Code

PSTAT 131 Final Project

Frankie Ma

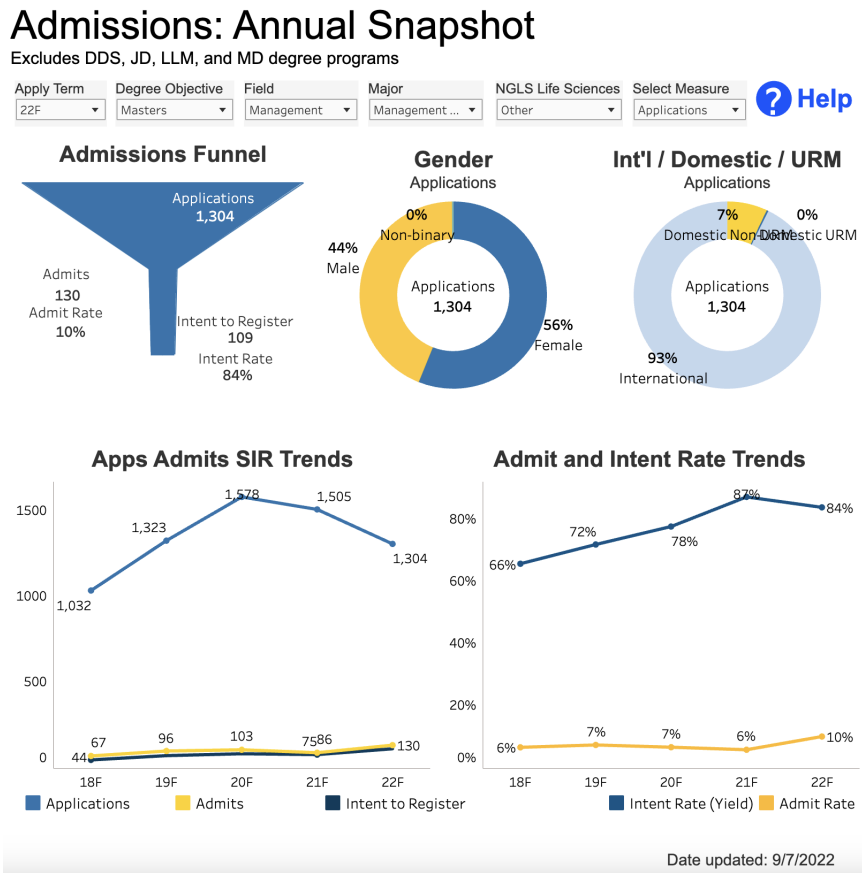
2023-09-17

Show

Introduction

The purpose of this project is to predict whether or not a student will be admitted in a Master program in the University of California, Los Angeles (UCLA). The data that will be used are downloaded from Kaggle and we will be saving it as `predicting_data`. (Credits to: Mohan S Acharya, Asfia Armaan, Aneeta S Antony : A Comparison of Regression Models for Prediction of Graduate Admissions, IEEE International Conference on Computational Intelligence in Data Science 2019) The link of the source can be found here: https://www.kaggle.com/datasets/mohansacharya/graduate-admissions?select=Admission_Predict_Ver1.1.csv (https://www.kaggle.com/datasets/mohansacharya/graduate-admissions?select=Admission_Predict_Ver1.1.csv)

Motive



Since a lot of students are determined to pursue a master degree after obtaining their bachelor's degree, including me, as a senior, I think it would be interesting to know what attracts the admission officers, or their standard of sending an offer to students. Based on the dataset, I desire to predict the possibility of admission for each students based on past admission statistics. This analytic project is specialized in the Master of Science in Business Analytics, a graduate program in UCLA that I am interested in applying, so this gives me a chance to get a better understanding of the program and my chance of getting in. After the obtaining the

predicted chance of admitting, the data will be categorized into "Admitted" and "Rejected" based on a certain scale (which I'm still deciding), so this project will end up as classification models.

Class of 2023 Admissions Statistics

Show

This video on YouTube shows some additional information about the program, and the admission statistics for the Class of 2023 can be found [20:38].

From UCLA official admission statistics of MSBA program in the video above, here are some important information:

The admittance rate is only about 7% (93 out of 1315) with international student taking up 60%, which means that only 55 international students got accepted!

The average GPA is 3.6. The average TOEFL is 110.

So, based on the information, it is fair to consider scores above the average is competitive.

Data Description

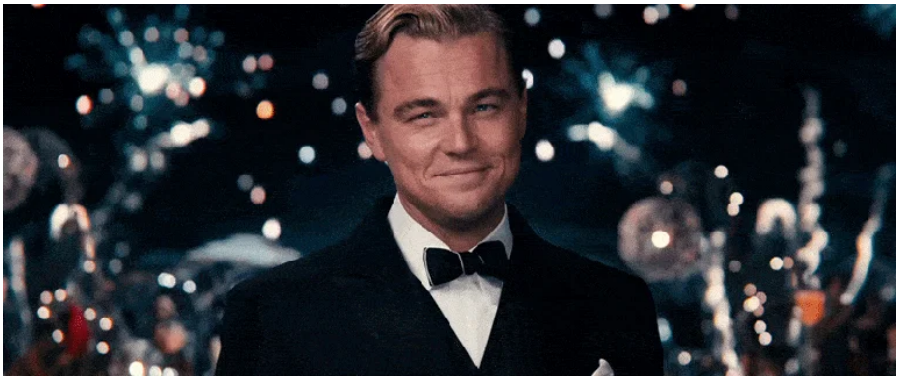
The dataset contains 500 entries of data and 6 numeric parameters:

1. GRE Scores (out of 340) This is a standard test that is similar to SAT and ACT that students used to apply for undergraduate programs. The test contains 3 sections: Analytical Writing, Verbal Reasoning, and Quantitive Reasoning. Nowadays, GRE scores are longer a "must" when applying to many master programs, but all programs do encourage students to submit it.
2. TOEFL Scores (out of 120) A language test that are required for international students who "do not attend a university where the medium of instruction was English, but the official language of the country was NOT English (this includes both India and Singapore). (Additional information can be found: https://www.anderson.ucla.edu/degrees/master-of-science-in-business-analytics/admissions/prerequisites#a-1170948 (https://www.anderson.ucla.edu/degrees/master-of-science-in-business-analytics/admissions/prerequisites#a-1170948))
3. University Rating (out of 5)
4. Statement of Purpose (out of 5)
5. Letter of Recommendation Strength (out of 5)
6. Undergraduate GPA （out of 10)
7. Research Experience (either 0 or 1)
8. Chance of Admit (ranging from 0 to 1)

Response Variable and Determination

Even though the default threshold is 60%, in this project, I would like to keep it at that level and not to rise the admitted level, because students are not limited to apply to one program and sometimes they might be wait-listed to wait for an empty spot which others might end up

declining the offer when they get the admission from another program which they prefer over this. So, it wouldn't hurt for students to apply without having an admitted rate high enough to guarantee their spot.



Let's get this party started!

Data Processing

First, let's load all the packages we are going to need for following analysis and check if there are any missing value or outliers and clean up the data first, and then take a general look at the data:

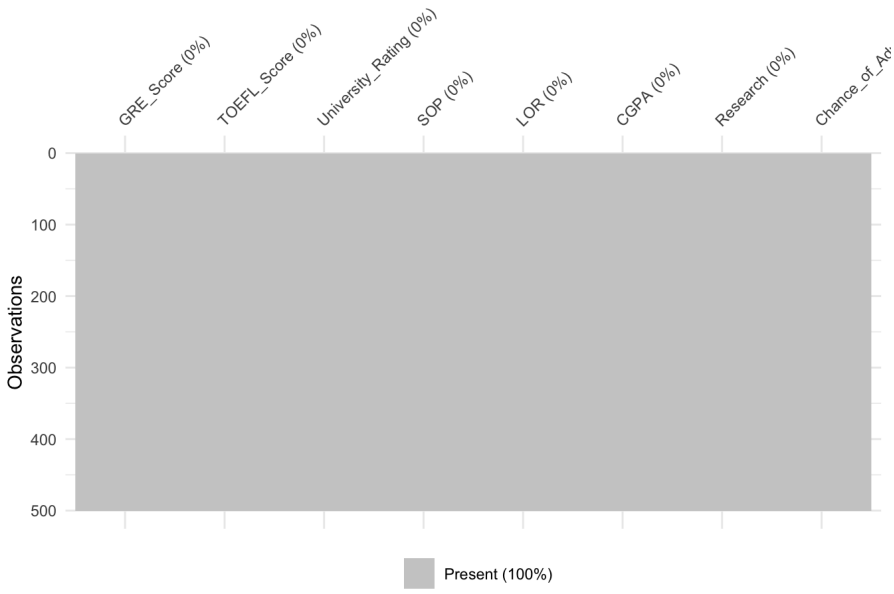
Show

Missing Values

Show

```
## [1] "CGPA"              "Chance_of_Admit"   "GRE_Score"
## [4] "LOR"               "Research"          "SOP"
## [7] "TOEFL_Score"       "University_Rating"
```

Show



Looks like we don't have any missing data in our data set, which is fantastic!

Exploring Data

GPA

Since the given GPA in the dataset was not documented using the 4.0 scale, for clear interpretation, let's convert it into the scale we are familiar with.

Show

```
## # A tibble: 144 × 8
##    GRE_Score TOEFL_Score University_Rating   SOP   LOR Research Chance
_of_Admit
##        <dbl>       <dbl>             <dbl> <dbl> <dbl>    <dbl>
<dbl>
##  1       337         118                 4   4.5   4.5        1
0.92
##  2       330         115                 5   4.5   3          1
0.9
##  3       327         111                 4   4     4.5        1
0.84
##  4       328         112                 4   4     4.5        1
0.78
##  5       328         116                 5   5     5          1
0.94
##  6       334         119                 5   5     4.5        1
0.95
##  7       336         119                 5   4     3.5        1
0.97
##  8       340         120                 5   4.5   4.5        1
0.94
##  9       338         118                 4   3     4.5        1
0.91
## 10       340         114                 5   4     4          1
0.9
## # ℹ 134 more rows
## # ℹ 1 more variable: GPA <dbl>
```

TOEFL Score

Show

```
## # A tibble: 193 × 8
##    GRE_Score TOEFL_Score University_Rating   SOP   LOR Research Chance
_of_Admit
##        <dbl>       <dbl>             <dbl> <dbl> <dbl>    <dbl>
<dbl>
##  1       337         118                 4   4.5   4.5        1
0.92
##  2       322         110                 3   3.5   2.5        1
0.8
##  3       330         115                 5   4.5   3          1
0.9
##  4       327         111                 4   4     4.5        1
0.84
##  5       328         112                 4   4     4.5        1
0.78
##  6       318         110                 3   4     3          0
0.63
##  7       325         114                 4   3     2          0
0.7
##  8       328         116                 5   5     5          1
0.94
##  9       334         119                 5   5     4.5        1
0.95
## 10       336         119                 5   4     3.5        1
0.97
## # ℹ 183 more rows
## # ℹ 1 more variable: GPA <dbl>
```

193 out of 500 students, which is 38.6%, have scored at least 110 on TOEFL.

Based on the class profile we have above and focus on these two essential preditors, let's find out their corresponding `chance of admit` compared to the actual statistic.

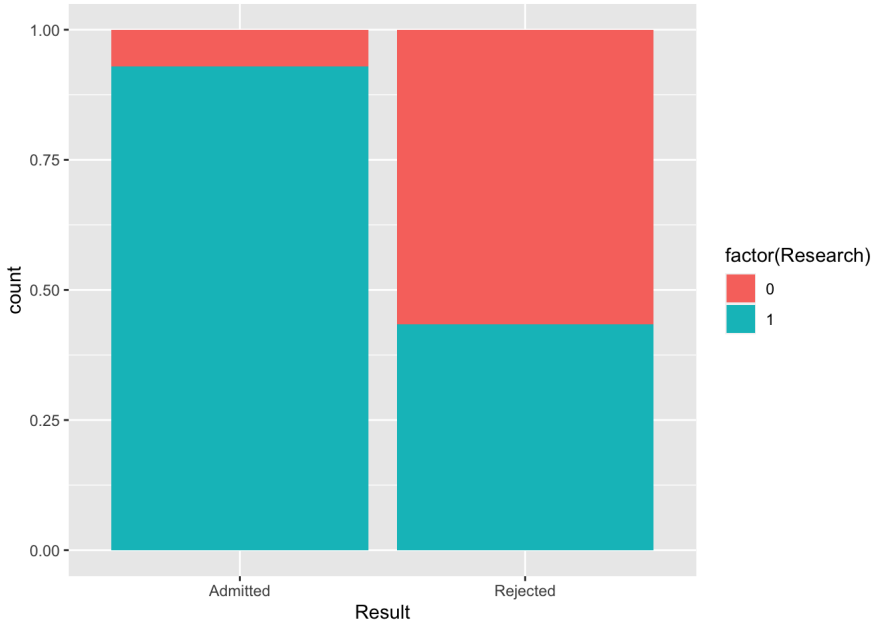
Show

```
## # A tibble: 2 × 2
##   Result   avg_rate
##   <chr>       <dbl>
## 1 Rejected    0.665
## 2 Admitted    0.890
```

Effect of number of research on addmittance

Do number of research have anything to do with their chance of getting accepted? Since I have decided the probability of admission above 60% to be "Admitted", let's compare it with the number of research under these categories:

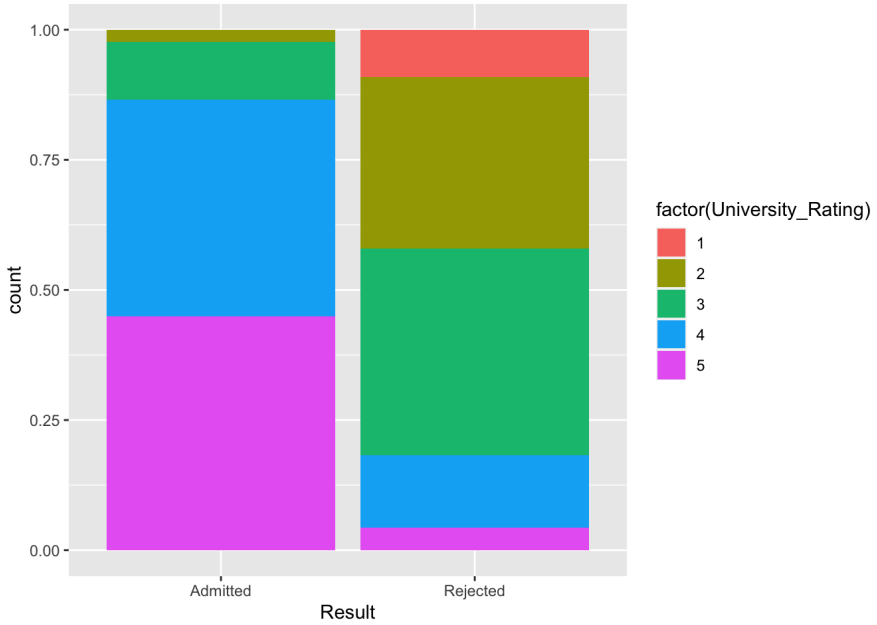
Show



Well, it is definitely good to have done a research, which can be a good representative of your academic skill, but not knowing what kind of research and whether it is relevant for this specific program, we will not emphasize on this aspect.

Effect of university rating on addmittance

Show



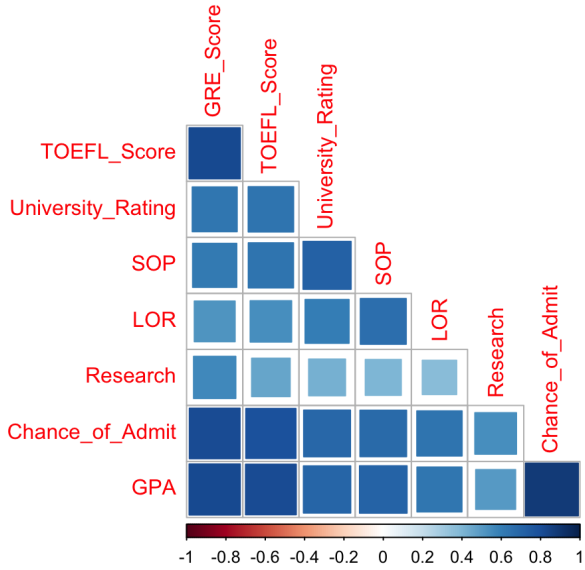
Based on the bar chart, we can see the admittance has some relevance between the `University Rating` and the final result that if students are from a better University they are more likely to get in, however, by observing the portion taken up under each category, we can

see that the largest percentage

Correlation Matrix

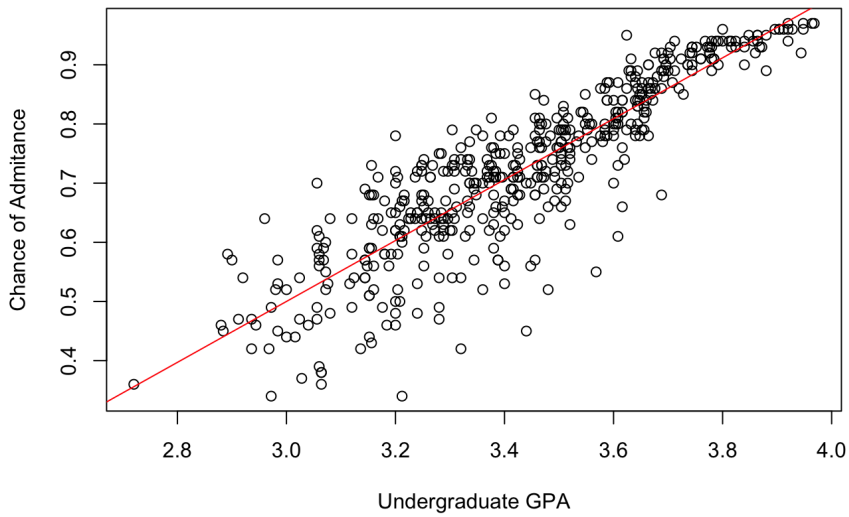
Now let's take a look at the distribution of `Chance of Admit`.

Show

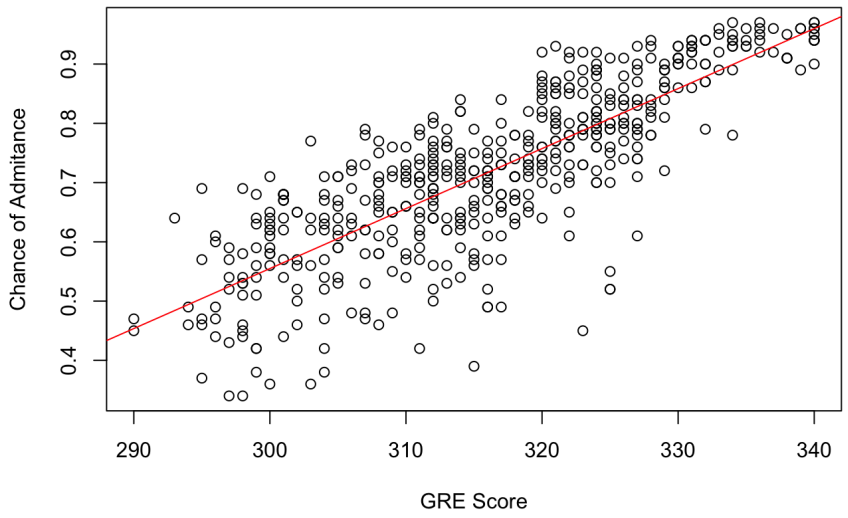


Looks like all the predictors have a relative strong positive correlation with each other except for `Research` since the color is much lighter. Since our purpose is to predict the chance of getting admitted, let's focus on the bottom line that we find `GPA`, `GRE_Score`, and `TOEFL_Score` have an incredible positive correlation with it, so I will dig into it later.

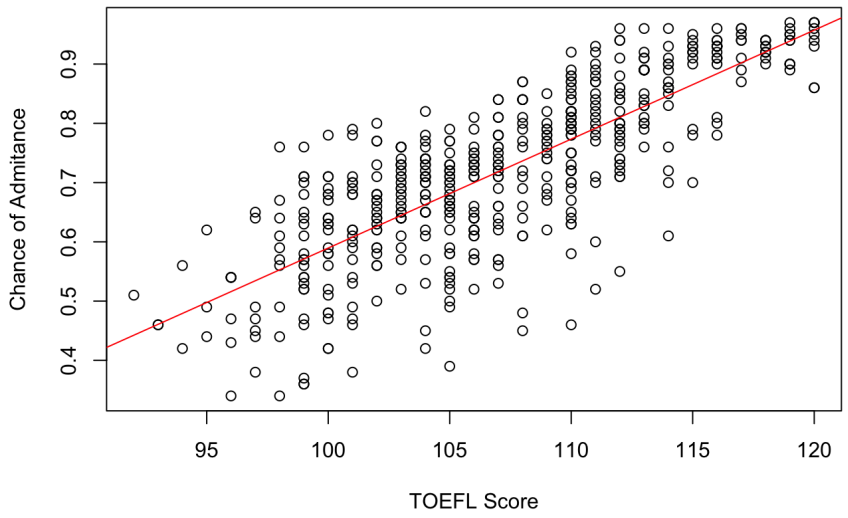
Show



Show



Show



Model Building

Data Splitting

Let's first start with a simple stratified sampling:

Show

Since we do not have a fairly large dataset with more than 1,000 observations and to build a better model, I decided to set the percentage as 80%.

Creating a Recipe

By using the training data, I create a recipe predicting the outcome variable, `Chance_to_Admit` , with all other predictor variables. And since interaction effect occurs when the effect of one variable depends on the value of another variable, I decided to build interactions between:

- `GPA` and `GRE_Score`
- `GPA` and `TOEFL_Score`

Show

```
## # A tibble: 6 × 10
##   GRE_Score TOEFL_Score University_Rating   SOP   LOR Research   GPA
##       <dbl>       <dbl>             <dbl> <dbl> <dbl>    <dbl> <dbl>
## 1       302         102                 1   2     1.5        0  3.2
## 2       307         109                 3   4     3          1  3.2
## 3       311         104                 3   3.5   2          1  3.28
## 4       298          98                 2   1.5   2.5        1  3
## 5       295          93                 1   2     2          0  2.88
## 6       310          99                 2   1.5   2          0  2.92
## # ℹ 3 more variables: Chance_of_Admit <dbl>, GPA_x_GRE_Score <dbl>,
## #   GPA_x_TOEFL_Score <dbl>
```

Linear Regression

First, specify the model engine we want to fit, in this case, linear regression model, and then setting up a workflow:

Show

Now, let's fit the training data into this model and see how it fits:

Show

```
## # A tibble: 10 × 5
##    term               estimate std.error statistic p.value
##    <chr>                 <dbl>     <dbl>     <dbl>   <dbl>
##  1 (Intercept)       -2.54       1.22       -2.09  0.0374
##  2 GRE_Score          0.00551    0.00688     0.800 0.424
##  3 TOEFL_Score        0.00422    0.0132      0.321 0.749
##  4 University_Rating  0.00623    0.00423     1.47  0.141
##  5 SOP                0.00599    0.00498     1.20  0.229
##  6 LOR                0.0140     0.00438     3.21  0.00146
##  7 Research           0.0223     0.00714     3.12  0.00192
##  8 GPA                0.648      0.361       1.79  0.0737
##  9 GPA_x_GRE_Score   -0.00102    0.00204    -0.497 0.620
## 10 GPA_x_TOEFL_Score -0.000402   0.00385    -0.104 0.917
```

Next, let's use the following code to predict `Chance_of_Admit` value for each observation in the training set and compare it with the actual observed `Chance_of_Admit` value:

Show

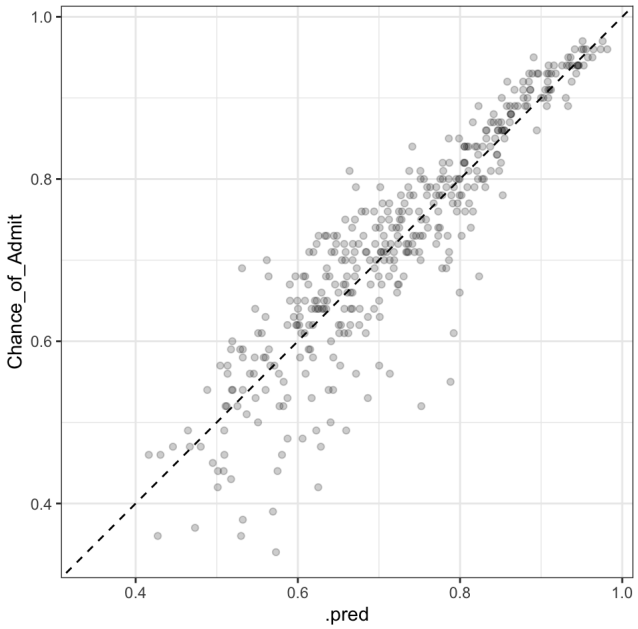
```
## # A tibble: 6 × 2
##   .pred Chance_of_Admit
##   <dbl>           <dbl>
## 1 0.551             0.5
## 2 0.651             0.62
## 3 0.651             0.61
## 4 0.509             0.44
## 5 0.416             0.46
## 6 0.488             0.54
```

Show

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0576
## 2 rsq     standard      0.828
## 3 mae     standard      0.0413
```

To have a better view and interpretation of the above data, let's put it in a plot:

Show



It is clear to see that our dots forms a straight line, it's a sign that this model did a good job! Congrats! But for a better comparision with the statistics we are going to have in through validation approach, let's find out its mean squared error (RMSE) and the **testing** RMSE.

Show

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0666
## 2 rsq     standard      0.800
## 3 mae     standard      0.0452
```

Understanding our criteria metric: - RMSE (root mean squared error): shows how far apart the predicted values are from the observed values in the dataset on average, the lower the better fit

- R^2 (range from 0 to 1): shows the proportion of the variance in the response variable of a regression model that can be explained by the predictor variables, the higher the better fit

K-Nearest Neighbors

Show

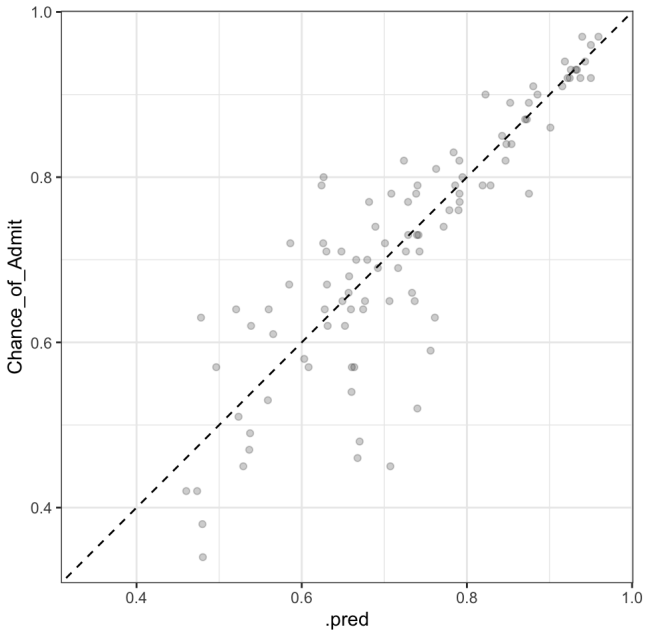
Let's fit the data to this model and review the results:

Show

```
## parsnip model object
##
##
## Call:
## kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(5,
data, 5))
##
## Type of response variable: continuous
## minimal mean absolute error: 0.04724016
## Minimal mean squared error: 0.004573518
## Best kernel: optimal
## Best k: 5
```

It has suggested our best k value to be 5, which is our default value. Let's generate predictions from this model for the training set and testing set and compare their RMSE like we did for linear regression:

Show



Show

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0365
## 2 rsq     standard      0.933
## 3 mae     standard      0.0255
```

Show

```
## # A tibble: 3 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0751
## 2 rsq     standard      0.746
## 3 mae     standard      0.0526
```

Conclusion

As we observe from the above statistics, even though KNN model performs better with a lower RMSE than linear regression, I consider linear regression as a better model since we want what is best for the testing set.

Evaluation

Validation Approach - Linear Model

We have finished the train-test split and deciced linear model to be our best model so far, let's now consider using the validation approach that we will train our models on the training sample, and then choose a best-fitting model by comparing their performances on the validation set.

Since we only have about 400 data entries in our original training set, let's have a lower percentages for splitting the data.

Show

Since we have already set up a basic model in previous section, there is no need to create it again, let's just fit the data using `fit_resamples()` instead of `fit()` and see how it results:

Show

```
## # A tibble: 2 × 6
##   .metric .estimator   mean     n std_err .config
##   <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   0.0588     1      NA Preprocessor1_Model1
## 2 rsq     standard   0.822      1      NA Preprocessor1_Model1
```

We can see there are no standard error values because we have only one resample (n=1). Furthermore, we observe a RMSE value of about 0.059 and an R^2 value of about 0.822.

Comparing this result from the resamples to our previous result we get from train-test split, we found them to have similar metrics for training set.

K-fold Cross-validation

Acknowledging the fact that K-fold Cross-validation usually generates best estimates of testing data, so let's try through this approach by tuning to find the best value of neighbors that yields the best performances. In this case, we need to create a new recipe:

Show

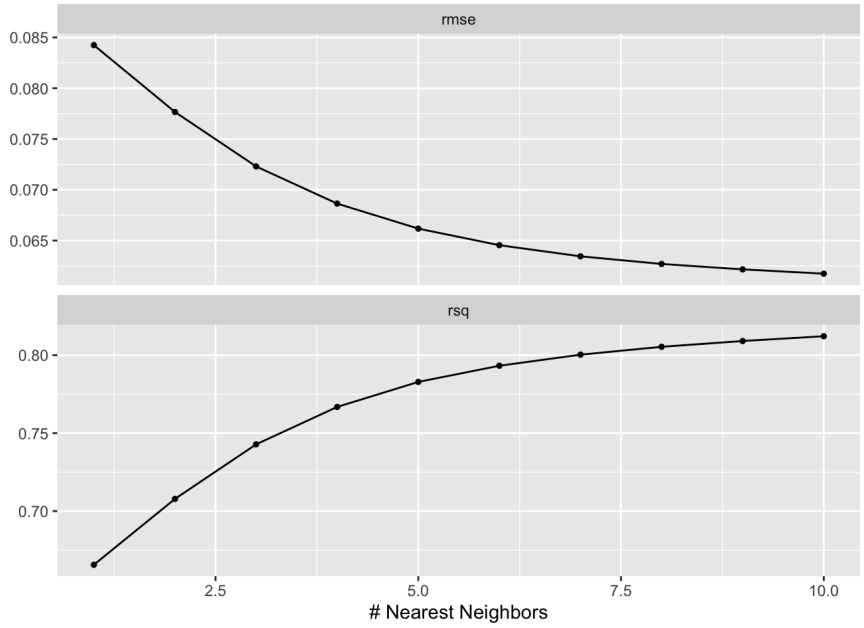
Next, let's create a k-fold dataset using the `vfold_cv()` function. The folds, `v` describe a number of groups we decide to partition the data, to make a better model, I decided to have 10 folds.

Show

```
## # A tibble: 40 × 8
##    GRE_Score TOEFL_Score University_Rating   SOP   LOR Research Chance
_of_Admit
##        <dbl>       <dbl>             <dbl> <dbl> <dbl>    <dbl>
<dbl>
##  1       305         108                 5   3     3          0
0.61
##  2       290         104                 4   2     2.5        0
0.45
##  3       306         106                 2   2     2.5        0
0.61
##  4       315          99                 2   3.5   3          0
0.63
##  5       295          96                 2   1.5   2          0
0.47
##  6       300         102                 3   3.5   2.5        0
0.63
##  7       297          98                 2   2.5   3          0
0.59
##  8       303          98                 1   2     2.5        0
0.56
##  9       302          99                 3   2.5   3          0
0.52
## 10       309         105                 2   2.5   4          0
0.55
## # ℹ 30 more rows
## # ℹ 1 more variable: GPA <dbl>
```

We also need a grid to fit the models within each fold, so we'll use `tune_grid()` to achieve that. Then, we will use `autoplot()` to have a better overview of the performance of different hyperparameter (neighbors) values:

Show



Show

```
## # A tibble: 20 × 7
##    neighbors .metric .estimator   mean     n std_err .config
##        <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
##  1         1 rmse    standard   0.0842    10 0.00434 Preprocessor1_Mod
el01
##  2         1 rsq     standard   0.666     10 0.0225  Preprocessor1_Mod
el01
##  3         2 rmse    standard   0.0777    10 0.00443 Preprocessor1_Mod
el02
##  4         2 rsq     standard   0.708     10 0.0231  Preprocessor1_Mod
el02
##  5         3 rmse    standard   0.0723    10 0.00427 Preprocessor1_Mod
el03
##  6         3 rsq     standard   0.743     10 0.0217  Preprocessor1_Mod
el03
##  7         4 rmse    standard   0.0686    10 0.00405 Preprocessor1_Mod
el04
##  8         4 rsq     standard   0.767     10 0.0197  Preprocessor1_Mod
el04
##  9         5 rmse    standard   0.0662    10 0.00392 Preprocessor1_Mod
el05
## 10         5 rsq     standard   0.783     10 0.0184  Preprocessor1_Mod
el05
## 11         6 rmse    standard   0.0645    10 0.00385 Preprocessor1_Mod
el06
## 12         6 rsq     standard   0.793     10 0.0178  Preprocessor1_Mod
el06
## 13         7 rmse    standard   0.0634    10 0.00384 Preprocessor1_Mod
el07
## 14         7 rsq     standard   0.800     10 0.0176  Preprocessor1_Mod
el07
## 15         8 rmse    standard   0.0627    10 0.00384 Preprocessor1_Mod
el08
## 16         8 rsq     standard   0.805     10 0.0174  Preprocessor1_Mod
el08
## 17         9 rmse    standard   0.0622    10 0.00382 Preprocessor1_Mod
el09
## 18         9 rsq     standard   0.809     10 0.0172  Preprocessor1_Mod
el09
## 19        10 rmse    standard   0.0617    10 0.00379 Preprocessor1_Mod
el10
## 20        10 rsq     standard   0.812     10 0.0169  Preprocessor1_Mod
el10
```

The line graph tells us that with increasing the number of neighbors, we are more likely to have a better performance. However, the extremely high numbers can also be a issue of overfitting, so we'll need to pay extra attention to that. But, through K-fold Cross-validation, we have decrease the danger of overfitting.

Let's focus on our top five performing models:

Show

```
## # A tibble: 5 × 7
##   neighbors .metric .estimator   mean     n std_err .config
##       <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
## 1        10 rmse    standard   0.0617    10 0.00379 Preprocessor1_Mode
110
## 2         9 rmse    standard   0.0622    10 0.00382 Preprocessor1_Mode
109
## 3         8 rmse    standard   0.0627    10 0.00384 Preprocessor1_Mode
108
## 4         7 rmse    standard   0.0634    10 0.00384 Preprocessor1_Mode
107
## 5         6 rmse    standard   0.0645    10 0.00385 Preprocessor1_Mode
106
```

We can observe the difference between using 10 neighbors and 6 neighbors is merely about 0.003, so I don't think it's worth increasing the neighbors to relatively high. However, let's use `select_by_one_std_err()` function to help us find the best model!

Show

```
## # A tibble: 1 × 9
##   neighbors .metric .estimator   mean     n std_err .config         .b
est .bound
##       <int> <chr>   <chr>       <dbl> <int>   <dbl> <chr>           <d
bl>  <dbl>
## 1        10 rmse    standard   0.0617    10 0.00379 Preprocessor1… 0.0
617 0.0655
```

It has selected 10 neighbors, so let's use this value to specify the previous unspecified `neighbors` argument in `knn_wkflow_cv` using `finalize_workflow()`:

Show

```
## == Workflow [trained] ==========================================
==========
## Preprocessor: Recipe
## Model: nearest_neighbor()
##
## -- Preprocessor ----------------------------------------------------
----------
## 2 Recipe Steps
##
## • step_interact()
## • step_interact()
##
## -- Model -----------------------------------------------------------
----------
##
## Call:
## kknn::train.kknn(formula = ..y ~ ., data = data, ks = min_rows(10L,
data, 5))
##
## Type of response variable: continuous
## minimal mean absolute error: 0.04441172
## Minimal mean squared error: 0.003938218
## Best kernel: optimal
## Best k: 10
```

Show

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0725
```

With a lower RMSE compared to the regular KNN models, it definitely performed better.

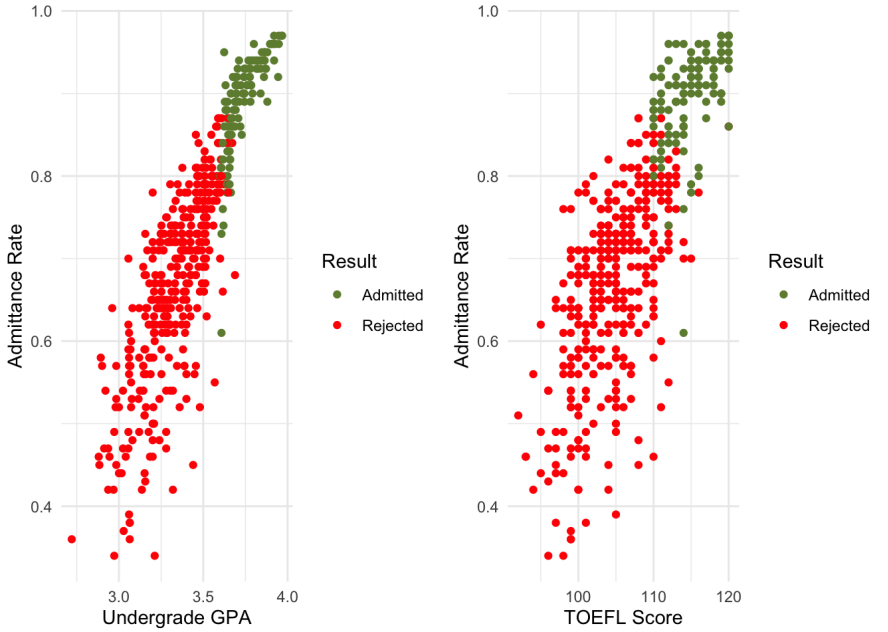
Classification Aproach

We have successfully build the model to anticipate the chance of admittance through the regression approach, let's categorize the result according to the criteria and compare our results through classification approach!

Show

Admittance between GPA, TOEFL Score and GRE Score

Show



We see the majority of the data result in red based on our criteria, but the result is reasonble since our overall admittance rate for this program is only 7%. The competition is fierce!

Model Building

Since we are analyzing the data from a differen approach, we need to set up a new recipe, while keeping others the same, we also need to exclude `Chance_of_Admit` and dummy code `Result`:

Show

```
## # A tibble: 399 × 10
##    GRE_Score TOEFL_Score University_Rating   SOP   LOR Research   GPA
Result
##        <dbl>       <dbl>             <dbl> <dbl> <dbl>    <dbl> <dbl>
<fct>
##  1       337         118                 4   4.5   4.5        1  3.86
Admitted
##  2       330         115                 5   4.5   3          1  3.74
Admitted
##  3       327         111                 4   4     4.5        1  3.6
Admitted
##  4       328         116                 5   5     5          1  3.8
Admitted
##  5       336         119                 5   4     3.5        1  3.92
Admitted
##  6       340         120                 5   4.5   4.5        1  3.84
Admitted
##  7       340         114                 5   4     4          1  3.84
Admitted
##  8       331         112                 5   4     5          1  3.92
Admitted
##  9       320         110                 5   5     5          1  3.68
Admitted
## 10       332         117                 4   4.5   4          0  3.64
Admitted
## # ℹ 389 more rows
## # ℹ 2 more variables: GPA_x_GRE_Score <dbl>, GPA_x_TOEFL_Score <dbl>
```

Logistic Regression

Let's specify a basic ***logistic regression*** for classification using the `glm` engine and create a responding workflow.

Show

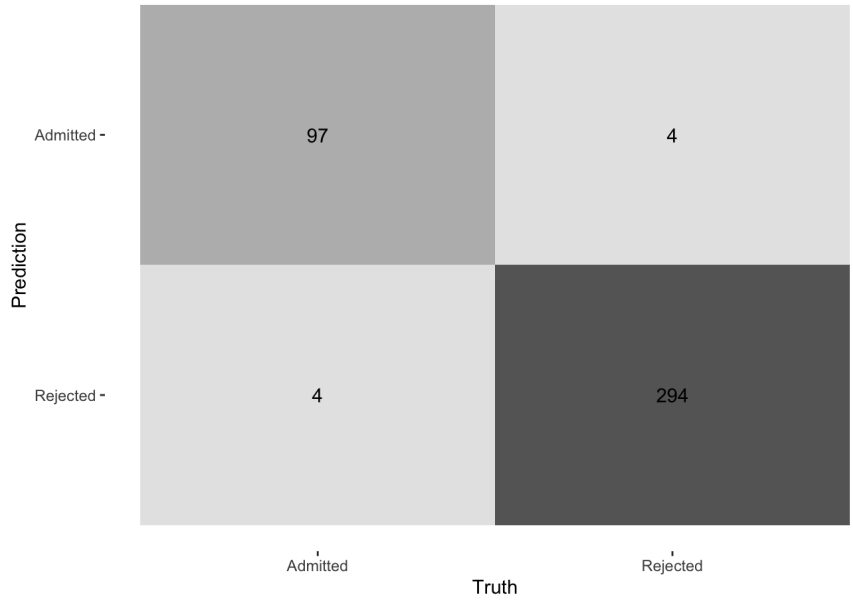
After fitting the training set to the logistic model, we can use `predict()` to assess the model's performance.

Show

```
## # A tibble: 399 × 2
##    .pred_Admitted .pred_Rejected
##             <dbl>          <dbl>
##  1          1.00        4.54e-13
##  2          1.00        1.28e- 7
##  3          0.260       7.40e- 1
##  4          1.00        1.18e-10
##  5          1           2.22e-16
##  6          1           2.22e-16
##  7          1.00        1.26e- 7
##  8          1.00        1.92e- 9
##  9          0.976       2.39e- 2
## 10          1.00        3.22e- 4
## # ℹ 389 more rows
```

From the above table, each row represents the probability predicted by the model that a given observation belongs to a certain class (admitted/rejected), however, the number in the tibble looks quite confusing and there are 399 rows, so we can try to summarize the predicted values using `augment()` and create a corresponding confusion matrix for better visualization:

Show



How to analyze a confusion matrix? As we can see the label on the axis is “Truth” and “Prediction” separately, so a good model will have more numbers if our predication matches the actual result. So, in this case, we have an incredible majority satisfying this condition. Let’s find out it’s precise accuracy:

Show

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.980
```

We have an approximately 98% accuracy of the logistic regression model, which is quite impressive, but I am a little worried about the model to be “overfitting”; thus, we will make the conclusion after seeing its performance on testing data.

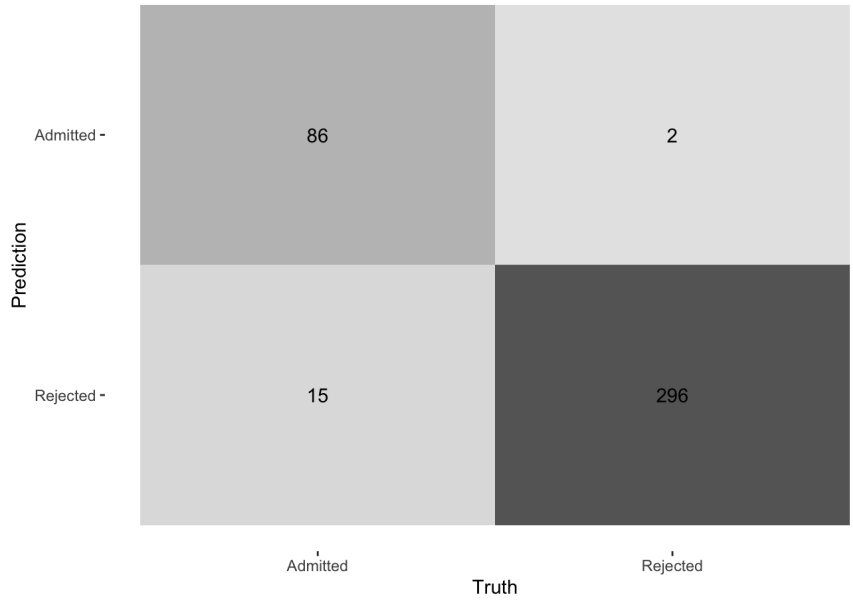
LDA

Setting up the LDA model:

Show

We will repeat the steps we did above to assess the model’s performance by constructing the confusion matrix and calcuating the accuracy.

Show



Show

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

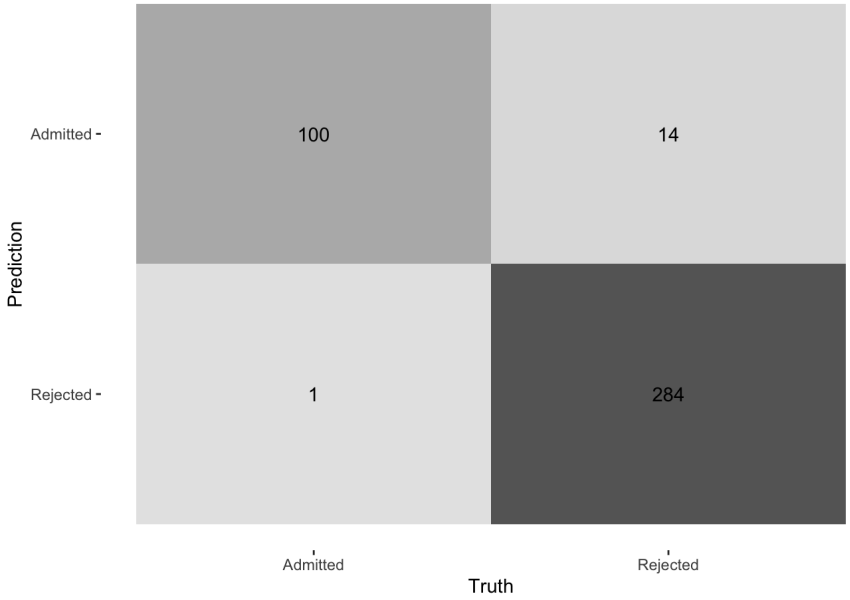
QDA

Setting up the QDA model:

Show

Confusion matrix and accuracy:

Show



Show

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.962
```

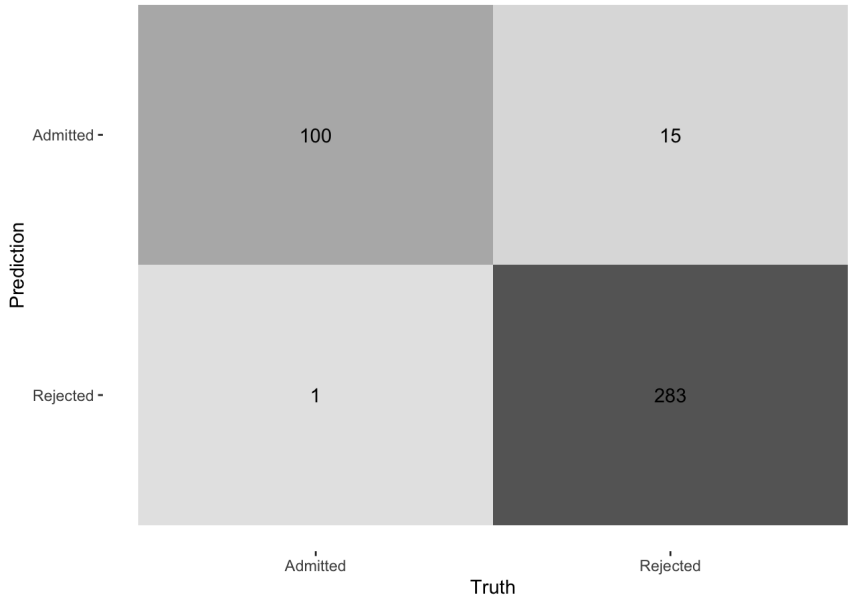
Naive Bayes

Setting up the Naive Bayes model:

Show

Confusion matrix and accuracy:

Show



Show

```
## # A tibble: 1 × 3
##   .metric  .estimator .estimate
##   <chr>    <chr>          <dbl>
## 1 accuracy binary         0.960
```

Model Comparison and Assessment

Let's put all the accuracies we got from each model above to compare them.

Show

```
## # A tibble: 4 × 2
##   accuracies models
##        <dbl> <chr>
## 1      0.980 Logistic Regression
## 2      0.962 QDA
## 3      0.960 Naive Bayes
## 4      0.957 LDA
```

Looks like the Logistic Regression model and QDA model has the highest accuracy, so we will fit these two models to the testing dataset.

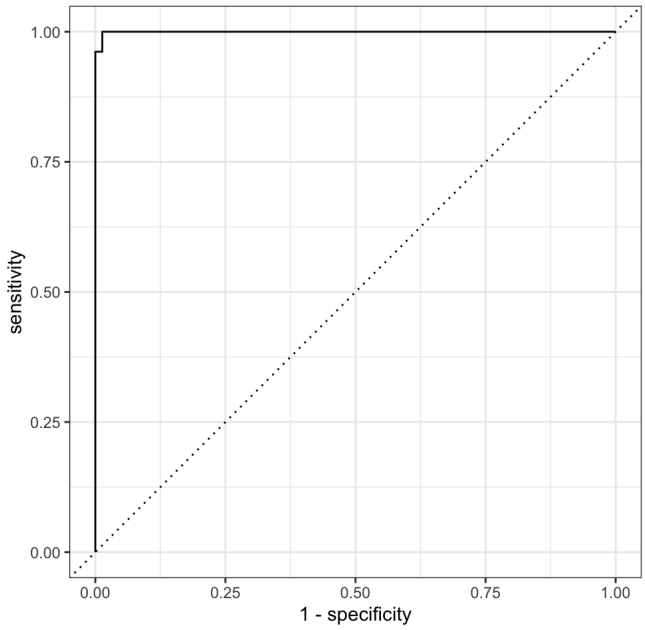
Testing Data

1. For Logistic Regression:

Show

```
##            Truth
## Prediction Admitted Rejected
##   Admitted       25        1
##   Rejected        1       74
```

Show

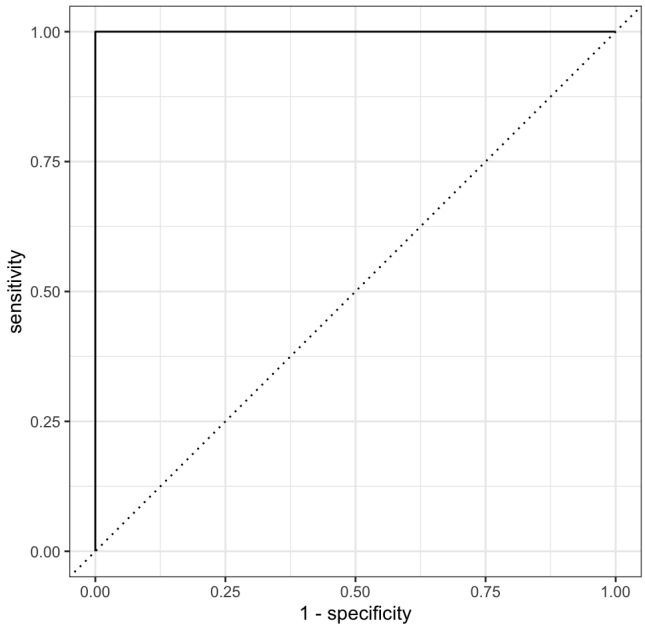


2. For QDA:

Show

```
##           Truth
## Prediction Admitted Rejected
##   Admitted       26        1
##   Rejected        0       74
```

Show



As we observe the confusion matrix and the ROC curve, QDA actually performs slightly better than logistic regression on testing data.

Regression or Classification

So should we choose the regression approach or the classification approach? Let's find out!!!

Elastic Net Regression

We can use the previous recipe we created and the folds for regression, but we need to create a fold set for classification.

Show

Next, we are trying to set up an elastic net regression for each method followed by the creation of a corresponding workflow:

Show

Using the resampled objects previously, let's try it using the hyperparameter tuning to determine the performance of the models. To avoid iteracy, we can create just one grid that's usable for both approach.

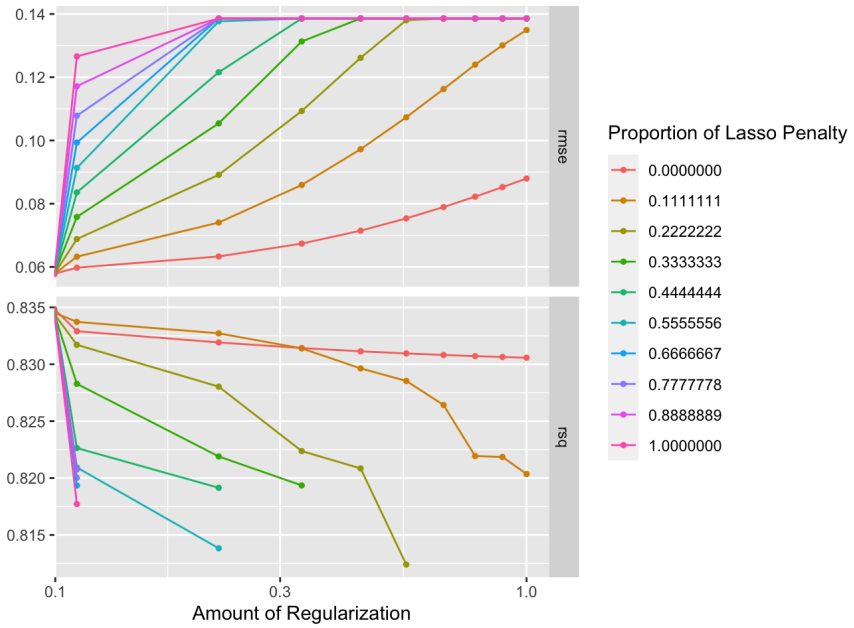
Show

Exciting! Let's fit all those models to our data! To save time for processing, I have used `save()` and `write_rds()`.

Show

Let's interpret our regression data first:

Show



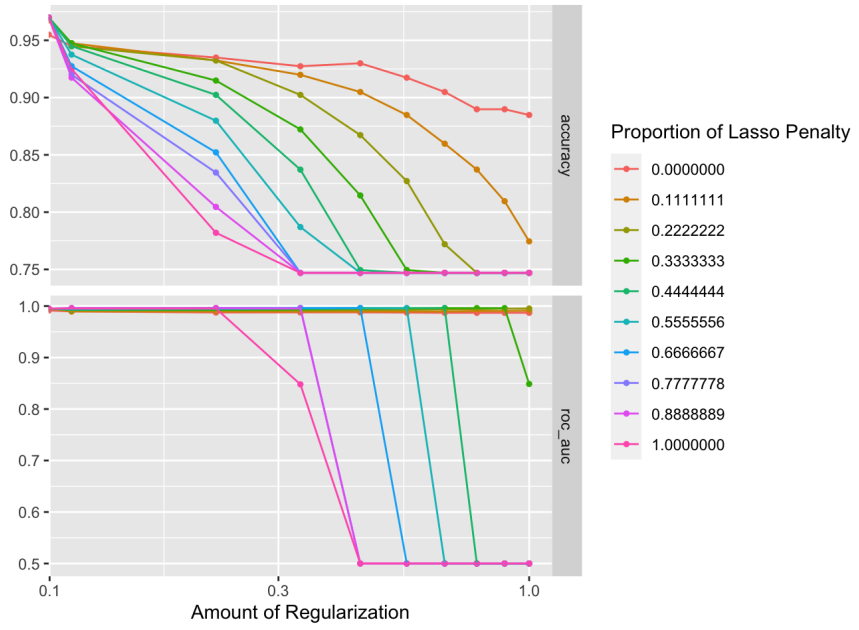
Show

```
## # A tibble: 200 × 8
##    penalty mixture .metric .estimator   mean     n std_err .config
##      <dbl>   <dbl> <chr>   <chr>       <dbl> <int>   <dbl> <chr>
##  1   0           0 rmse    standard   0.0578    10 0.00398 Preprocesso
r1_Model0…
##  2   0           0 rsq     standard   0.835     10 0.0187  Preprocesso
r1_Model0…
##  3   0.111       0 rmse    standard   0.0598    10 0.00407 Preprocesso
r1_Model0…
##  4   0.111       0 rsq     standard   0.833     10 0.0181  Preprocesso
r1_Model0…
##  5   0.222       0 rmse    standard   0.0633    10 0.00420 Preprocesso
r1_Model0…
##  6   0.222       0 rsq     standard   0.832     10 0.0180  Preprocesso
r1_Model0…
##  7   0.333       0 rmse    standard   0.0674    10 0.00431 Preprocesso
r1_Model0…
##  8   0.333       0 rsq     standard   0.831     10 0.0179  Preprocesso
r1_Model0…
##  9   0.444       0 rmse    standard   0.0715    10 0.00441 Preprocesso
r1_Model0…
## 10   0.444       0 rsq     standard   0.831     10 0.0179  Preprocesso
r1_Model0…
## # ℹ 190 more rows
```

The x-axis shows the ***amount of regularization*** which is the penalty hyperparameter that covers the scope of values we indicated (0 to 1), and the upsides of combination are addressed by the different-hued lines. As we can observe from the scale of our y-axis for both metrics (RMSE and R^2), the range is relatively small which means the variation of the resulting performance between models is very small.

Now, let's take a look at how our classification model has performed:

Show



Show

```
## # A tibble: 200 × 8
##    penalty mixture .metric  .estimator  mean     n std_err .config
##      <dbl>   <dbl> <chr>    <chr>      <dbl> <int>   <dbl> <chr>
##  1   0           0 accuracy binary     0.955    10 0.00970 Preprocesso
r1_Model0…
##  2   0           0 roc_auc  binary     0.991    10 0.00271 Preprocesso
r1_Model0…
##  3   0.111       0 accuracy binary     0.945    10 0.0110  Preprocesso
r1_Model0…
##  4   0.111       0 roc_auc  binary     0.990    10 0.00327 Preprocesso
r1_Model0…
##  5   0.222       0 accuracy binary     0.935    10 0.0130  Preprocesso
r1_Model0…
##  6   0.222       0 roc_auc  binary     0.987    10 0.00384 Preprocesso
r1_Model0…
##  7   0.333       0 accuracy binary     0.927    10 0.0120  Preprocesso
r1_Model0…
##  8   0.333       0 roc_auc  binary     0.988    10 0.00376 Preprocesso
r1_Model0…
##  9   0.444       0 accuracy binary     0.930    10 0.0128  Preprocesso
r1_Model0…
## 10   0.444       0 roc_auc  binary     0.988    10 0.00376 Preprocesso
r1_Model0…
## # ℹ 190 more rows
```

For the ***classification*** dataset, the scale for the y-axis for both metrics (accuracy and ROC AUC) is relatively large compared to the plot ***regression*** especially ROC AUC, it does not change much between 0.1 and 0.3, but between 0.3 and 1.0, the value change drastically.

Model Performance Selection

Show

```
## # A tibble: 1 × 10
##   penalty mixture .metric .estimator   mean     n std_err .config   .b
est .bound
##     <dbl>   <dbl> <chr>   <chr>       <dbl> <int>   <dbl> <chr>     <d
bl>  <dbl>
## 1       0       0 rmse    standard   0.0578    10 0.00398 Preproc… 0.0
578 0.0618
```

Show

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0579
```

We can see that the RMSE in the testing dataset is smaller than the result from K-fold cross validation. And it has a similar RMSE to linear regression.

Show

```
## # A tibble: 1 × 10
##   penalty mixture .metric .estimator  mean     n std_err .config    .b
est .bound
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>      <d
bl>  <dbl>
## 1       0   0.111 roc_auc binary     0.995    10 0.00257 Preproces… 0.
996  0.993
```

Show

```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 roc_auc binary         0.997
```

It also performances better for classification approach with extremely high ROC AUC.

Random Forest

Instead of building a single decision tree, I decide to construct a random forest that includes a set of decision trees. But we need to make some changes based on previous recipe that it is neccessary to normalize all predictors:

Show

Then, let's set up two separate models and workflow, one for regression (`rf_reg_spec` and `rf_reg_wf`) and one for classification (`rf_class_spec` and `rf_class_wf`) accordingly. And let's flag three hyperparameters for tunning, `mtry` , `trees` and `min_n` .

Show

After having these three hyperparameters, we need to set up a grid for them to consider:

Show

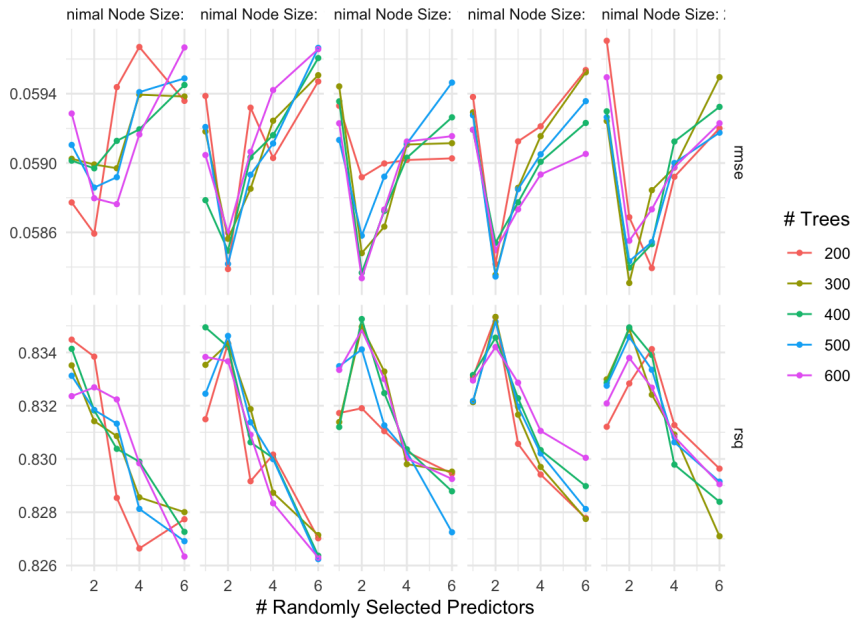
```
## # A tibble: 125 × 3
##    mtry trees min_n
##   <int> <int> <int>
## 1     1   200    10
## 2     2   200    10
## 3     3   200    10
## 4     4   200    10
## 5     6   200    10
## 6     1   300    10
## 7     2   300    10
## 8     3   300    10
## 9     4   300    10
## 10    6   300    10
## # ℹ 115 more rows
```

Next, we need to fit all the random forest models we've specified in the previous code chunk to each dataset. Since it takes up a long time to run the code, I have saved the results to two files for each model.

Show

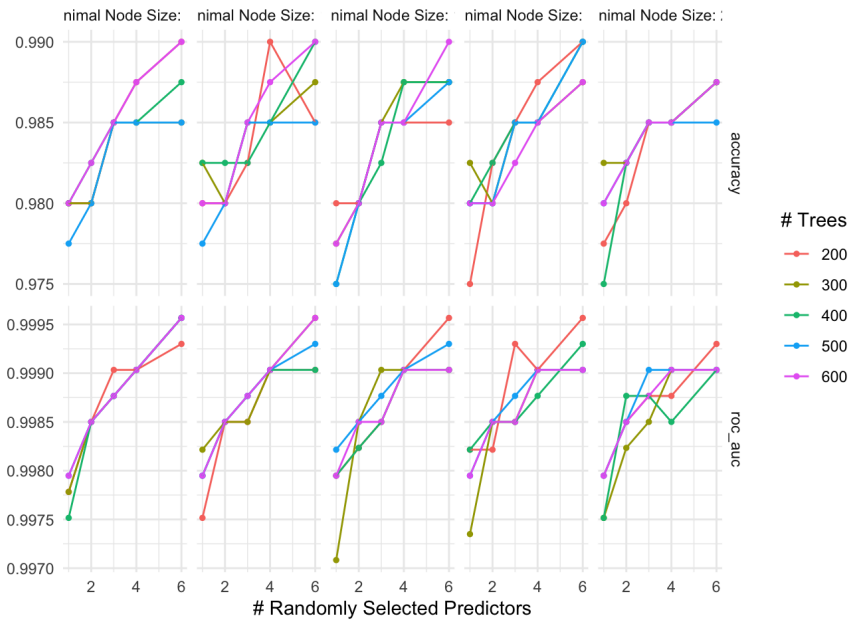
Now, we can plot the model results and take a look at them:

Show



This plot shows the result for regression dataset that the lowest RMSE result when the randomly selected predictors equals two and R^2 tends to decrease as we increase `mtry` . The number of `trees` , as indicted in the legend, does not make much of a difference overall that these lines seems to have the same trend. The smallest RMSE and the highest R^2 yields from a minimal node size of 20 that seems to produce slightly better results than a minimum size of 10.

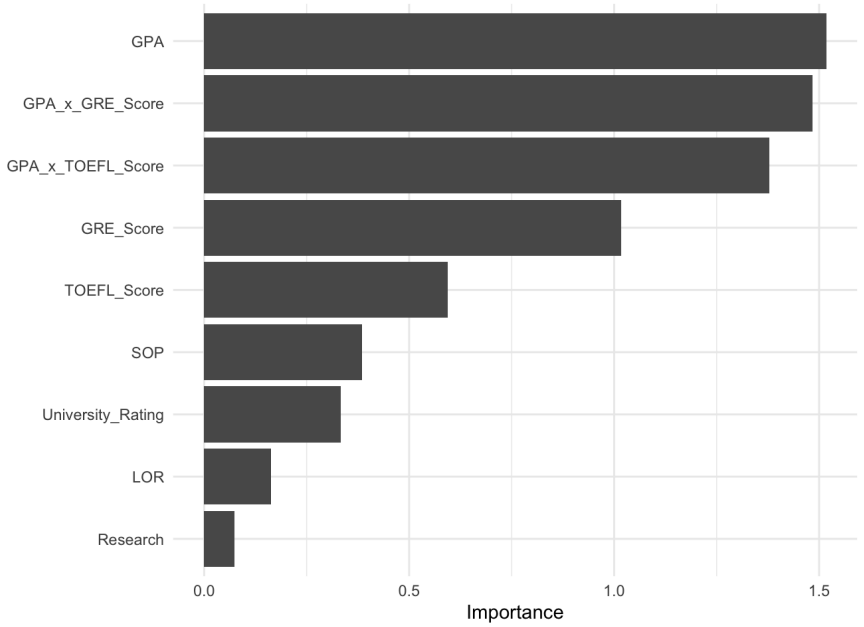
Show



This plot shows the result for the classification dataset, unlike our results from regression, our performances seems to improve as we increase `mtry` that the accuracy and ROC AUC begins to increase. The variation between trees is not drastic as well. A minimal node size of 10 tends to produce slightly better results than the minimum size of 20.

After a brief analyze, we'll need to select the optimal random forest model for each dataset and fit each of those to the entire training sets respectively, and then use `extract_fit_parsnip()` and `vip()` to create and view the variable importance plot, thus decide which is the best approach to handle the data:

Show

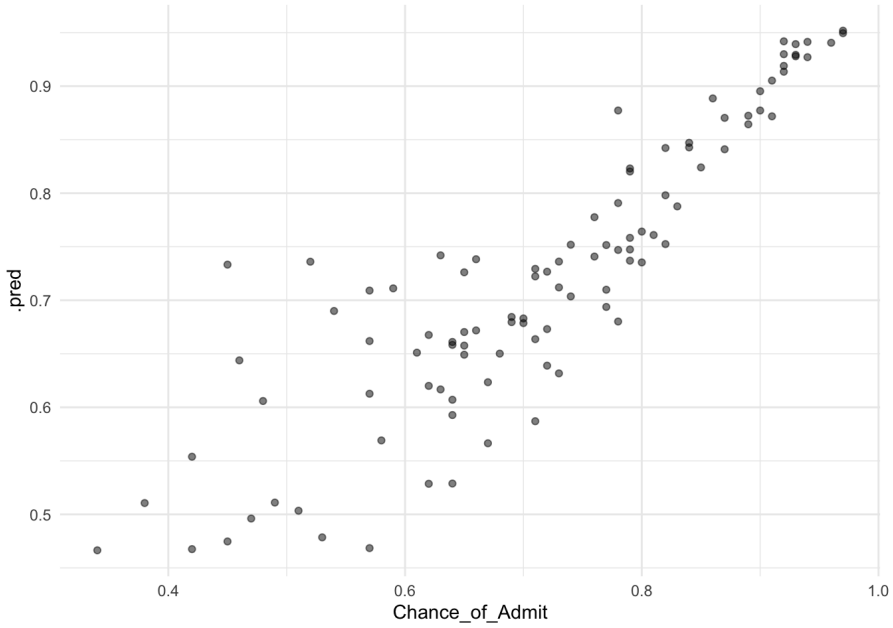


This graph shows us the three most useful predictors of `Chance_of_Admit` in this model are the GRE Score, GPA and TOEFL Score. Let's see its performance on the testing data by checking out its resulted RMSE and further create a scatterplot comparing the actual values and the predicted values:

Show

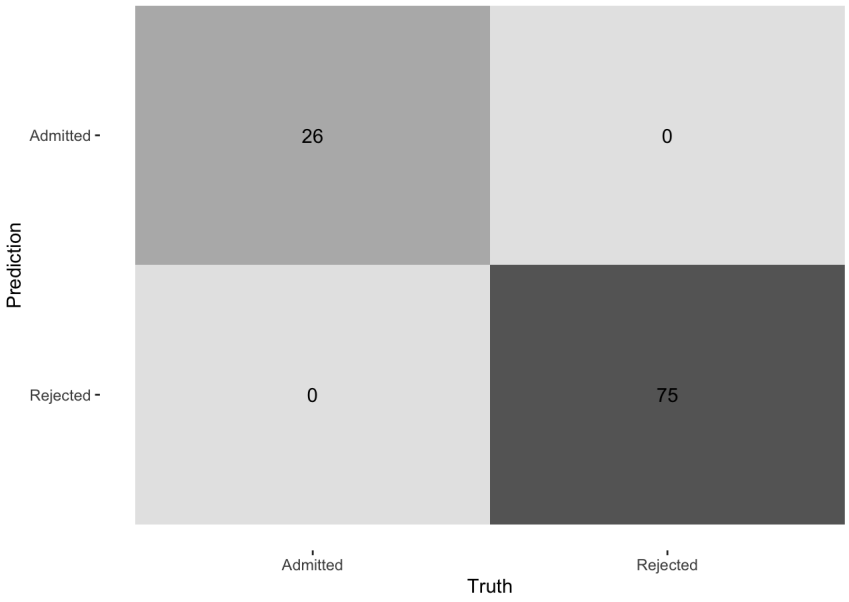
```
## # A tibble: 1 × 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard      0.0689
```

Show



The RMSE is actually smaller than the RMSE we had for the testing data resulted from other models. Now, let's take a look at our classification data:

Show



Our model for classification has done an incredible job without having any wrong predictions!

Conclusion - Best Model

By comparing the results from both approach under the same model, I think the best approach to analyze this dataset is by classification. No matter under what kind of model, it will always generate a high ROC AUC and accuracy, but the best is definitely through random trees model.

Congratulations! We are done! Good luck to all the graduates on their master application!

Concerns

The dataset does not contain any missing value, however, there are bias in the original source data, since the ranking scale on both `University Rating` and `SOR` (Statement of Purpose and Letter of Recommendation Strength) are not stated. Furthermore, the data were collected for prediction was conducted from an Indian perspective, so the prediction will be limited/narrowed.

Learnings

The course and project I have undertaken have served as my introduction to the field of machine learning. It has solidified my determination to pursue postgraduate studies in this domain. As elucidated in the introduction, this project has not only aided me in predicting my personal probability of admission but has also instilled in me the aspiration to apply similar functionality on platforms like “U.S. News,” akin to the “College Admissions Calculator” that you can find out your whether or not you are competitive by inputting your score and comparing it with the past result.

Furthermore, I think the reason that the result from classification approach are more precise because the dataset was categorized based exactly on last year data, but since it only provided us the average score of each criteria rather than the specific background information of each applicants, we cannot create a perfect model. The models will be more precise if we have access to the dataset.

Moreover, there are still many aspects to make improvements on for this project, for example, the categories made were limited. The result is not only limited to “Admitted” or “Rejected”, it can also be “Waitlisted” that gives applicants more hope in getting in since there are some students are not limited applying to just one college that might give up this admittance and choose to go to another one they prefer.

This project is extremely inspiring for me that it introduced me to various models and taught me how to analyze them. And I wish to pursue deeper learning in model constructing later on.