MSCS 341: Final Exam

Andrew Noecker

5/17/2022

Exam 2 Guidelines

- You have until the scheduled day of your final to complete this exam. Submit it as a knitted markdown PDF document in the appropriate dropbox in moodle. If you submit the solution as an .Rmd you will be penalized by taking 20 points off your grade.
- You are to work completely independently on the exam. You are allowed to use your class notes, moodle, worksheets, homeworks, textbooks, plus the "Help" feature from Rstudio.
- You **are not** permitted to do web searches.
- For questions that ask for interpretations and explanations, usually no more than a sentence or two is needed for justification. Be thorough, but do not "brain dump".

PLEDGE: I pledge on my honor that I have neither received nor given assistance during this exam nor have I witnessed others receiving assistance, and I have followed the guidelines as described above.

PRINTED NAME: Andrew Noecker

◯ I have intentionally **not signed** the pledge.

Trends in superbowl commercials (10 points)

For this exercise we will be using the following dataset, which contains information about the superbowl commercials

```
superbowl.tbl <- read_csv("~/Mscs 341 S22/Class/Data/superbowl.csv")
```

The variable description of the dataset can be found at:

https://github.com/rfordatascience/tidytuesday/blob/master/data/2021/2021-03-02/readme.md#data-dictionary

In particular we are interested in the **characteristics** of the commercial which are encoded by the following columns:

- `animals`
- `celebrity`
- `danger`
- `funny`
- `patriotic`
- `show_product_quickly`

Use a linear model to establish how much each of these characteristics are changing through the years. What are your input and response variables and their types? According to your analysis, are commercials becoming less funny lately? Are they increasingly using more celebrities? Make sure to interpret the coefficients associated with each input variable of your linear model.

In order to create a singular model that examines how much these characteristics have changed, we can set time/year (a quantitative variable) as our response variable. Our input variables can then be animals, celebrity, danger, funny, patriotic, and show_product_quickly. These are all categorical, boolean/logical variables that are either true or false. We will now go through the process of modeling and then answer the other questions and interpret the variables.

```
sb.model <- linear_reg()%>%
  set_engine("lm")

sb.recipe <-
  recipe(year ~ animals + celebrity + danger + funny + patriotic + show_product_quickly, data = superbou
  step_dummy(all_nominal_predictors())


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


sb.fit <- fit(sb.wflow, superbowl.tbl)
```

```
glance(sb.fit)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic     p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>       <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.148         0.127  5.48      6.96 0.000000775     6  -767. 1550. 1578.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
tidy(sb.fit)
```

```
## # A tibble: 7 x 5
##   term                       estimate std.error statistic   p.value
##   <chr>                         <dbl>     <dbl>     <dbl>     <dbl>
## 1 (Intercept)               2011.         0.945 2129.     0
## 2 animalsTRUE                  0.0243     0.744    0.0327 0.974
## 3 celebrityTRUE                2.13       0.782    2.73   0.00680
## 4 dangerTRUE                   0.565      0.796    0.709  0.479
## 5 funnyTRUE                   -3.54       0.845   -4.19   0.0000395
## 6 patrioticTRUE                2.19       1.03     2.13   0.0342
## 7 show_product_quicklyTRUE     0.796      0.756    1.05   0.294
```

Using the output from the tidy command, we can examine the coefficients to determine the relationship between different commercial characteristics and how they have changed throughout the years.

Looking at the coefficient for 'funny', we see that it has a coefficient of -3.54 and a p-value of essentially 0. Thus, we can say that if all other characteristics of the commercial remain constant, then a funny commercial would have a year that is 3.54 years less than a commercial that is not funny. This suggests that commercials that come from earlier years were funnier, meaning commercials are less funny lately.

Next, we can look at the celebrity coefficient. Here we see a coefficient of 2.13, meaning that if all other characteristics are held constant, then the year of a commercial that has a celebrity is 2.13 years higher than a commercial without a celebrity. This suggests that celebrities are appearing more in commercials lately. Other coefficient interpretations:

- animals: The coefficient for animals is 0.0243. This means that if all other characteristics are held constant, then the year of a commercial with animals is around 0.0243 years higher than a commercial without animals. This suggests that animals are ever so slightly more common lately, but with a p-value

of 0.974, it seems like there's nothing significant here and animal presence in commercials is holding pretty constant.

- danger: The coefficient for danger is 0.565. Thus, if all other characteristics are held constant, then the year of a commercial that is danger is around 0.565 years greater than a commercial that is not considered dangerous. In other words, there's been a slight increase in dangerous commercials as of late, but the p-value of 0.479 suggests that the relationship is not strong and not significant.
- patriotic: The coefficient for patriotic is 2.19. This means that if all other characteristics of the commercial are held constant, then we'd expect the year of a dangerous commercial to be 2.19 years greater than a 'safe' commercial. Simplified, this suggests that commercials are increasingly portraying 'dangerous' scenes in recent years.
- show_product_quickly: Our final coefficient is 0.796 for show_product_quickly. Thus, if all other characteristics are held constant, then a commercial that shows its product quickly is expected to come from a year that is 0.796 greater/later than a commercial that does not show its product quickly. In other words, commercials are showing their products quickly with greater frequencies in recent years, although the p-value of 0.294 suggests the relationship is not statistically significant.

Decision boundaries (10 points)

For each of the decision boundaries depicted in Figure 1, justify whether or not they can come from one of the following classifiers:

1. Logistic regression
2. Linear Discriminant Analysis (LDA).
3. Quadratic Discriminant Analysis (QDA).
4. k-Nearest Neighbor with $k = 3$.
5. A decision tree of depth 2
6. A random forest with $B = 3$ trees, each of a maximum depth 3

Notice that each classifier only uses x_1 and x_2 as inputs and that a particular decision boundary can originate from more than one classifier

Left out image because there were some issues when knitting with the image file path

Picture 1

1. This picture can be logistic regression because logistic regression establishes a linear boundary between the two classes.
2. This picture certainly can be LDA because LDA creates a linear boundary, as specified in the name. We saw these boundaries when classifying two MNIST digits as well.
3. This picture cannot represent a QDA model - QDA uses quadratic equations and thus the decision boundaries are curves, not lines.
4. There would need to be a very specific dataset, but a KNN decision boundary can become a line - the data would have to be set up so that the nearest neighbors along either side of the line gave the correct classifier. In reality, it's unlikely that we'd see this decision boundary from knn = 3, but it's theoretically possible.
5. This cannot be a decision tree with depth 2 because decision trees have horizontal decision boundaries based on levels of the predictors, whereas in this picture we have a diagonal line.
6. Similar to the decision tree, we can't get this perfect diagonal line from random forests, especially if the maximal depth is only 3. If we had a very high depth we could get a boundary that approximates this line but with a depth of only 3 then we can't get this boundary using random forests.

Picture 2

1. This is not possible with logistic regression because logistic regression decision boundaries must be linear.
2. This is not possible with LDA because the decision boundaries of LDA must be linear, which is not the case here.
3. This is definitely possible with QDA, as QDA provides decision boundaries based on quadratic equations, which is what's shown in this graph.
4. This is possible with KNN, because we can have a decision boundary with k = 3 that makes any shaped line, although the data would need to be very specific to get a decision boundary this smooth.
5. For similar reasons to #1, this is not possible with a tree of depth = 2 because we don't have straight horizontal lines that represent tree nodes.
6. Similar to #1, we can't get a smooth curve from a random forest tree because of the tree nodes representing strict decisions, whereas the curve represents an infinite number of decisions along x1 and x2.

Picture 3

1. Definitely not possible with logistic regression, we can't get a decision boundary with that many breaks when doing logistic regression.
2. Not possible with LDA because LDA provides a linear decision boundary, of which this is not.
3. Not possible with QDA because it is not a quadratic decision boundary.
4. This is possible with KNN where k=3 because knn boundaries are rugged and very dependent on each individual data point, so this is a possible knn decision boundary.
5. Similar to #1 and #2, this is not possible with decision tree of depth=2 because trees require explicit straight boundaries, which is not at all what's shown here.
6. The random forest has a decision boundary similar to the simple decision tree, where each decision is based on a characteristic of x1 or x2, so the boundaries should be straight lines, which is not the case here.

Picture 4

1. Not possible because logistic regerssion has a decision boundary that is usually a line or curve, not a piecewise tree-like decision boundary.
2. LDA is not possible because LDA decision boundaries are linear when there are just two groups, and are not able to have piecewise decision boundaries.
3. This is also not possible for QDA because QDA decision boundaries are curves based on quadratic equations, not piecewise functions.
4. Similar to pictures 1 and 2, this is possible with KNN but the data would need to be very specific. KNN decision boundaries can be 'piecewise' in this way, but they often are not straight lines like this.
5. This is not a possible decision boundary for this tree because the depth is 2. The first branches could check if x1 is less than ~4 or not, but for values less than 4 another branch is needed which takes us beyond depth = 2. Thus, this is not possible with a tree of depth = 2.

6. This is possible with a random forest of depth = 3. The key part is depth = 3 because it allows for the third layer that decides between white/black for x1 values less than ~4. With depth = 3, we can check if x1 < ~4, then check if x2> 5, then check if x2 < ~1.8. This gives us our 3 layers that are possible with a random forest, and is why the decision tree of depth = 2 doesn't work.

Cats or dogs? (30 points)

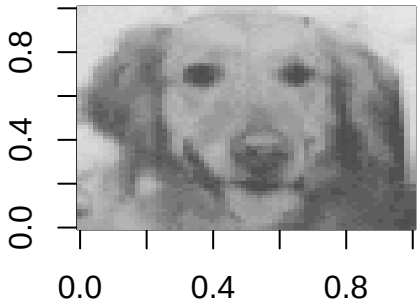
Consider the following dataset:

```
image.tbl <- read.csv("~/Mscs 341 S22/Class/Data/cats_dogs.csv")
```

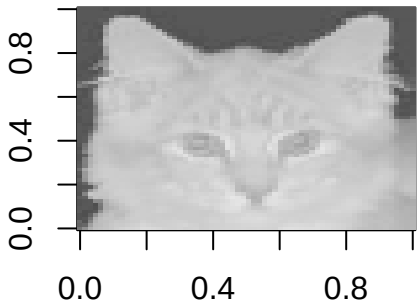
which contains the images of 99 cats and 99 dogs represented as 64 by 64 images. The first 4096 (64×64) columns correspond to the pixels of the image, and the column `type` contains information of whether the image corresponds to a `cat` or a `dog`.

The function `plotImage` allows us to show an image represented in a matrix. Notice that we can use it to plot the images in our dataset by selecting the appropriate row and columns and converting into a matrix (see the examples below)

```
plotImage <- function(dat,size=64){
  imag <- matrix(dat,nrow=size,byrow=T)
  image(imag,col=grey.colors(256))
}
#Plot the first image-dog
plotImage(as.matrix(image.tbl[1,1:4096]))
```




```
#Plot the 150th image-cat
plotImage(as.matrix(image.tbl[150,1:4096]))
```



Finally let's create training/testing datasets as well as 10-fold cross-validation dataset. Make sure to use these datasets in the following subsections.

```
set.seed(654321)
image.split <- initial_split(image.tbl, prop=0.6)
image.train.tbl <- training(image.split)
image.test.tbl <- testing(image.split)
images.folds <- vfold_cv(image.train.tbl, v = 10)
```

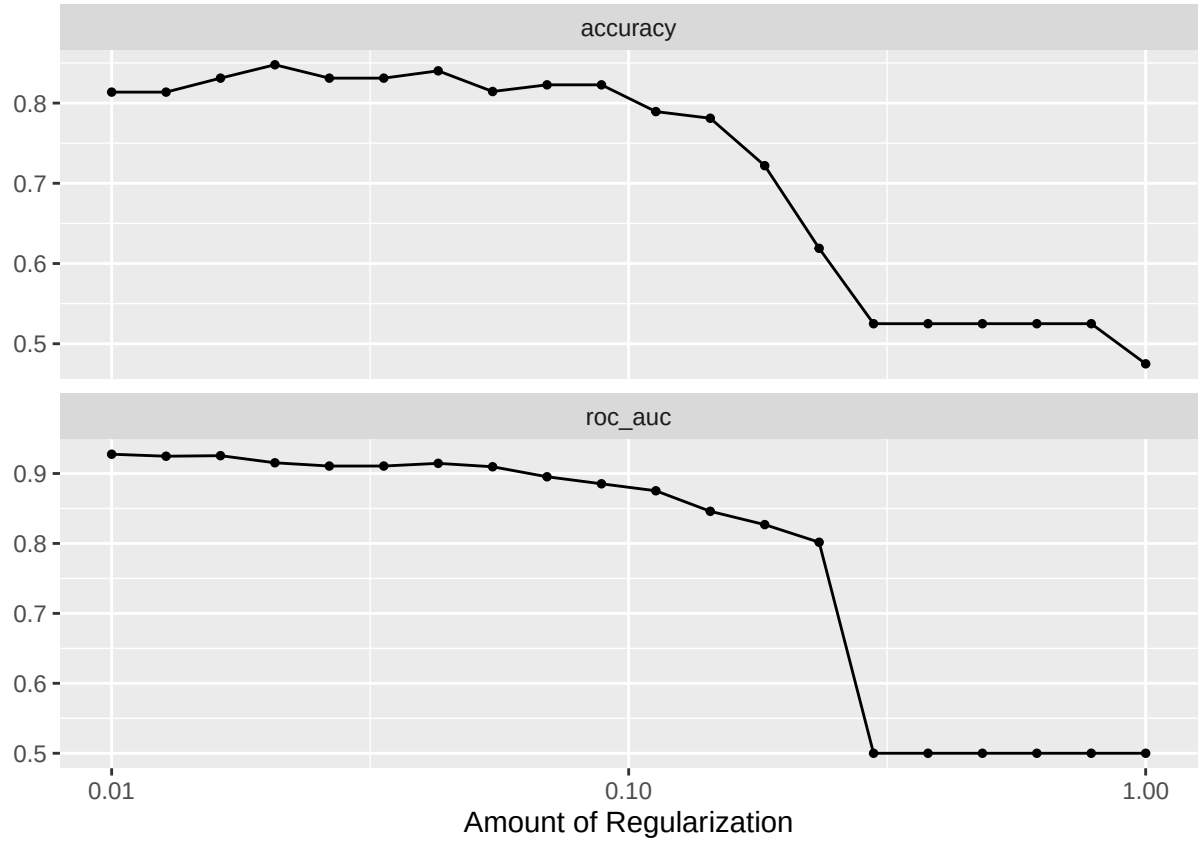
LASSO model (10 points)

Describe in your own words what a LASSO model is and what the penalty term (λ) is. Create a LASSO model that distinguishes between cats and dogs by selecting an optimal penalty by using 10 fold cross-validation. Establish the accuracy and confusion matrix of the model using your testing dataset. Show two incorrectly classified images and the probabilities of being a dog or a cat assigned by the model.

A LASSO model is a model that also serves as predictor selection. LASSO can be used both for linear regression and for classification (we show a classification example in this problem). The penalty term λ is the unique part of LASSO that allows it to serve as a variable selector as well. At its simplest, the penalty term shrinks the coefficients towards 0. This is ultimately done by minimizing an equation that involves the RSS and the sum of $|\beta_j|$ terms. What makes LASSO unique compared to its counterpart ridge regression is that when λ is large enough, then some variable coefficients actually become 0, whereas with ridge regression the coefficients only approach 0 (this is because ridge minimizes β_j^2 terms). By eliminating less important predictors, LASSO models are easier to interpret.

```
animal.model <-
  logistic_reg(mixture = 1, penalty=tune()) %>%
  set_mode("classification") %>%
  set_engine("glmnet")

animal.recipe <-
  recipe(formula = type ~ ., data = image.train.tbl) %>%
  step_normalize(all_predictors())

animal.wf <- workflow() %>%
  add_recipe(animal.recipe) %>%
```

```
  add_model(animal.model)

penalty.grid <-
  grid_regular(penalty(range = c(-2, 0)), levels = 20)

tune.res <- tune_grid(
  animal.wf,
  resamples = images.folds,
  grid = penalty.grid
)
autoplot(tune.res)
```




```
#Grabbing best penalty within one SE, making final model with that penalty
(best.penalty <- select_by_one_std_err(tune.res,
                                       metric = "accuracy",
                                       desc(penalty)))
```

```
## # A tibble: 1 x 9
##   penalty .metric  .estimator  mean     n std_err .config           .best .bound
##     <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <fct>             <dbl>  <dbl>
## 1  0.0428 accuracy binary     0.840    10  0.0225 Preprocessor1_Mo~ 0.848  0.831
```

```
animal.final.wf <- finalize_workflow(animal.wf, best.penalty)
animal.final.fit <- fit(animal.final.wf, data = image.train.tbl)

augment(animal.final.fit, new_data = image.test.tbl) %>%
  conf_mat(truth = type, estimate = .pred_class)
```

```
##           Truth
## Prediction cat dog
##        cat  32  15
##        dog   5  28
```

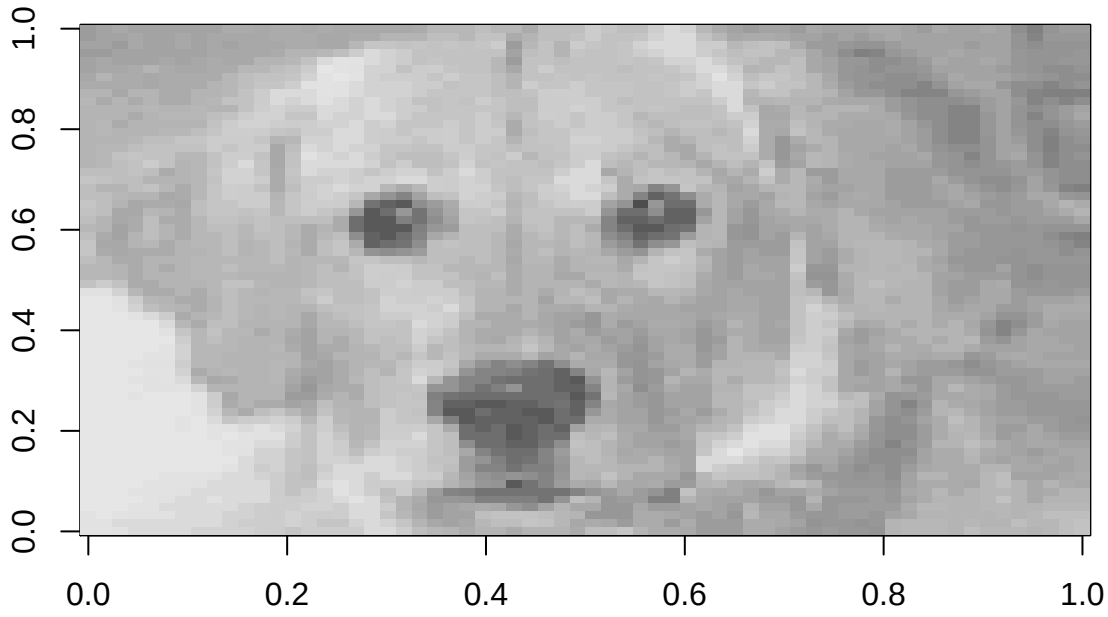
```
augment(animal.final.fit, new_data = image.test.tbl) %>%
  accuracy(truth = type, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary          0.75
```

```
#Accuracy of 0.75
```

We see an accuracy of 75%, and a confusion matrix where only 5 out of 37 cats are classified as dogs, but 15 of 43 dogs are classified as cats.

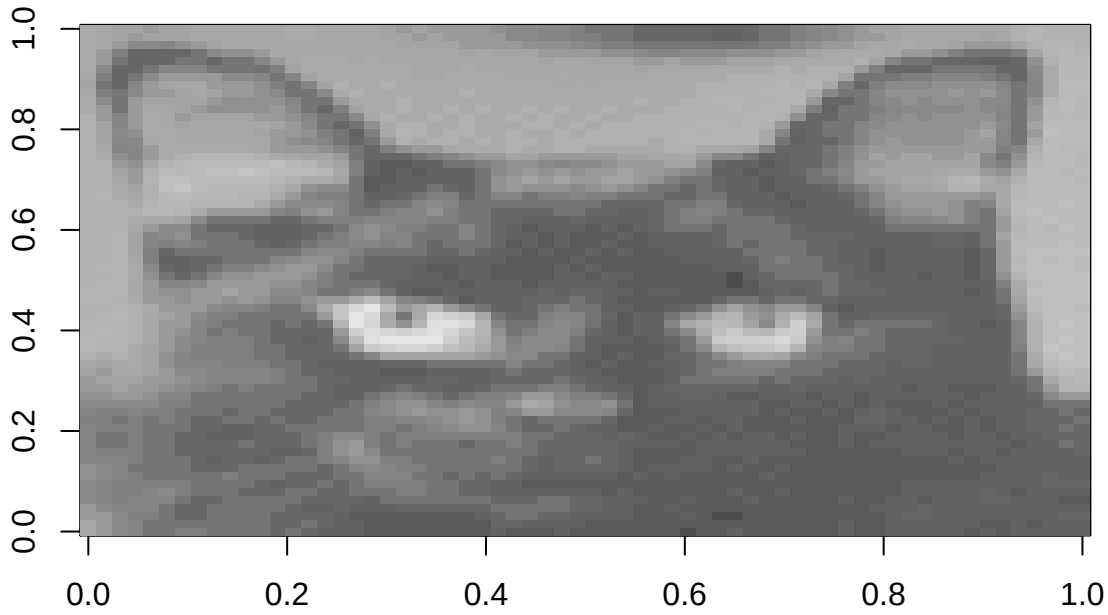
```
lasso.miss <- augment(animal.final.fit, new_data = image.test.tbl)%>%
  select(4097, 4098, 4099, 4100, 1:4096)%>%
  filter(type != .pred_class)

#Dog mistaken for a cat with corresponding probabilities
plotImage(as.matrix(lasso.miss[1,5:4100]))
```




```
(lasso.miss%>%select(1:4))[1,]
```

```
## # A tibble: 1 x 4
##   type  .pred_class .pred_cat .pred_dog
##   <fct> <fct>           <dbl>     <dbl>
## 1 dog   cat             0.589     0.411
```

```
#Cat mistaken for a dog with corresponding probabilities
plotImage(as.matrix(lasso.miss[16, 5:4100]))
```




```
(lasso.miss%>%select(1:4))[16,]
```

```
## # A tibble: 1 x 4
##   type  .pred_class .pred_cat .pred_dog
##   <fct> <fct>           <dbl>     <dbl>
## 1 cat   dog             0.459     0.541
```

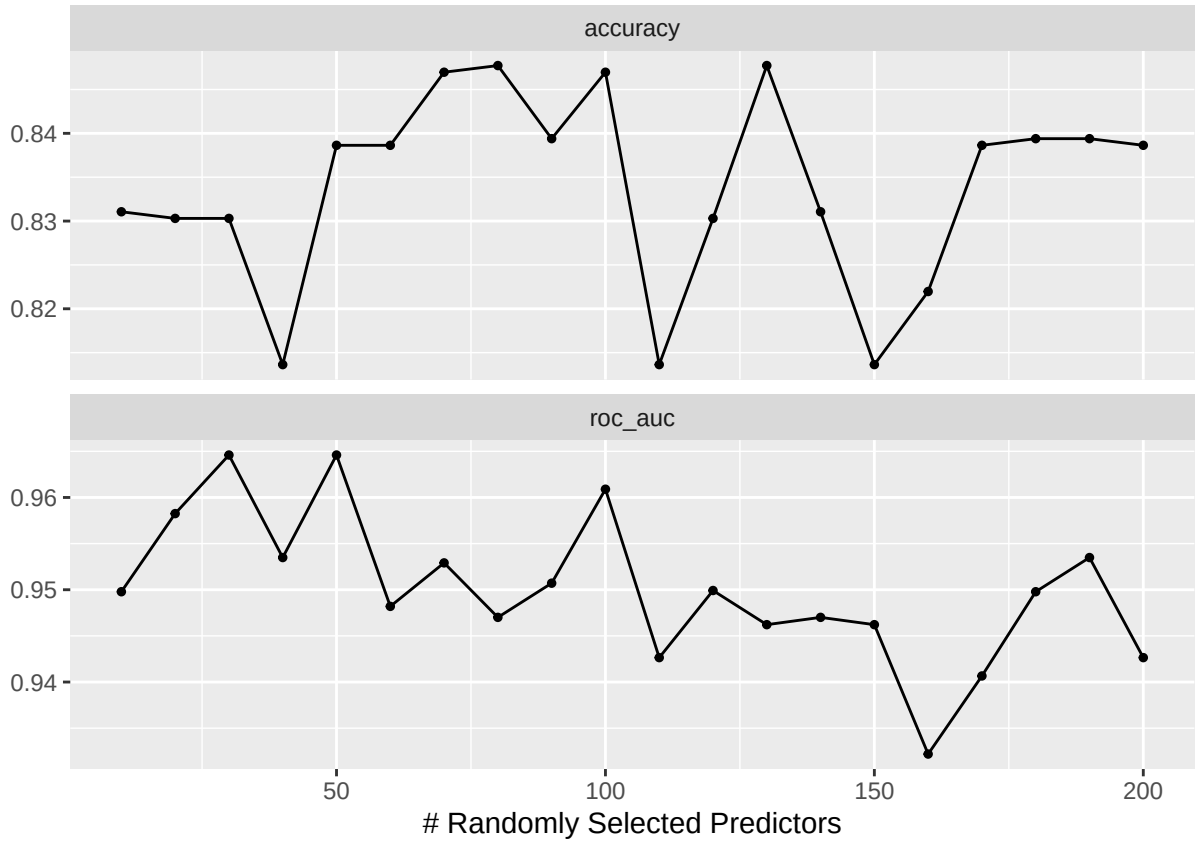
Random forest (10 points)

Describe in your own words what a random forest model is and what the parameter `mtry` refers to in the `ranger` implementation of random forests. Create a random forest model that distinguishes between cats and dogs by selecting an optimal value of `mtry` using 10 fold cross-validation. Establish the accuracy and confusion matrix of the model using your testing dataset. Show two incorrectly classified images and the probabilities of being a dog or a cat assigned by the model.

Random forests are a type of decision tree model that attempts to decorrelate maximal trees. When we try to build maximal trees with bagging (which uses all possible predictors), then certain predictors dominate across all the trees, leading to correlation between trees (something we want to avoid). So, a random forest model uses a random subset of predictors at each level of the tree, trying to ensure that different trees are

unrelated when training. The mtry parameter specifies how many predictors should be used at each level of the tree. It can be cross validated, but is often set to be $\sqrt{n}$ for classification trees and $\frac{n}{3}$ for regression trees.

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

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

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


#mtry is normally sqrt(n) = 64 in this case,
#so going to go from 10 to 200 just to be thorough in the cross validation
animals.grid <- tibble(mtry = seq(10, 200, by = 10))

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

autoplot(animals.res)
```



```
#Grabbing best penalty within one SE, making final model with that penalty
(best.penalty <- select_by_one_std_err(animals.res,
                                       metric = "accuracy",
                                       desc(mtry)))
```

```
## # A tibble: 1 x 9
##    mtry .metric  .estimator  mean     n std_err .config              .best .bound
##   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <fct>                <dbl>  <dbl>
## 1   200 accuracy binary     0.839    10  0.0233 Preprocessor1_Mode~ 0.848  0.818
```

```
animal.final.wf <- finalize_workflow(ranger_workflow, best.penalty)
animal.final.fit <- fit(animal.final.wf, data = image.train.tbl)

augment(animal.final.fit, new_data = image.test.tbl) %>%
  conf_mat(truth = type, estimate = .pred_class)
```

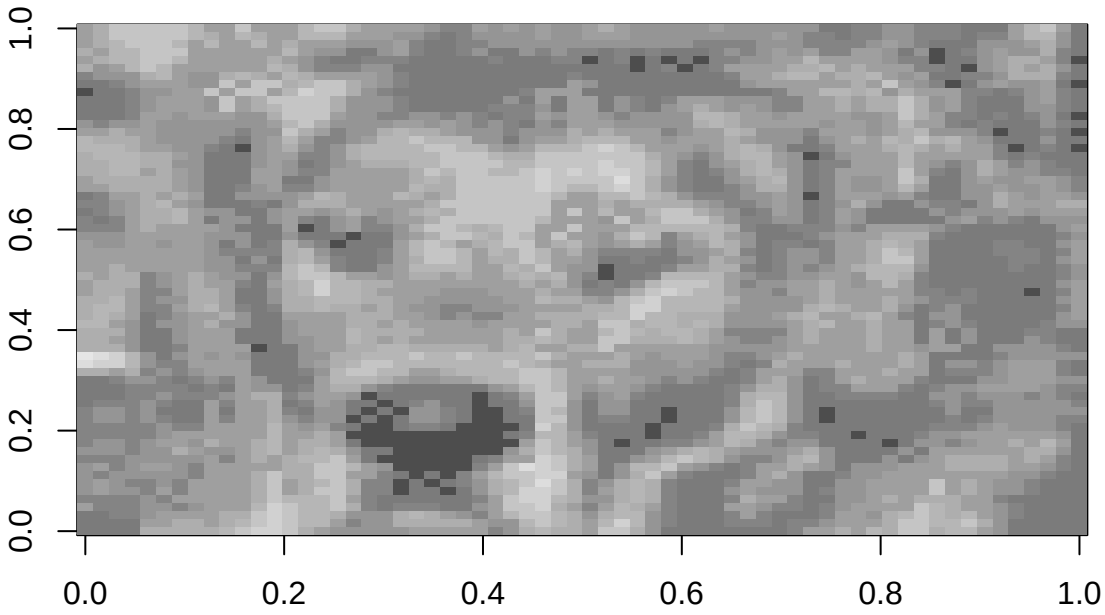
```
##           Truth
## Prediction cat dog
##        cat  34  12
##        dog   3  31
```

```
augment(animal.final.fit, new_data = image.test.tbl) %>%
  accuracy(truth = type, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.812
```

With forests, we see an accuracy improvement up to 81.2%. When looking at the confusion matrix, we see that only 3 out of 37 cats are misclassified, while only 12 of 43 dogs are misclassified, an improvement in both categories from the LASSO model.

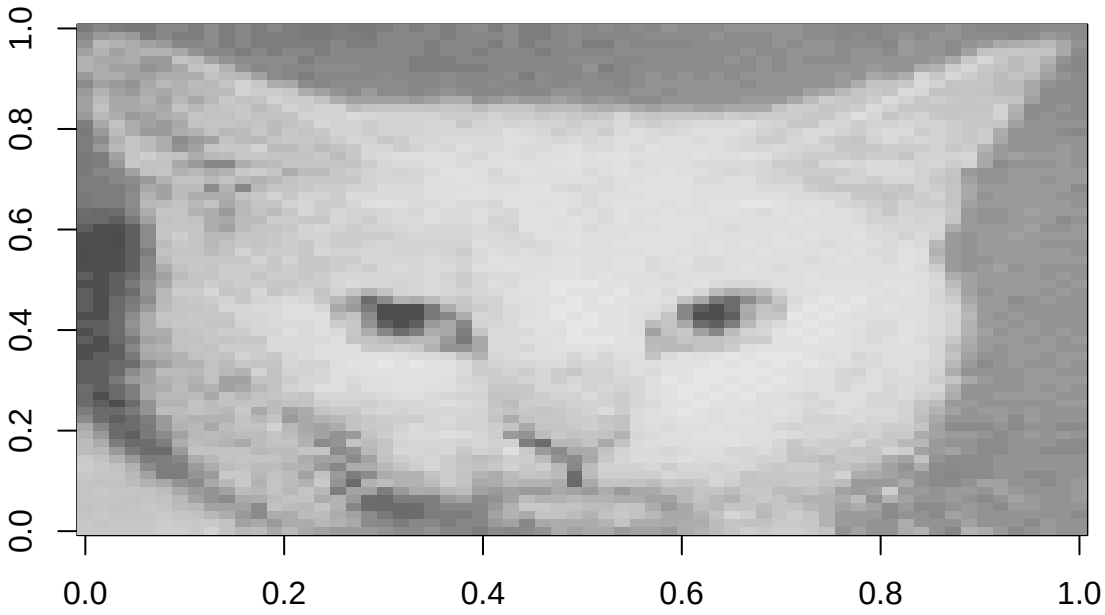
```
forest.miss <- augment(animal.final.fit, new_data = image.test.tbl)%>%
  select(4097, 4098, 4099, 4100, 1:4096)%>%
  filter(type != .pred_class)

#Dog mistaken for a cat with corresponding probabilities
plotImage(as.matrix(forest.miss[1,5:4100]))
```




```
(forest.miss%>%select(1:4))[1,]
```

```
## # A tibble: 1 x 4
##   type  .pred_class .pred_cat .pred_dog
##   <fct> <fct>           <dbl>     <dbl>
## 1 dog   cat             0.549     0.451
```

```
#Cat mistaken for a dog with corresponding probabilities
plotImage(as.matrix(forest.miss[14, 5:4100]))
```




```
(forest.miss%>%select(1:4))[14,]
```

```
## # A tibble: 1 x 4
##   type  .pred_class .pred_cat .pred_dog
##   <fct> <fct>           <dbl>     <dbl>
## 1 cat   dog             0.481     0.519
```

Boosting (10 points)

Describe in your own words what a boosting model is and what the learning rate is in a boosting model. Create a boosting model that distinguishes between cats and dogs by selecting an optimal learning rate using 10 fold cross-validation. Establish the accuracy and confusion matrix of the model. Show two incorrectly classified images and the probabilities of being a dog or a cat assigned by the model.

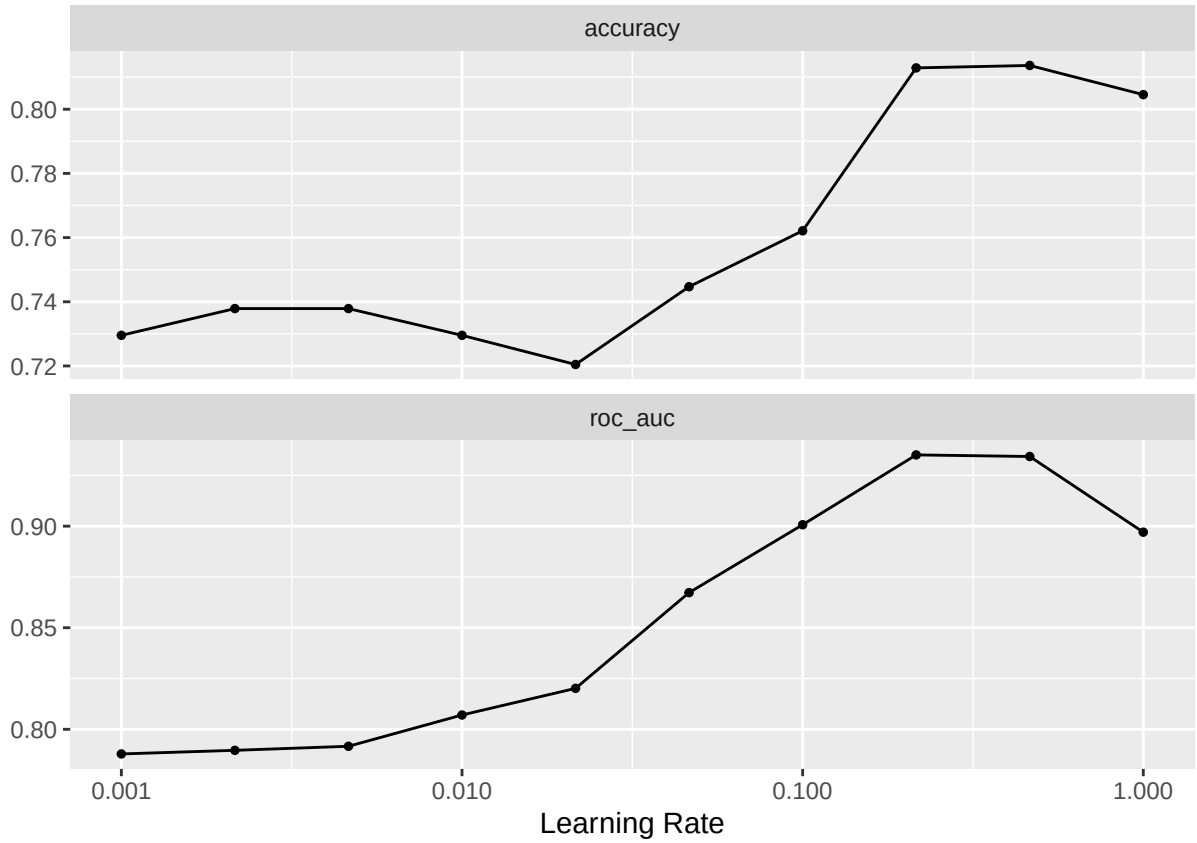
Boosting is a process that slowly builds up a full-fledged model. In its simplest form, boosting involves building a bunch of small, simple models and then using those to slowly build up a final model. It can be used with any type of model, but is often employed with decision trees. It starts with the weakest model, and then each model is built on the residuals of the previous model, thus building the final tree in a sequential or recursive manner. With trees, the depth of the 'weak learner' or small trees is often 1 or 2. The learning rate is a small positive number that controls the rate at which boosting learns (and thus the rate at which the full model/tree is built). If the learning rate is very small, then we will need to build a lot of small trees to build the final model.

```
xgboost_recipe <-
  recipe(formula = type ~ ., data = image.train.tbl) %>%
  step_zv(all_predictors())
```

```
xgboost_spec <-
  boost_tree(learn_rate = tune()) %>%
  set_mode("classification") %>%
  set_engine("xgboost")

(lr.grid <- grid_regular(learn_rate(range=c(-3,0)), levels = 10))
```

```
## # A tibble: 10 x 1
##    learn_rate
##         <dbl>
##  1    0.001
##  2    0.00215
##  3    0.00464
##  4    0.01
##  5    0.0215
##  6    0.0464
##  7    0.1
##  8    0.215
##  9    0.464
## 10    1
```

```
xgboost_workflow <-
  workflow() %>%
  add_recipe(xgboost_recipe) %>%
  add_model(xgboost_spec)

xgboost_tune <-
  tune_grid(xgboost_workflow,
            resamples = images.folds,
            grid = lr.grid)

autoplot(xgboost_tune)
```




```
#Grabbing best penalty within one SE, making final model with that penalty
(best.se.penalty <- select_by_one_std_err(xgboost_tune,
                                          metric = "accuracy",
                                          desc(learn_rate)))
```

```
## # A tibble: 1 x 9
##   learn_rate .metric  .estimator  mean     n std_err .config        .best .bound
##        <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <fct>          <dbl>  <dbl>
## 1          1 accuracy binary     0.805    10  0.0223 Preprocessor1~ 0.814  0.790
```

```
animal.final.wf <- finalize_workflow(xgboost_workflow, best.se.penalty)
animal.final.fit <- fit(animal.final.wf, data = image.train.tbl)

augment(animal.final.fit, new_data = image.test.tbl) %>%
  conf_mat(truth = type, estimate = .pred_class)
```

```
##           Truth
## Prediction cat dog
##        cat  30  12
##        dog   7  31
```

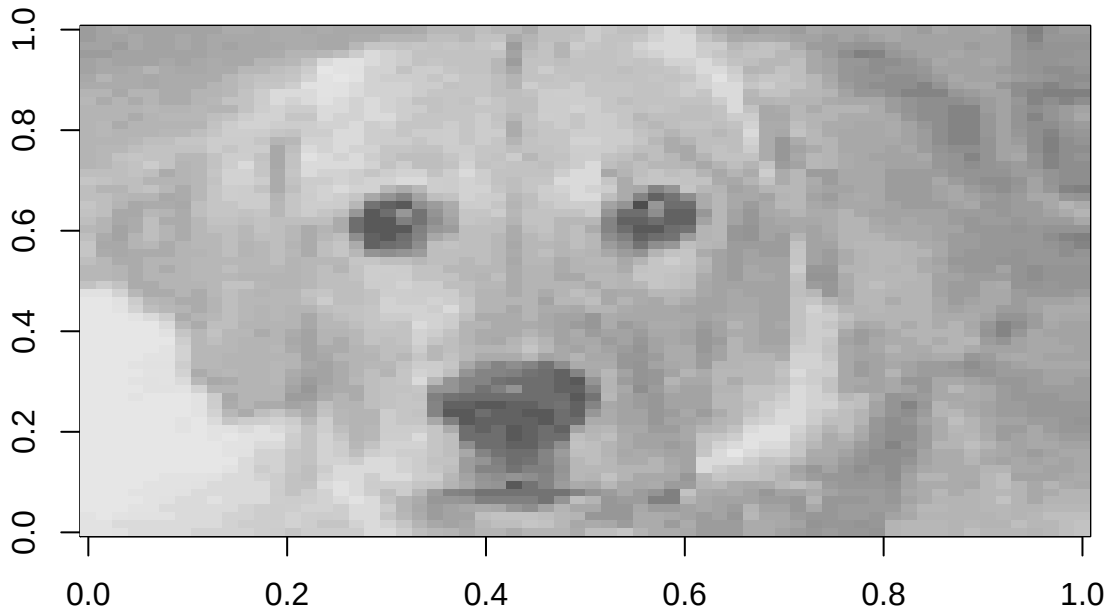
```
augment(animal.final.fit, new_data = image.test.tbl) %>%
  accuracy(truth = type, estimate = .pred_class)
```

```
## # A tibble: 1 x 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.762
```

```
#Accuracy of 0.762
```

When we employ boosting, we see an accuracy of 76.2%, a decrease from forests but a slight increase from LASSO. The confusion matrix shows that 7 out of 37 cats were misclassified, and 12 of 43 dogs were misclassified.

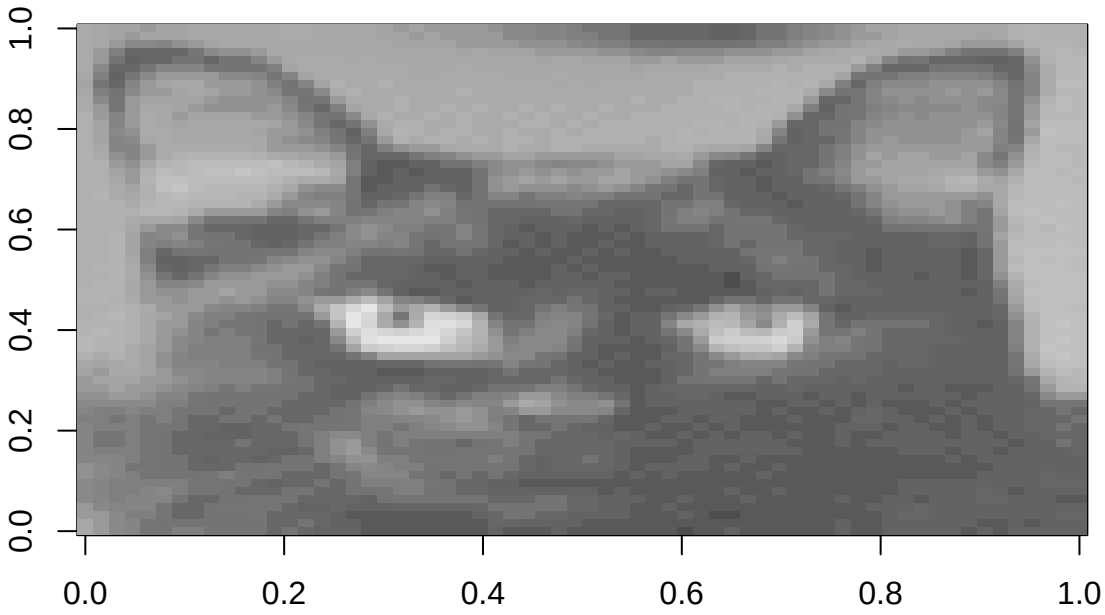
```
boost.miss <- augment(animal.final.fit, new_data = image.test.tbl)%>%
  select(4097, 4098, 4099, 4100, 1:4096)%>%
  filter(type != .pred_class)

#Dog mistaken for a cat with corresponding probabilities
plotImage(as.matrix(boost.miss[1,5:4100]))
```




```
(boost.miss%>%select(1:4))[1,]
```

```
## # A tibble: 1 x 4
##   type  .pred_class .pred_cat .pred_dog
##   <fct> <fct>           <dbl>     <dbl>
## 1 dog   cat             0.712     0.288
```

```
#Cat mistaken for a dog with corresponding probabilities
plotImage(as.matrix(boost.miss[13, 5:4100]))
```




```
(boost.miss%>%select(1:4))[13,]
```

```
## # A tibble: 1 x 4
##   type  .pred_class .pred_cat .pred_dog
##   <fct> <fct>           <dbl>     <dbl>
## 1 cat   dog             0.189     0.811
```