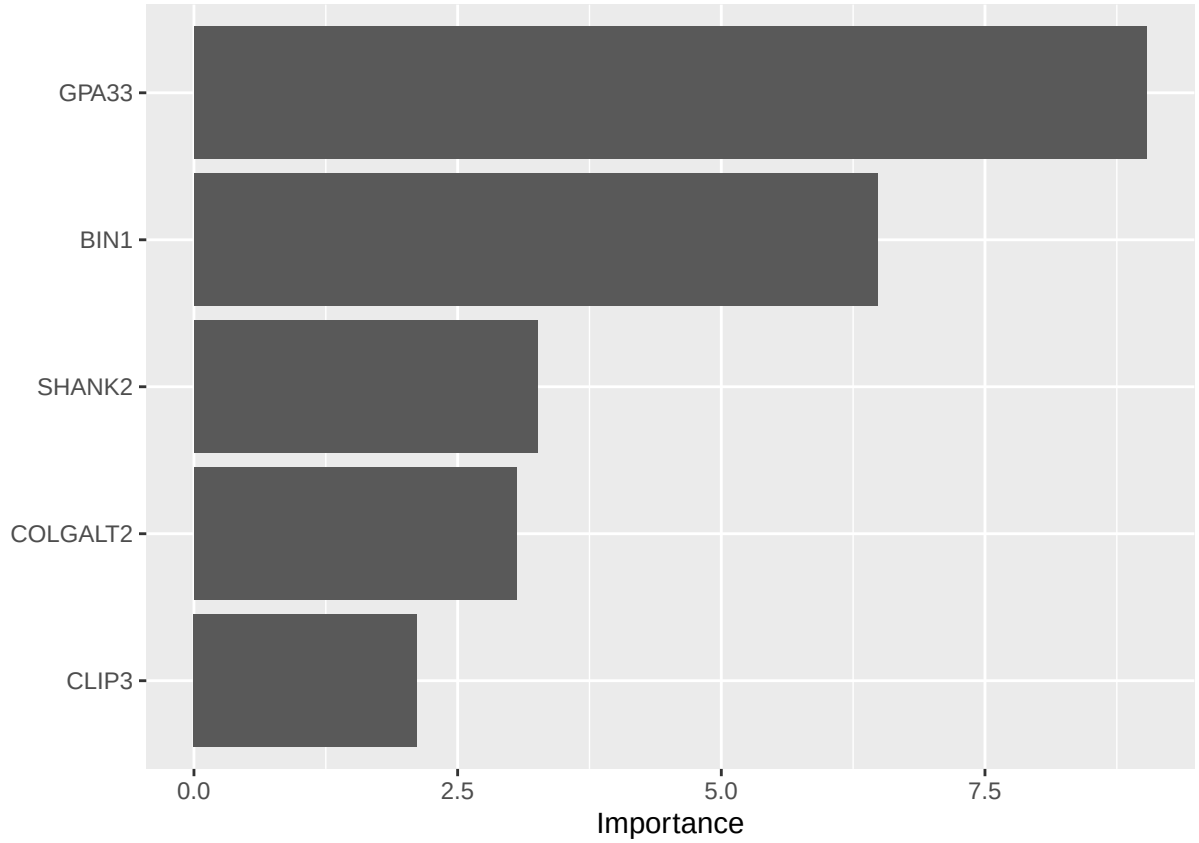
accuracy on testing 0.979. better from #3 and #4.

endometrium misclassified as kidney 2 times. placenta classified correctly this time.

This did not work: create_image_vip(tissue.bag.model) %>% plotImage()

this did:

```
extract_fit_parsnip(tissue.bag.model) %>%
  vip(num_features = 5L)
```



Top 5 genes: GPA33, BIN1, MYO1D, DOCK4, SHANK2

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

how is bagging model different from random forest? they both use random forest with the same parameters.

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

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

```
#recipe, mode + engine, workflow
ranger_recipe <-
  recipe(formula = tissue ~ ., data = tissue.train.tbl)
ranger_spec <-
  rand_forest(trees = 100, mtry = 22, min_n = 1) %>% #  mtry is number of predictors that each bag cons
  set_mode("classification") %>%
  set_engine("ranger",importance = "impurity")  # keep track of importance of variables so the importan
```

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

cost_complexity is not a parameter for this.

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

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

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

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

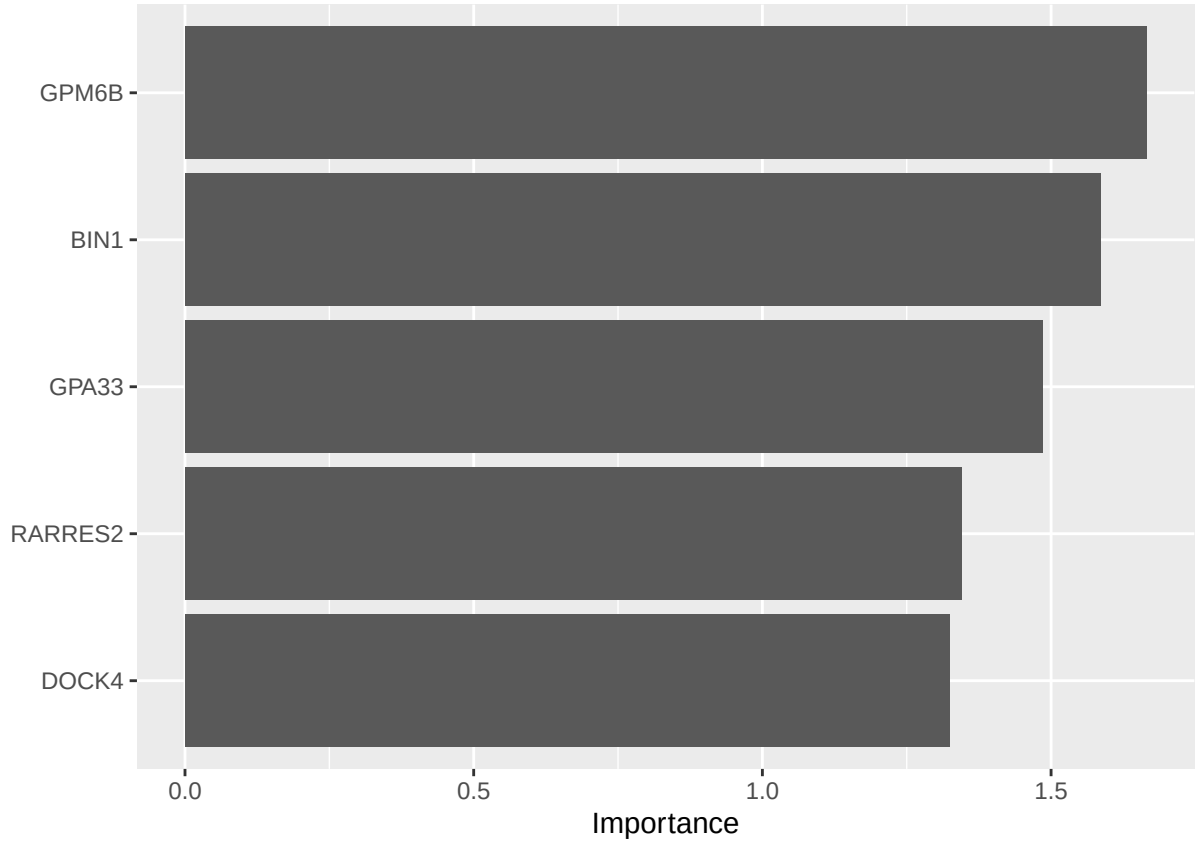
accuracy on testing is 1.00 with mtry = 22

all classified correctly, including placenta. improvement in accuracy from 3-5.

this did not work: {r fig.show="asis", echo=TRUE} create_image_vip(tissue.bag.model) %>% plotImage()

this did:

```
extract_fit_parsnip(tissue.bag.model) %>%
  vip(num_features = 5L)
```



Top 5 important genes: MYO1D, BIN1, TMEM184B, H2AFY, CLIP3

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(12345)
tissue.folds <- vfold_cv(tissue.train.tbl, v = 10)
tissue.grid <- grid_regular(mtry(range = c(50,200)), levels = 25)
```

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

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

```
#recipe, mode + engine, workflow
ranger_recipe <-
  recipe(formula = tissue ~ ., data = tissue.train.tbl)
```

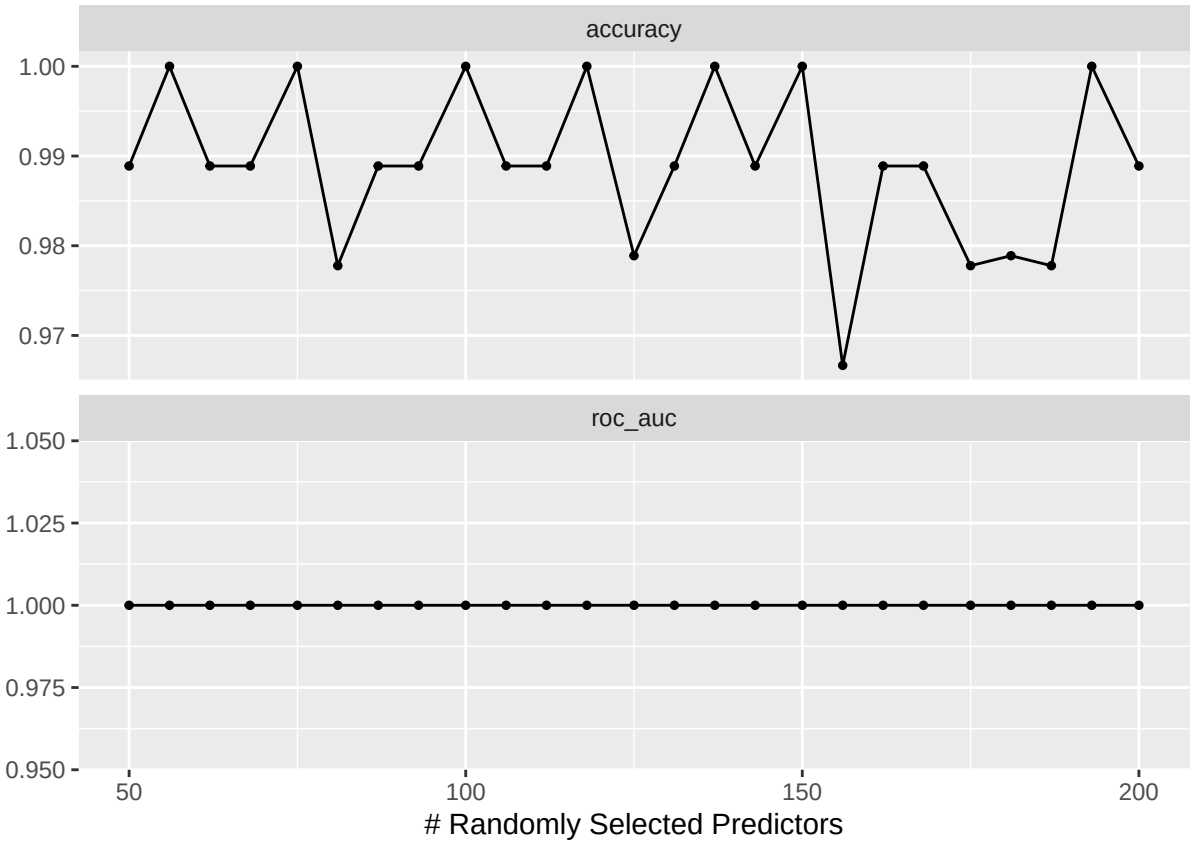
```
ranger_spec <-
  rand_forest(trees = 100, mtry = tune(), min_n = 1) %>% #  mtry is number of predictors that each bag
  set_mode("classification") %>%
  set_engine("ranger",importance = "impurity")  # keep track of importance of variables so the importan
ranger_workflow <-
  workflow() %>%
  add_recipe(ranger_recipe) %>%
  add_model(ranger_spec)
```

cost_complexity is not a parameter for this.

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

```
## # A tibble: 5 x 7
##    mtry .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <fct>
## 1    50 roc_auc hand_till      1    10       0 Preprocessor1_Model01
## 2    56 roc_auc hand_till      1    10       0 Preprocessor1_Model02
## 3    62 roc_auc hand_till      1    10       0 Preprocessor1_Model03
## 4    68 roc_auc hand_till      1    10       0 Preprocessor1_Model04
## 5    75 roc_auc hand_till      1    10       0 Preprocessor1_Model05
```

```
autoplot(tissue.res)
```




```
  # Select best penalty
(best.penalty <- select_best(tissue.res,metric = "accuracy"))
```

```
## # A tibble: 1 x 2
##    mtry .config
##   <int> <fct>
## 1    56 Preprocessor1_Model02
```

```
bag.final.wf <- finalize_workflow(ranger_workflow,  best.penalty)

tissue.final.fit <- fit(bag.final.wf, tissue.train.tbl)

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

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

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

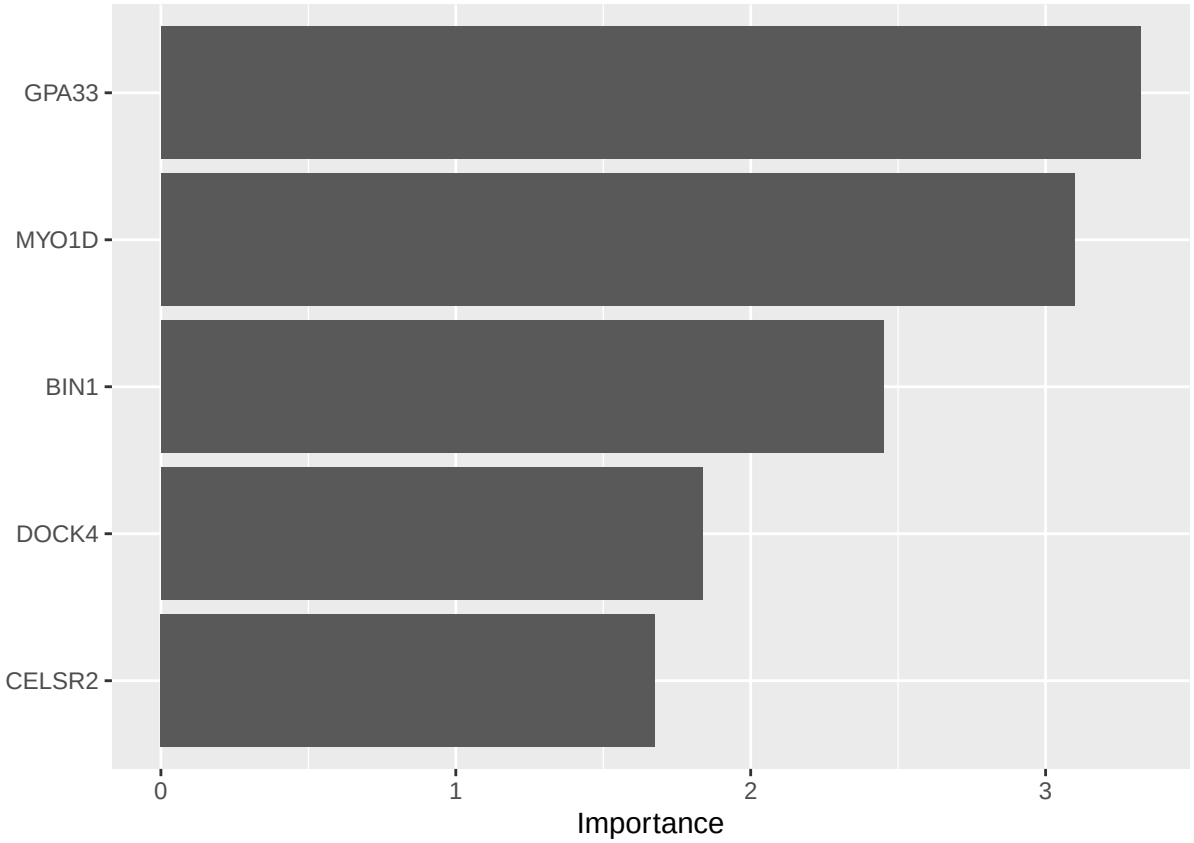
```
##              Truth
## Prediction    cerebellum colon endometrium hippocampus kidney liver placenta
##   cerebellum          19     0           0           0      0     0        0
##   colon                0    17           0           0      0     0        0
##   endometrium          0     0 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

```
#plot_model(tissue.final.fit, tissues.2008.tbl)   regression
```

accuracy on testing is 1.00

previous accuracy the same.

```
extract_fit_parsnip(tissue.final.fit) %>%
  vip(num_features = 5L)
```



Top 5 important genes: BIN1, GPA33, CEP55, COLGALT2, CLIP3.

top 5 variables in #5: GPA33, BIN1, MYO1D, DOCK4, SHANK2

top 5 variables in #4: BIN1, GPM6B, COLGALT2, CLIP3, DOCK4

BIN1 is common to all three, GPA33 is in two, COLGALT2 is in two, CLIP3 is in two, DOCK4 is in two, everything else appears once.

THey all have different top-2 genes.