Predicting Stellar Classifications

Using Machine Learning Models to Predict the Classifications of Stellar Phenomena

Kayla Katakis

UCSB Winter 2023

Introduction

The goal of this project is to build the optimal machine learning model to predict the classification of different stellar phenomena. These phenomena can be categorized as stars, galaxies or quasars, and we will be using data from the Sloan Digital Sky Survey (SDSS) that has been compiled into a (data set (https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17?resource=download)) by user @fedesoriano on Kaggle. We will be using multiple techniques to find the most accurate model for this multi-class classification. Before we get into our model-building and assessment, let's take a look at our data and gather some background information.

What are Stars, Galaxies, and Quasars?

Arguably, we all know what stars and galaxies are, but quasars may be a bit more complex and unknown. But, for sake of clarity, here is a bit of background on the phenomena we wish to predict.

STARS - Gaseous spheres that reflect light, are typically made of hydrogen and helium, and are held together by their own gravity

GALAXIES- Collections of up to billions of stars, planets, solar systems, and miscellaneous dusts and gases that are held together by gravity. They are typically elliptical or spiral, with some that are irregularly shaped.

QUASARS- Supermassive black holes that feed on gasses at the center of some distant galaxies.They typically have jets beaming out radiation from two sides, and are extremely luminous. They are also known as quasi-stellar objects, or QSO for short!

Fun Fact: Some quasars can admit more than 100 trillion times (https://www.universetoday.com/119126/could-the-milky-way-become-a-quasar/#:~:text=According%20to%20New%20York%20University,the%20intensity%20we%20get%20from) the energy of the Sun!

Project Outline

Following this brief introduction, there is still a lot of work to be done before we can fit our models. Of course, we first need to read in the dataset. We will then clean it up, see if any variables need to be altered, and deal with any missing values as needed. Some of the feature variables in this dataset may not be required or helpful for the classifications, so these will be removed. After tidying up our data, we will perform some exploratory analysis to look at any correlations or interactions between predictors as well as the distribution of our outcome variable and any other interesting trends that come up. Then we get to the fun stuff! After our exploratory analysis, we can finally split our data into training/testing sets and set up a 10-fold cross validation to assess the performance of our training models. For this project, we will implement the following models:

- Elastic Net Logistic Regression
- Linear Discriminant Analysis (LDA)
- K Nearest Neighbors (KNN)
- Random Forest

These models will be fitted to our training set and assessed across all folds to determine the best performing model and the parameters to be used if said model requires them. Finally, we fit our best model to the testing set and see just how well we can predict the classification of stellar phenomena!

\(~\)

Understanding our Data

Before modeling, we have to take a thorough look at our data, clean up any missing values or variables, and make sure each predictor variable is both relevant to our outcome and in the proper format. For example, this dataset contains a couple of variables that simply identify the observation and may not necessarily impact its classification, so we will not need those variables for our models. For clarity, our outcome, or response variable, will be `class` , with all others acting as explanatory variables.

Loading and Cleaning Data

Here, we are loading the dataset from its csv format and turning it into a dataframe. Here, we will also turn our outcome variable, `class` into a factor for easier assessment down the road.

```
stellar_data <- read_csv('data/star_classification.csv',
                         show_col_types = FALSE) #reading in csv

stellar_data$class <- factor(stellar_data$class,
                             labels = c('GALAXY', 'QSO', 'STAR'))
```

Currently, there are 100,000 observations across 18 variables, one of which is our outcome variable. Let's first see if any of our variables are irrelevant to our classification.

```
colnames(stellar_data)
```

```
##  [1] "obj_ID"      "alpha"       "delta"       "u"           "g"
##  [6] "r"           "i"           "z"           "run_ID"      "rerun_ID"
## [11] "cam_col"     "field_ID"    "spec_obj_ID" "class"       "redshift"
## [16] "plate"       "MJD"         "fiber_ID"
```

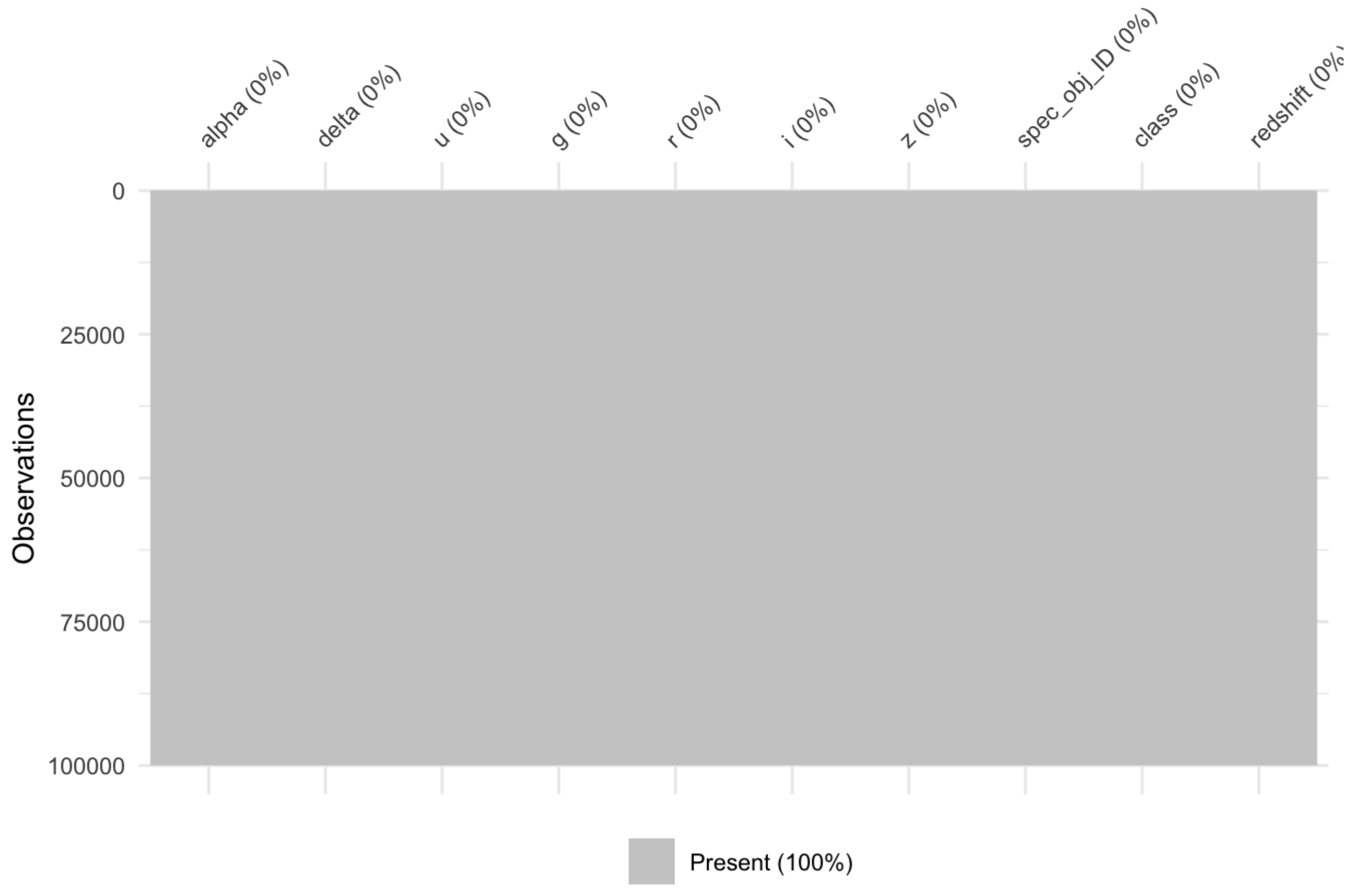
At first glance, a couple of columns that stood out as potentially unnecessary were any of the ID variables except for `spec_obj_ID` . All of the variables ending in ID have to do with how the observation was recorded as opposed to features of the observation that may impact its classification. As such, I will remove them from the dataframe for sake of simplicity. I chose to keep `spec_obj_ID` because this ID is unique to each object, meaning that two observations with the same value for the variable will have the same classification. So, it may be useful to keep this as a predictor of `class` , but we won't know for sure until later! I am also going to remove `plate` ,the plate ID in SDSS, `cam_col` , the camera column, and `MJD` , the modified Julian date on which the observation was recorded, for the same reason as the ID variables.

```
stellar_data <- subset(stellar_data, select = -c(obj_ID, run_ID, rerun_ID,
                                                 cam_col, field_ID, MJD,
                                                 fiber_ID,plate))
head(stellar_data)
```

```
## # A tibble: 6 × 10
##   alpha  delta     u     g     r     i     z spec_obj_ID class  redshift
##   <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>       <dbl> <fct>     <dbl>
## 1  136. 32.5    23.9  22.3  20.4  19.2  18.8     6.54e18 GALAXY    0.635
## 2  145. 31.3    24.8  22.8  22.6  21.2  21.6     1.18e19 GALAXY    0.779
## 3  142. 35.6    25.3  22.7  20.6  19.3  18.9     5.15e18 GALAXY    0.644
## 4  339. -0.403  22.1  23.8  21.6  20.5  19.3     1.03e19 GALAXY    0.932
## 5  345. 21.2    19.4  17.6  16.5  16.0  15.5     6.89e18 GALAXY    0.116
## 6  341. 20.6    23.5  23.3  21.3  20.3  19.5     5.66e18 QSO       1.42
```

Now, we can check for any missing data and handle it accordingly.

```
library(visdat)
vis_miss(stellar_data, warn_large_data = FALSE)
```



There is no missing data in our dataset, stellar! Now that our data is nice and tidy, we can go ahead and download our processed dataset and move on to the descriptive and visual analysis!

Variable Description

Now that we've narrowed down our data to only the important variables and made sure there is no missing data, it's time to look at what we're really dealing with. We've already discussed the `class` variable and what each classification represents. Now, here is the selection of 9 predictor variables that we will be working with in our models. The term J2000 epoch (https://en.wikipedia.org/wiki/Epoch_(astronomy)) is in reference to a time close to noon GMT on January 1, 2000 as part of a coordinate system often used by astronomers.

- `alpha` : The Right Ascension angle (at J2000 epoch). This is the angular distance along the celestial equator from the Sun. The celestial equivalent of longitude!
- `delta` : Declination angle (at J2000 epoch). Similar to ascension, but measured on a north-south axis. The celestial equivalent of latitude!
- `u` : Ultraviolet filter in the photometric system
- `g` : Green filter in the photometric system

- `r` : Red filter in the photometric system
- `i` : Near infrared filter in the photometric system
- `z` : Infrared filter in the photometric system
- `spec_obj_ID` : A unique ID for spectroscopic objects (2 objects with the same ID must share the same class)
- `redshift` : Redshift value based on increase in wavelength

These explanatory variables are all numeric and represent common qualities used by astronomers to identify spectral objects. If ascension and declination seem confusing, make sure to keep in mind that Earth rotates on a tilt!

\(~\)

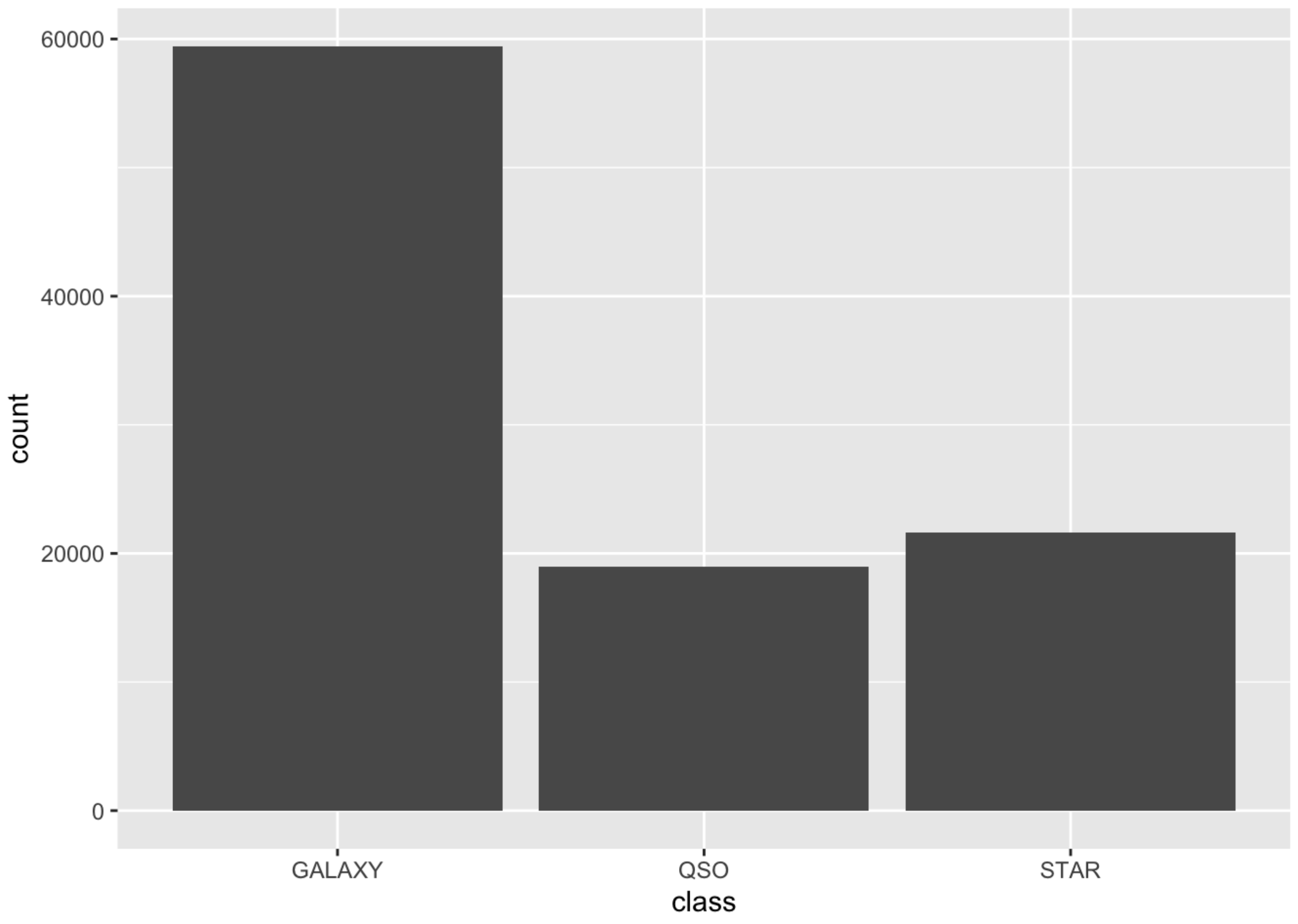
Exploring our Data Visually

Now that our data is organized, and we understand what we're working with, it's time to make some initial observations and investigate the relationships between our explanatory variables and how they affect our classification.

Response Proportions

First, let's look at how our stellar classifications are distributed.

```
ggplot(stellar_data, aes(x = class)) + geom_bar()
```

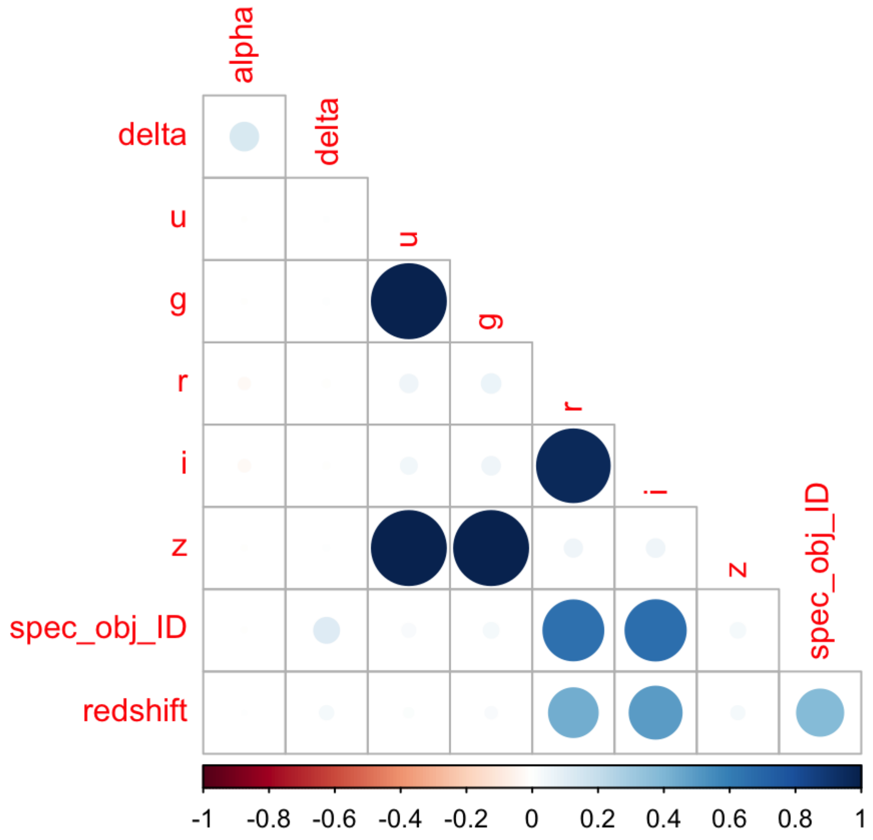


We can see that galaxy has about 3 times more representation than quasars and stars, which may needed to be adjusted for in our models by using `step_upsample()` to even out the ratios.

Predictor Correlations

Now, let’s take a look at how our predictors are correlated with one another to see if there are any interactions.

```
stellar_data %>% select(-class) %>% cor() %>%
  corrplot(type = 'lower', diag = FALSE)
```



Here, we see perfect correlations between `u` and `g`, `g` and `z`, and `z` and `u`, indicating that ultraviolet, green, and infrared filters are interrelated and have similar effects on the classification. Additionally, `r` and `i` have a correlation of 1.0, which makes sense as 'red' and 'infrared' have very similar wavelength being next to each other on the electromagnetic spectrum spectrum. `spec_obj_ID` and `redshift` are also positively correlated with both `r` and `i`, and, while `redshift` makes sense for these variables that describe the aspects of the observations concerning "redness", we're still not quite sure of `spec_obj_ID`'s role in classification.

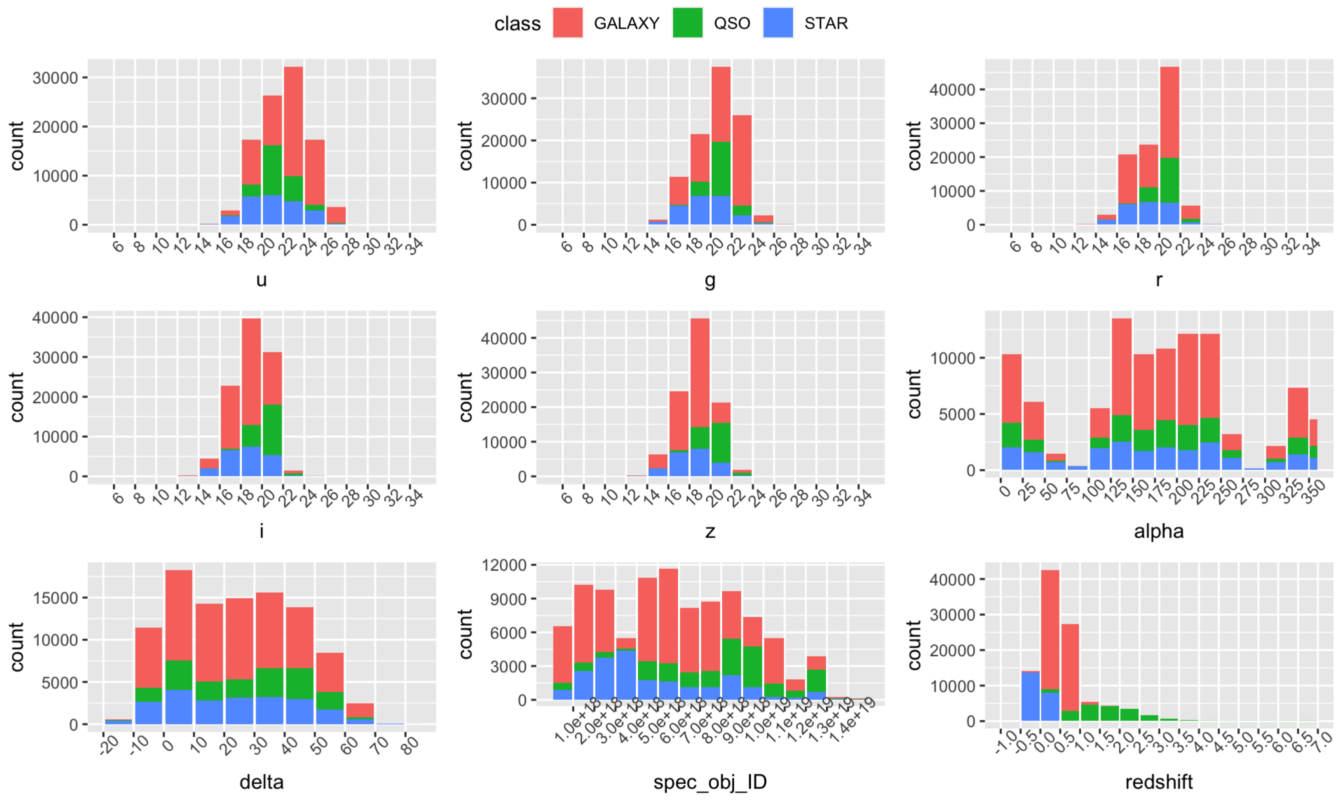
Relationships Between Classes and Predictors

Now that we can see which predictors are correlated with each other, we can take a look at how our classifications are distributed within each predictor. We can see which values of certain predictor result in certain classifications, and this may give us some insight as to which predictors more heavily influence our response. Below is an arrangement of barplots that show the proportion of classifications that occur at each value of the given predictor.

NOTE: For each plot, the number of bins, or sections along the x-axis, and limits, the range of the x-axis, were chosen in order to accurately represent the range of values that a predictor has and make the plot as easily readable as possible. This needed to be changed manually, as one observation of some variables have values of `-9999`, which is meant to be a placeholder for `undefined`. Rather than deleting these observations, which are few and far between, I have chosen to manually remove them from the creation of these bar plots because removing this observation before plotting has caused some inexplicable issues with the plots. This observation will be removed before building models so that it will not skew any results!

```
#install.packages('ggpubr')
library(ggpubr)
#ultraviolet filter
uv <- ggplot(stellar_data, aes(u, fill = class)) + geom_bar() +
  scale_x_binned(n.breaks = 15, limits = c(5, 35)) + theme(axis.text.x = elem
ent_text(angle = 45))

#green filter
green <- ggplot(stellar_data, aes(x = g, fill = class)) + geom_bar() +
  scale_x_binned(n.breaks = 15, limits = c(5, 35)) + theme(axis.text.x = elem
ent_text(angle = 45))

#red filter
red <- ggplot(stellar_data, aes(x = r, fill = class)) + geom_bar() +
  scale_x_binned(n.breaks = 15, limits = c(5, 35)) + theme(axis.text.x = elem
ent_text(angle = 45))

#near infrared filter
near_ir <- ggplot(stellar_data, aes(x = i, fill = class)) + geom_bar() +
  scale_x_binned(n.breaks = 15, limits = c(5, 35)) + theme(axis.text.x = elem
ent_text(angle = 45))

#infrared filter
ir  <- ggplot(stellar_data, aes(x = z, fill = class)) + geom_bar() +
  scale_x_binned(n.breaks = 15, limits = c(5, 35)) + theme(axis.text.x = elem
ent_text(angle = 45))

#right ascension angle
raa <- ggplot(stellar_data, aes(x = alpha, fill = class)) + geom_bar() +
  scale_x_binned(n.breaks = 15, limits = c(0, 360)) + theme(axis.text.x = ele
ment_text(angle = 45))

#declination angle
dec <- ggplot(stellar_data, aes(x = delta, fill = class)) + geom_bar() +
  scale_x_binned(n.breaks = 15, limits = c(-20, 85)) + theme(axis.text.x = el
ement_text(angle = 45))

#specific object ID
soi <- ggplot(stellar_data, aes(x = spec_obj_ID, fill = class)) + geom_bar()
+
  scale_x_binned(n.breaks = 15) + theme(axis.text.x = element_text(angle = 4
5))

#redshift
rshift <- ggplot(stellar_data, aes(x = redshift, fill = class)) + geom_bar()
+
  scale_x_binned(n.breaks=15, limits = c(-1, 7)) + theme(axis.text.x = elemen
t_text(angle = 45))
```

```
ggarrange(uv, green,red, near_ir, ir, raa, dec, soi, rshift, ncol = 3, nrow =
3, common.legend = TRUE)
```



Before we look into these relationships, recall that the classifications are approximately 60% galaxy, 20% quasar, and 20% star, so we are looking for plots that indicate those proportions are skewed at certain values. At first glance, it seems that these proportions hold relatively true across most of our variables. This may indicate that no single variable has a major influence on classification. However, there is a noticeable difference in `redshift` . While, typically, the values that correlate with a classification of `star` are spread, stars are only classified with values of `redshift` between -0.5 and 0.5. Similarly, most of the observations in the 1.0-4.0 range are classified as `quasar` . Since this predictor alters the distribution of the classifications, we may be able to infer that `redshift` has more of an impact on our outcome than the other predictors. We will have to look at this mathematically!

\(~\)

Model Setup

Now that we have introduced our data, cleaned it up, and taken an initial look at the relationships between our variables, it’s time to model! Before we can construct and test our models, we need to prep our data for testing. We will do so by splitting our dataset up into training and testing sets, creating a recipe, or framework, for our training set that we will use to fit our models, and then set up a k-fold cross validation to help increase the effectiveness of our model fitting.

Splitting the Data

The first step in setting up our model is creating a train/test split. We do this so that we can use part of our dataset to train the model a determine the best combination of parameters for each model, and also to be able to compare model performance. For this project, we will be using a 70/30 split to ensure that our training set is large enough to effectively tune our model while also ensuring we have enough testing observations. We can then fit the model to our testing set (only once!) to see how effective and accurate our final model. For this split, we will stratify on our response variable, or our classification variable, `class` .

```
set.seed(1105) #for reproducibility

stellar_split <- initial_split(stellar_data, prop = 0.8,
                               strata = 'class')
stellar_train <- training(stellar_split)
stellar_test <- testing(stellar_split)
```

Now, we have 79999 observations in our training set and 20001 in our testing set!

Building the Recipe

Now that we've split our data, we can create a 'recipe' for our models using our training data set. Our recipe allows us to select which predictors we want to use in our models, as well as any modifications that need to be made to our predictors, such as dummy coding categorical predictors or standardizing them. In this case, all of the predictors are numeric, so they won't need to be dummy coded, only standardized. Standardizing our predictors makes them more easier to work in our models, and we will be using all 9 predictors in our recipe. We will also be using `step_downsample()` to correct the issue of our imbalanced response distribution we discovered in our data exploration.

```
library(themis)
stellar_recipe <- recipe(class ~., data = stellar_train) %>%
  step_center(all_predictors()) %>%
  step_scale(all_predictors()) %>%
  step_downsample(class) # equal proportions of each classification

prep(stellar_recipe) %>% bake(new_data = stellar_train)
```

```
## # A tibble: 79,999 × 10
##     alpha   delta        u       g      r      i        z spec_o…¹ redsh…²
class
##     <dbl>   <dbl>    <dbl>   <dbl>  <dbl>  <dbl>    <dbl>    <dbl>   <dbl>
<fct>
##  1 -0.340  0.361   0.0796   0.0656  1.58   1.18   0.0837    1.80    0.278
GALA…
##  2 -0.368  0.580   0.0933   0.0608  0.520  0.150  0.00857  -0.189   0.0937
GALA…
##  3  1.67  -1.25    0.00520  0.0922  1.06   0.807  0.0171    1.36    0.488
GALA…
##  4 -1.79  -0.617   0.00838  0.0427  0.375  0.229  0.00580   0.356  -0.135
GALA…
##  5  0.235  1.17    0.0690   0.0522  0.520  0.216  0.00886   0.505   0.115
GALA…
##  6  1.56  -0.303   0.108    0.0568  0.535  0.395  0.0117   -0.0386 -0.159
GALA…
##  7  0.688  0.0790  0.0511   0.0928  0.721  0.411  0.0229   -0.138   0.0215
GALA…
##  8  1.60  -0.720  -0.0316  -0.0494 -1.15  -1.18  -0.0569   -0.0233 -0.580
GALA…
##  9  1.74  -1.25    0.0354   0.0645  1.31   1.58   0.0904    1.38    0.332
GALA…
## 10  0.697  0.0493  0.0827   0.0459  0.432  0.231  0.00931  -0.138  -0.124
GALA…
## # … with 79,989 more rows, and abbreviated variable names ¹spec_obj_ID,
## #   ²redshift
```

Cross Validation

Recall that our data set is rather large, and we want to make sure that our model runs well because it's an effective model, not because we got lucky or we over-fit our training set to the model. To avoid over-fitting, we set up a cross validation with *k* folds,where *k* is a positive integer. What this method does is create *k* random subsets of the training set (with replacement) to provide multiple data sets for models and their hyperparameters to be trained on. To do this, we create folds within our data that represent different subsets, stratifying on our response `class` to make sure each fold is balanced in the number of observations for each classification.

```
stellar_folds <- vfold_cv(stellar_train, v = 10,
                          strata = class)
```

This is a 10-fold cross validation.

\(~\)

Model Building

Now, we've arrived at the most important part of this project, building our models! As stated in the introduction, we will be training and testing five different models to determine how well we can predict the classification of stellar phenomena as either stars, galaxies, or quasars. For this project, we will be fitting the following models:

- Elastic Net Logistic Regression
- Linear Discriminant Analysis (LDA)
- K Nearest Neighbors (KNN)
- Random Forest

Elastic Net logistic regression has two parameters, `mixture` and `penalty`. `penalty` represents the amount of regularization, and `mixture` the proportion of Lasso penalty.

K-Nearest Neighbors and Random Forest models also require certain hyperparameters, so we will also be looking for the optimal values of these parameters via tuning. These parameters will have to be tuned before we can fit the models to our training set!

K-nearest neighbors has one hyperparameter, the number of neighbors, or `k`. This value represents the number of data points nearest to new predictor values that the model will average and use to determine a classification. For example, if `k=5`, and the nearest values to a new point are classified as `star, galaxy, star, quasar, star`, then the new point will likely be classified as a star, since that is the most likely class.

Random Forests are slightly more complicated in that they have 3 hyperparameters, `mtry`, `trees`, and `min_n`.

- `mtry` represents the number of randomly selected predictors that will be used in decision making. We will tune values of this parameter in the range of 1 to 8. Any smaller than 1 and there will not be enough predictors to make a decision, and any more than 8 predictors risks over-complicating and over-fitting the model.
- `trees` represents the number of trees that will be used in our random forest.
- `min_n` represents the minimal node size, or the minimum number of observations in a terminal (end) node. Tuning this parameter also helps to ensure we don't over-fit our random forest.

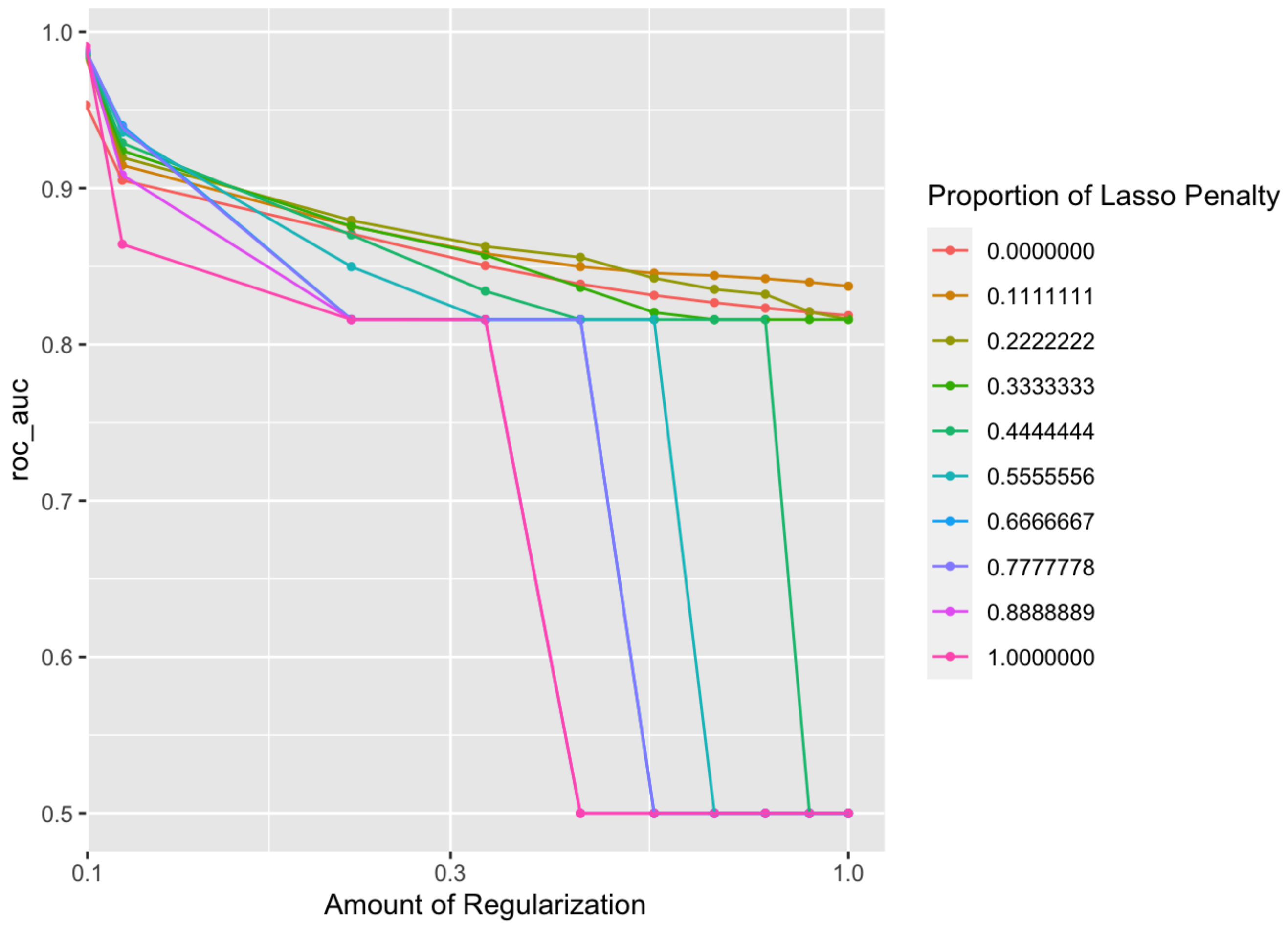
The metric we will use to examine the performance of our models is area under the ROC curve, which takes into account the difference between sensitivity and specificity, which are inversely related. This metric will pop up as `roc_auc`, and higher values indicate better model performance. We will keep track of each model's area under the ROC curve to assess which one performs the best.

NOTE: The dataset being used for this project has 100,000 observations, with 79,999 in our training set that will be used to train these models. This means that the more complicated models may take a VERY long time to run, such as Multinomial Logistic Regression, KNN and Random Forest. As such, they will be saved as RData files and loaded into the global environment to avoid rerunning each model every time the file is run.

Elastic Net Logistic Regression

Recall for Elastic Net Logistic Regression requires the tuning of 2 parameters, `mixture` and `penalty` . When setting up our model, we set `penalty = tune()` and `mixture = tune()` rather than setting them to integers. This allows us to tune for the optimal value, then go back and 'replace' it later when we fit our knn model to our training set. We will follow the same procedure for all of our models that require tuning parameters

```
#setting up the model
en_mod <- multinom_reg(penalty = tune(), mixture = tune()) %>%
  set_engine('glmnet') %>%
  set_mode('classification')

#setting up workflow
en_wflow <- workflow() %>%
  add_recipe(stellar_recipe) %>%
  add_model(en_mod)

#setting up grid of possible mixture and penalty values
en_grid <- grid_regular(penalty(range = c(0, 1),
                                     trans = identity_trans()),
                        mixture(range = c(0, 1)),
                             levels = 10)

# tuningthe model to our folds
# en_tune <- tune_grid(
#   object = en_wflow,
#   resamples = stellar_folds,
#   grid = en_grid
# )

#save(en_tune, file = 'en_tune.rda')
load('parameter tuning/en_tune.rda')
autoplot(en_tune, metric = 'roc_auc')
```



Here, we can see that little to no regularization correlates with higher ROC AUC, but it is unclear which proportions of Lasso penalty follows suit. We will figure this out mathematically when we determine which model performs the best across our folds.

Linear Discriminant Analysis

For Linear Discriminant Analysis, there are no tuning parameters, so we can fit this model to the entire training set rather than fitting to the folds to assess how each parameter combination performs.

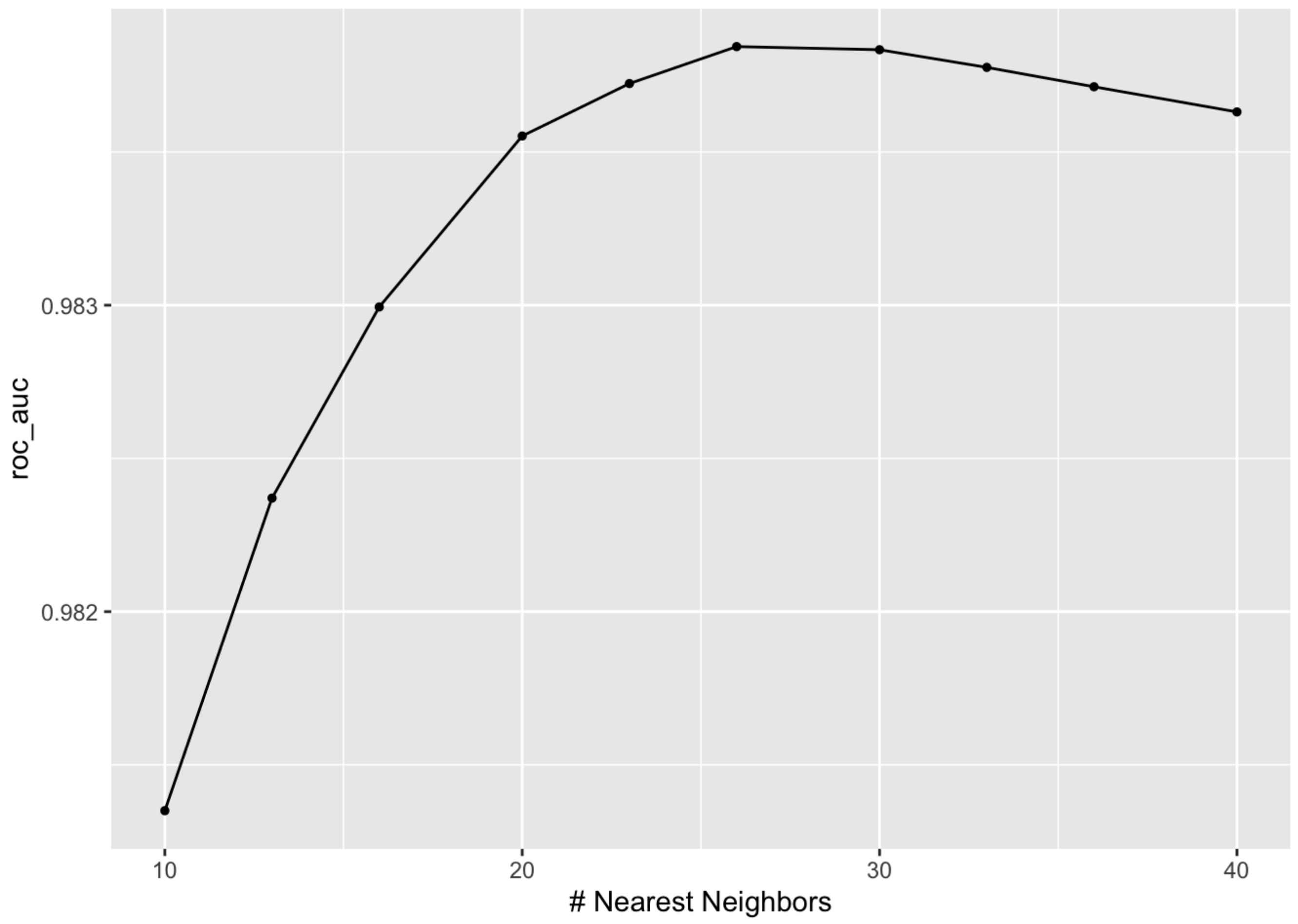
```
library(discrim)
#setting up the models
lda_mod <- discrim_linear() %>%
  set_engine('MASS') %>%
  set_mode('classification')

#setting up workflows
lda_wflow <- workflow() %>%
  add_recipe(stellar_recipe) %>%
  add_model(lda_mod)

#fitting to the training set
# lda_fit <- fit(lda_wflow,stellar_train)
# save(lda_fit, file = 'lda_fit.rda')

load('training models/lda_fit.rda')

#retrieving ROC AUC value
lda_pred_res <- augment(lda_fit, new_data = stellar_train)
lda_pred_res %>% roc_auc(class,.pred_GALAXY:.pred_STAR)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.929
```

The ROC AUC value for our LDA is 0.929, which is not quite as good as our Elastic Net Logistic Regression that has maximum values closer to 1. But 0.929 is still an indicator of an effective model, so let’s see if we can do better!

K Nearest Neighbors

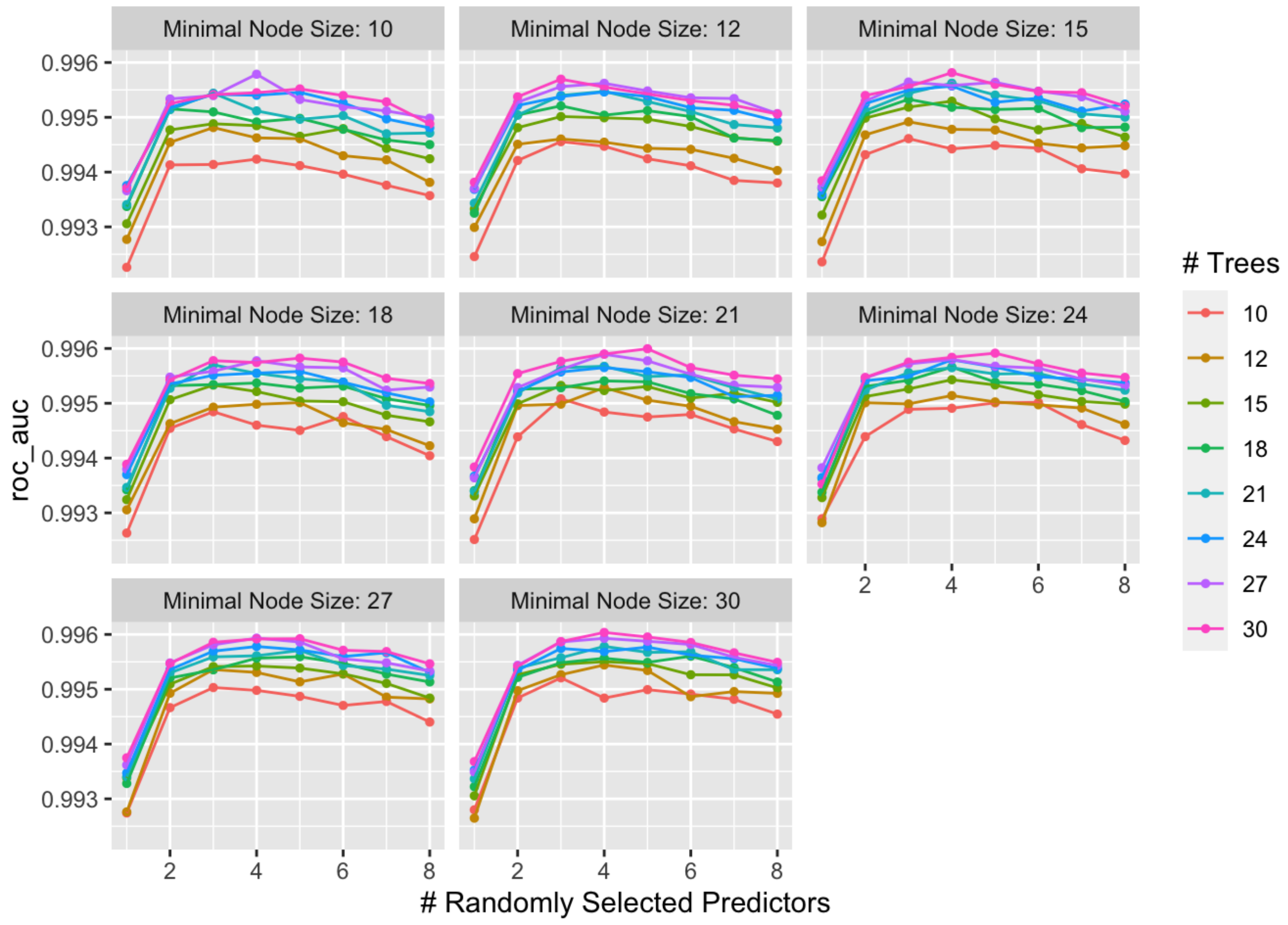
```r
#setting up model
knn_mod <- nearest_neighbor(neighbors = tune()) %>%
  set_engine('kknn') %>%
  set_mode('classification')

#setting up workflow
knn_wflow <- workflow() %>%
  add_recipe(stellar_recipe) %>%
  add_model(knn_mod)
#creating grid of possible k values
knn_grid <- grid_regular(neighbors(range=c(10,40)), levels = 10)

#fitting to the folds
# knn_tune <- tune_grid(
#   object = knn_wflow,
#   resamples = stellar_folds,
#   grid = knn_grid
# )
#save(knn_tune, file = 'knn_tune.rda')
load('parameter tuning/knn_tune.rda')
autoplot(knn_tune, metric = 'roc_auc')
```



It looks like around 26 or 27 neighbors are needed to optimize ROC AUC for our KNN model. Even with this number, our average ROC_AUC value is around 0.984 or 0.985 across folds, which is lower than both the Elastic Net and LDA models.

Random Forest

```r
# setting up model
rf_mod <- rand_forest(mtry = tune(), trees = tune(), min_n = tune()) %>%
  set_engine('ranger') %>%
  set_mode('classification')

# setting up workflow
rf_wflow <- workflow() %>%
  add_recipe(stellar_recipe) %>%
  add_model(rf_mod)

# setting up grid for possible mtry, trees, and min_n values
rf_grid <- grid_regular(mtry(range = c(1,8)), trees(range = c(10,30)),
                            min_n(range = c(10,30)), levels = 8)

# tuning the model to the folds
# rf_tune <- tune_grid(
#   object = rf_wflow,
#   resamples = stellar_folds,
#   grid = rf_grid
# )
# save(rf_tune,file = 'rf_tune.rda')

load('parameter tuning/rf_tune.rda')
autoplot(rf_tune, metric = 'roc_auc')
```



Here, it looks like more trees increases the ROC AUC, but minimum node size and number of predictors are slightly less clear. It looks like approximately 2-6 predictors will be most effective depending on the node size, but we will determine which exact combination of hyperparameters yields the highest ROC AUC in the next step.

\(~\)

Selecting our Best Model

We have fit our LDA to the entire training set, but we still have to figure out which parameter combinations yield the highest ROC AUC values for each model and fit them to our entire training set to compare performances.

First, we will select the best performing model across our folds for KNN, Elastic Net, and Random Forest. Then we will compare the 3 ROC AUC values to our LDA and select our model that performs the best on our training set to be fit to our testing set. Recall that the ROC AUC for our LDA model was 0.929, which is already very high!

We will also be saving our final models on our training sets as RData files and commenting our the actual code, so that we can load them in and save runtime in the future.

```
#selecting our best models for Elastic Net, KNN, and Random Forest

# best_en <- select_best(en_tune) #mixture = 1, mixture = 0
# best_knn <- select_best(knn_tune) #neighbors = 26
# best_rf <- select_best(rf_tune) #mtry = 4, trees = 30, min_n = 30
#
# #finalizing our workflows and fitting to the entire training set
#
# final_en_wflow <- finalize_workflow(en_wflow, best_en)
# final_en <- fit(final_en_wflow, stellar_train)
#
# final_knn_wflow <- finalize_workflow(knn_wflow, best_knn)
# final_knn <- fit(final_knn_wflow, stellar_train)
#
# final_rf_wflow <- finalize_workflow(rf_wflow, best_rf)
# final_rf <- fit(final_rf_wflow, stellar_train)

#saving our final models
# save(final_en, file = 'final_en.rda')
# save(final_knn, file = 'final_knn.rda')
# save(final_rf, file = 'final_rf.rda')

load('training models/final_en.rda')
load('training models/final_knn.rda')
load('training models/final_rf.rda')
```

Now hat we have all 4 models fitted to our training set, we can compare their ROC AUC values to determine which model performs the best and should be fit to our testing set. Here are the `roc_auc` values displayed together.

NOTE: `knn_roc` has been saved as an RData file and then loaded in to reduce knitting time!

```
en_roc <- augment(final_en, stellar_train) %>%
  roc_auc(class, .pred_GALAXY:.pred_STAR) %>% select(.estimate)

# knn_roc <- augment(final_knn, stellar_train) %>%
#   roc_auc(class, .pred_GALAXY:.pred_STAR) %>% select(.estimate)
#save(knn_roc, file = 'knn_roc.rda')

load('knn_roc.rda')

rf_roc <- augment(final_rf, stellar_train) %>%
  roc_auc(class, .pred_GALAXY:.pred_STAR) %>% select(.estimate)

lda_roc <- lda_pred_res  %>% roc_auc(class, .pred_GALAXY:.pred_STAR) %>%
  select(.estimate)

roc_auc <- c(en_roc$.estimate, lda_roc$.estimate,
             knn_roc$.estimate, rf_roc$.estimate)
models <- c('Elastic Net Logistic Regression', 'LDA', 'KNN', 'Random Forest')

roc_auc_table <- tibble(models, roc_auc) %>% arrange(-roc_auc)
roc_auc_table
```

```
## # A tibble: 4 × 2
##   models                          roc_auc
##   <chr>                             <dbl>
## 1 Random Forest                     0.999
## 2 KNN                               0.992
## 3 Elastic Net Logistic Regression   0.991
## 4 LDA                               0.929
```

Turns out, our models performed even better on the entire training set than on our folds, and our LDA model ended up being the worst performer - and even LDA has a ROC AUC of over 0.9! Our best performing model was our Random Forest with 4 predictors, 30 trees, and a minimal node size of 30 observations, with a ROC AUC value of 0.999, which is near perfect. Now, we can fit the model to our testing set and draw some final conclusions about our model performance and its implications.

\(~\)

Testing and Analyzing our Best Model

Below, we fit our best model, the Random Forest, to the testing set. We will then take a look at some visualizations to analyze our testing model's performance before we draw some conclusions

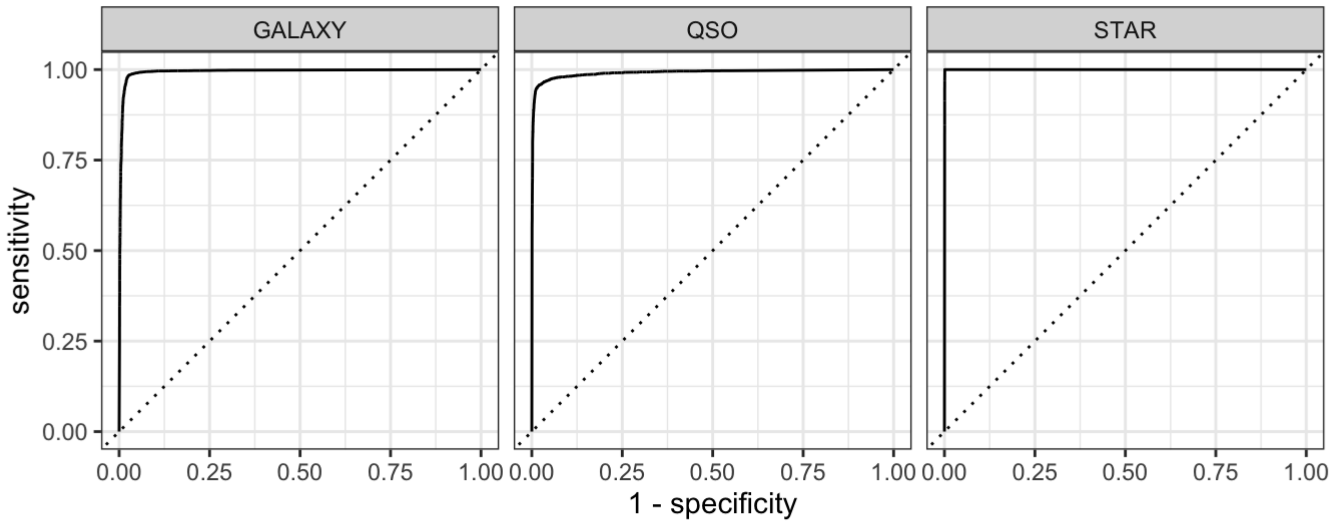
```
rf_test <- augment(final_rf, new_data = stellar_test)
```

Let's take a look at the ROC AUC value for our testing set and the corresponding ROC curve.

```
rf_test %>% roc_auc(class, .pred_GALAXY:.pred_STAR)
```

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc hand_till      0.996
```

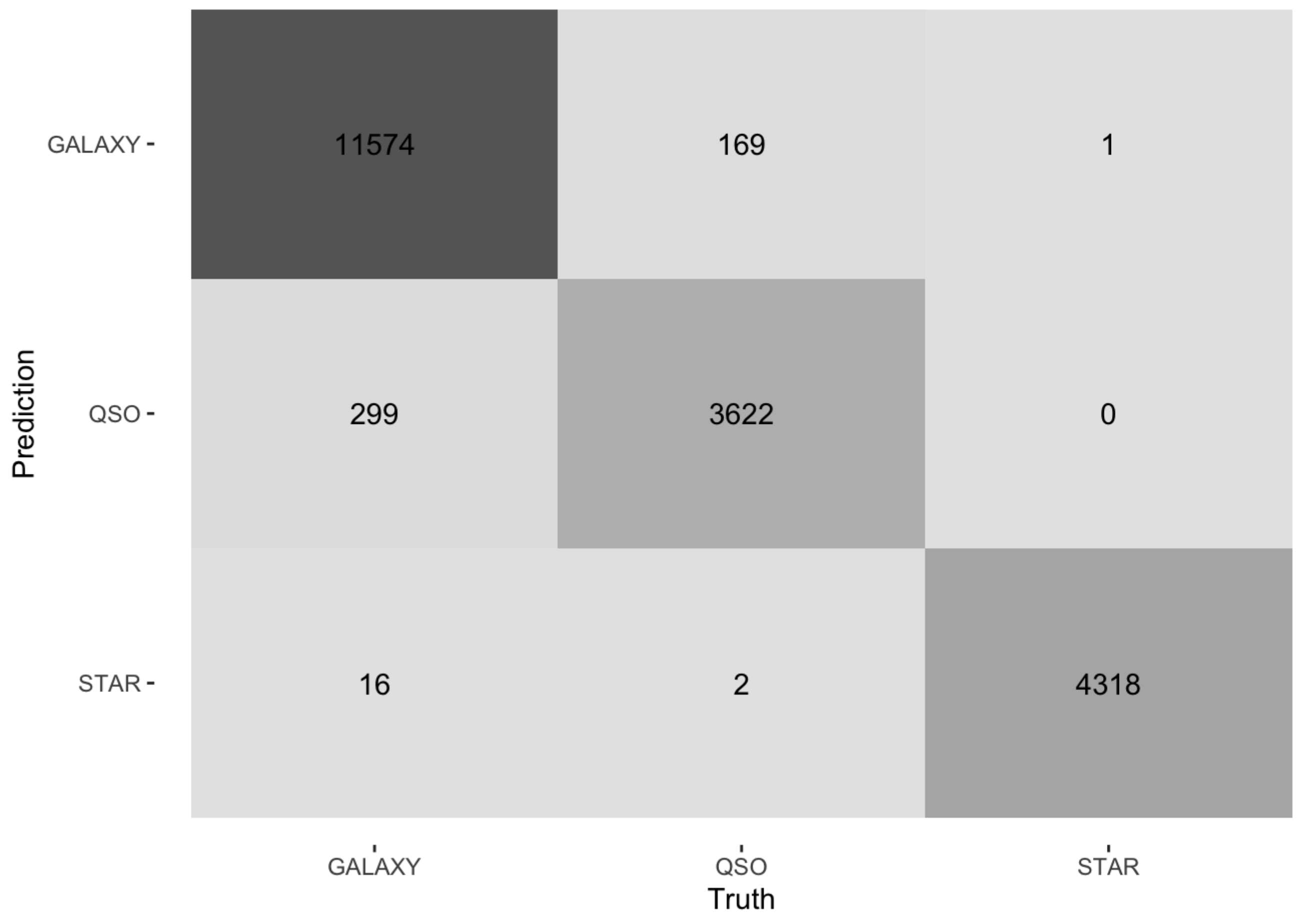
```
rf_test %>% roc_curve(class, .pred_GALAXY:.pred_STAR) %>% autoplot()
```



Our testing `roc_auc` is 0.996, which is only 0.003 lower than our training value,indicating that our chosen random forest model performs nearly just as well on the testing set as it does on the training set. When looking at the curves, we want them to be as far away from the dotted line as possible. All three classes follow this pattern, with `star` perfectly so.

Another helpful visual for analyzing model performance is a confusion matrix. Confusion matrices show what the model predicted versus the true value for each observation, so it’s easy to visualize where the model did well with predictions and wehre it did poorly.

```
rf_test %>% conf_mat(class, .pred_class) %>%
  autoplot(type = 'heatmap')
```



Out of 20,001 observations in our testing set, only 487 were predicted incorrectly, giving us approximately a 97.5% accuracy rate, which we can confirm mathematically below. This matrix shows us where the model predicted well and where it did not. For example, galaxies were incorrectly predicted as stars or quasars more than the other two classes, whereas stars were only predicted incorrectly once. We can see from the matrix that, overall, this model did very well at classifying stellar phenomena

```
rf_test %>% accuracy(class, .pred_class)
```

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy multiclass     0.976
```

\(~\)

Conclusion

To sum everything up, our research and testing thus far has brought us to our Random Forest model, with 4 predictors, 30 trees, and a minimum node size of 30 observations. This model did an exemplary job of predicting stellar classifications, with a 0.996 ROC AUC value and 97.6% accuracy rate. Although the Random Forest was chosen as our best model and fit to the testing set, the LDA, KNN, and Elastic Net Logistic Regression were not far behind and all has training ROC AUC values above 0.9.

Our analysis of stellar classifications led to some interesting conclusions. Primarily, the confusion matrix gave a lot of insight. Though galaxies have the most representation in the testing set, they were also predicted most inaccurately. Specifically, the most common issue was that galaxies were incorrectly predicted as quasars. I struggled to figure out why, until I went back and looked at the visual exploratory analysis. It appears that the distribution of predictor values for galaxies and quasars were relatively similar, meaning they similar values across observations. This may have been confusing for the model to differentiate, and there may be a solution in more precise tuning and predictor manipulation. It was challenging working with a multi-class classification as opposed to binary, which caused some problems when looking at the relationships between predictor and response as well as between the classifications themselves.

Despite some challenges, I think we still were able to produce great results and were able to accurately classify stars, galaxies, and quasars given information about their light properties and positioning in the sky. In the future, if I were to work with this dataset again, I would want to more deeply investigate the relationships between predictors. I think that this next step would create a better understanding of what goes into classifying stellar images and a deeper knowledge of how variables interact with each other. Also, personally, I would have liked to create some interesting visualizations of the data and improve my skills in this area.

Overall, this project has been a great opportunity to develop my machine learning techniques and skills in an area of study that I am passionate about, astronomy, and dive into the world of light spectroscopy and stellar imaging!