Admission Prediction Report

Paris Rerkshanandana

6/15/2020

Overview

In this project, I will be predicting the chance of admission of graduate students in the Graduate-Admission-2 dataset (links: https://www.kaggle.com/mohansacharya/graduate-admissions). I will be trying various machine-learning algorithms to see which approach works best with the dataset and how I can improve it. I will implement linear Regression Model, KNN, Random Forest, and some other models and ensembles them together. This report will go through various steps I took to build the final model for the project.

Data Wrangling

First, I downloaded the dataset using read_csv() and stored it in dat.

```
tmp=tempfile()
download.file("https://raw.githubusercontent.com/PRerk/Graduate-Admission-Prediction/master/datasets_14
dat <- read_csv(tmp)
file.remove(tmp)
```

Then I look at the structure of the data.

```
str(dat)
```

```
## tibble [500 x 9] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ Serial No.       : num [1:500] 1 2 3 4 5 6 7 8 9 10 ...
##  $ GRE Score        : num [1:500] 337 324 316 322 314 330 321 308 302 323 ...
##  $ TOEFL Score      : num [1:500] 118 107 104 110 103 115 109 101 102 108 ...
##  $ University Rating: num [1:500] 4 4 3 3 2 5 3 2 1 3 ...
##  $ SOP              : num [1:500] 4.5 4 3 3.5 2 4.5 3 3 2 3.5 ...
##  $ LOR              : num [1:500] 4.5 4.5 3.5 2.5 3 3 4 4 1.5 3 ...
##  $ CGPA             : num [1:500] 9.65 8.87 8 8.67 8.21 9.34 8.2 7.9 8 8.6 ...
##  $ Research         : num [1:500] 1 1 1 1 0 1 1 0 0 0 ...
##  $ Chance of Admit  : num [1:500] 0.92 0.76 0.72 0.8 0.65 0.9 0.75 0.68 0.5 0.45 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   `Serial No.` = col_double(),
##   ..   `GRE Score` = col_double(),
##   ..   `TOEFL Score` = col_double(),
##   ..   `University Rating` = col_double(),
##   ..   SOP = col_double(),
##   ..   LOR = col_double(),
##   ..   CGPA = col_double(),
```

```
##   ..   Research = col_double(),
##   ..   `Chance of Admit` = col_double()
##   .. )
```

Viewing the first few row of the data.

```
head(dat)
```

```
## # A tibble: 6 x 9
##   `Serial No.` `GRE Score` `TOEFL Score` `University Rat~   SOP   LOR  CGPA
##          <dbl>       <dbl>         <dbl>             <dbl> <dbl> <dbl> <dbl>
## 1            1         337           118                 4   4.5   4.5  9.65
## 2            2         324           107                 4   4     4.5  8.87
## 3            3         316           104                 3   3     3.5  8
## 4            4         322           110                 3   3.5   2.5  8.67
## 5            5         314           103                 2   2     3    8.21
## 6            6         330           115                 5   4.5   3    9.34
## # ... with 2 more variables: Research <dbl>, `Chance of Admit` <dbl>
```

And checks if any row contain NA.

```
sum(!complete.cases(dat))
```

```
## [1] 0
```

After Inspecting the dataset, The data looks well-formatted, and there seem to be no missing values in it. here all features are numeric and can be passed on to our model directly without any transformation. I will now divide the data into a train set and test set using this code.

```
set.seed(1,sample.kind = "Rounding")
test_index <- createDataPartition(y = dat$`Chance of Admit`, times = 1, p = 0.2,
                                  list = FALSE)
train_set <- dat[-test_index,]
test_set <- dat[test_index,]
```

Here I partition 20% of data as the test set and rest is used for training.

Loss function

The loss function I am going to use here is the residual mean squared error (RMSE). I will be estimating the performance of the models based on the residual mean squared error (RMSE) calculated on the test set.

```
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

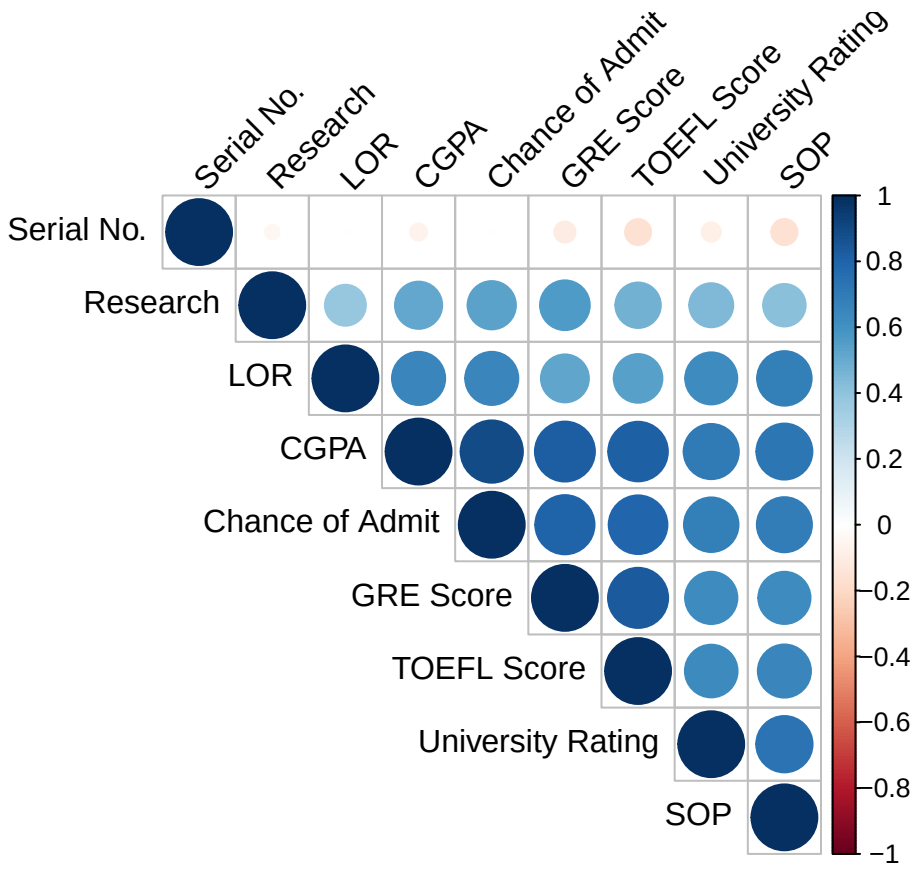
Features Analysis

Now I will compute the Correlation Matrix on the training set to see the dependence between the features of the data.

```r
res <- cor(train_set)
round(res, 2)
```

```
##                    Serial No. GRE Score TOEFL Score University Rating   SOP
## Serial No.               1.00     -0.10       -0.16             -0.09 -0.16
## GRE Score               -0.10      1.00        0.83              0.63  0.63
## TOEFL Score             -0.16      0.83        1.00              0.64  0.65
## University Rating       -0.09      0.63        0.64              1.00  0.73
## SOP                     -0.16      0.63        0.65              0.73  1.00
## LOR                     -0.01      0.53        0.55              0.62  0.69
## CGPA                    -0.07      0.83        0.82              0.70  0.72
## Research                -0.05      0.57        0.48              0.45  0.42
## Chance of Admit          0.01      0.81        0.79              0.69  0.70
##                      LOR  CGPA Research Chance of Admit
## Serial No.         -0.01 -0.07    -0.05            0.01
## GRE Score           0.53  0.83     0.57            0.81
## TOEFL Score         0.55  0.82     0.48            0.79
## University Rating   0.62  0.70     0.45            0.69
## SOP                 0.69  0.72     0.42            0.70
## LOR                 1.00  0.65     0.39            0.65
## CGPA                0.65  1.00     0.51            0.89
## Research            0.39  0.51     1.00            0.54
## Chance of Admit     0.65  0.89     0.54            1.00
```

```r
corrplot(res, type = "upper", order = "hclust",
         tl.col = "black", tl.srt = 45)
```



It can be seen that the chance of admission has a strong positives correlation with CGPA, GRE score, and TOEFL score. This is apparent since higher-scoring students tends to get admitted more easily.

predictive modeling for 'chance of admission'

Linear Regression

First, I will implement Linear Regression model on the dataset using the train() function of the caret package.

```
train_lm <- train(`Chance of Admit`~.,
                  data = train_set,
                  method = "lm"
                  )
pred_lm <- predict(train_lm, test_set)
rmse_lm <- RMSE(test_set$`Chance of Admit`,pred_lm)
rmse_results <- tibble(method = "lm", RMSE = rmse_lm)
```

method	RMSE
lm	0.0631181

We get the RMSE of 0.0631181. lets inspect the first few row of predicted value and true value.

```
## [1] 0.6069472 0.5715732 0.6389929 0.9427007 0.5702240 0.8203508
```

```
## [1] 0.65 0.68 0.62 0.94 0.48 0.88
```

I think Linear Regression perform quite well on this dataset.

k-nearest neighbors

```
train_knn <- train(`Chance of Admit`~.,
                   data = train_set,
                   method = "knn",
                   tuneGrid = data.frame(k = c(3,5,7,9,11)))
pred_knn <- predict(train_knn, test_set)
rmse_knn <- RMSE(test_set$`Chance of Admit`,pred_knn)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="knn",
                                     RMSE = rmse_knn ))
```

method	RMSE
lm	0.0631181
knn	0.0668677

Using K-nn we get RMSE of 0.0668677.

K-nn while tuning with different value of K still gives the same RMSE as the default values. The RMSE of knn is comparetivly higher than that of Linear Regreesion and so its not a very good model to use with this dataset.

Random forests

```
train_rf <- train(`Chance of Admit`~.,
                   data = train_set,
                   method = "rf")
pred_rf <- predict(train_rf, test_set)
rmse_rf <- RMSE(test_set$`Chance of Admit`,pred_rf)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="rf",
                                 RMSE = rmse_rf ))
```

method	RMSE
lm	0.0631181
knn	0.0668677
rf	0.0564256

We get RMSE as low as 0.0564256 using rf. we can see that Random Forests clearly outperformed k-nn and linear regression here.

Artificial neural network

Artificial neural networks have grabbed a lot of attention recently. A neural network can be designed to detect patterns in input data and produce an output free of noise. I will be implementing an Artificial neural network using nnet method of train() function of the caret package.

```
train_nnet <- train(`Chance of Admit`~.,
                    data = train_set,
                    method = "nnet")
pred_nnet <- predict(train_nnet, test_set)
```

```
rmse_nnet <- RMSE(test_set$`Chance of Admit`,pred_nnet)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="nnet",
                                 RMSE = rmse_nnet ))
```

method	RMSE
lm	0.0631181
knn	0.0668677
rf	0.0564256
nnet	0.0637076

Using nnet, I get varying results of RMSE between 5 to 8. This is likely due to lack of data since Neural Networks require much more data than traditional Machine Learning algorithms to do their job well. So I would say nnet is not a very reliable model to use here.

Ensemble: Linear Regression and Random Forests

We can see that Random Forest gives significantly better performance than the k-nn and the Linear Regression model. I will ensemble Random Forest and Linear regression to see if I can get better results with the new model. The chance of admission of the ensemble model will be calculated by averaging the predicted values of Random Forest and Linear Regression. k-nn and nnet are not ensembled here because k-nn did not perform very well with this dataset and nnet is not at all reliable here.

```
pred_en <- (pred_lm+pred_rf)/2
rmse_en <- RMSE(test_set$`Chance of Admit`,pred_en)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="ensemble: lm + rf",
                                 RMSE = rmse_en ))
```

method	RMSE
lm	0.0631181
knn	0.0668677
rf	0.0564256
nnet	0.0637076
ensemble: lm + rf	0.0587080

With RMSE of 0.0587080, our new model performed better than Linear Regression but still not better than the Random Forests. We did not succeed in improving the model so I will be trying some more different models.

Since the Random Forests performed very well with this dataset, I will be try two models similar to Random Forests.

They are

1. Gradient Boosted Model
2. Support Vector Machine

Gradient Boosted Model

The Gradient Boosted Model produces a prediction model composed of an ensemble of decision trees. The distinguishing characteristic of the GBM is that it builds its trees one tree at a time. Each new tree helps to correct errors made by the previously trained tree‘—unlike in the Random Forest model, in which the trees bear no relation.

```
train_gbm <- train(`Chance of Admit`~.,
                   data = train_set,
                   method = "gbm")
pred_gbm <- predict(train_nnet, test_set)
rmse_gbm <- RMSE(test_set$`Chance of Admit`,pred_gbm)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="gbm",
                                 RMSE = rmse_gbm ))
```

method	RMSE
lm	0.0631181
knn	0.0668677
rf	0.0564256
nnet	0.0637076
ensemble: lm + rf	0.0587080
gbm	0.0637076

GBM gives the RMSE of 0.0637076 which is still far from what we get with the Random Forests.

Support Vector Machine

A Support Vector Machine (SVM), given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane that categorizes new examples.

```
train_svm <- train(`Chance of Admit`~.,
                   data = train_set,
                   method = "svmLinear")

pred_svm <- predict(train_svm, test_set)
rmse_svm <- RMSE(test_set$`Chance of Admit`,pred_svm)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="svm linear",
                                 RMSE = rmse_svm ))
```

method	RMSE
lm	0.0631181
knn	0.0668677
rf	0.0564256
nnet	0.0637076
ensemble: lm + rf	0.0587080
gbm	0.0637076
svm linear	0.0632642

Here we get RMSE of 0.0632642, a bit lower than that of GBM but we still a lot higher than the Random forests.

Support vector machine methods can handle both linear and non-linear class boundaries. the SVM algorithm can perform a non-linear classification using the kernel trick. The most commonly used kernel transformations are polynomial kernel and radial kernel. We will try using a radial kernel.

```
train_svm_radial <- train(`Chance of Admit`~.,
                   data = train_set,
                   method = "svmRadial")
```

```
pred_svm_radial <- predict(train_svm_radial, test_set)
rmse_svm_radial <- RMSE(test_set$`Chance of Admit`,pred_svm_radial)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="svm non linear - radial",
                                 RMSE = rmse_svm_radial ))
```

method	RMSE
lm	0.0631181
knn	0.0668677
rf	0.0564256
nnet	0.0637076
ensemble: lm + rf	0.0587080
gbm	0.0637076
svm linear	0.0632642
svm non linear - radial	0.0579285

Here we get RMSE of 0.0579285 which is almost as good as with the Random Forests we can see that it performed significantly better than other models.

I won't be implementing SVM using polynomial kernel here since it gives similar results to other models and not much improvement over others.

the code for executing SVM using polynomial kernel is given below:

```
train_svm_Poly <- train(`Chance of Admit`~.,
                          data = train_set,
                          method = "svmPoly")
```

Ensemble: Random Forests and SVM Radial

I will ensemble the two best performing models i.e Random Forests and SVM Radial. These two models performed much better than the rest. We will see if the new model will perform better than Random Forests.

```
pred_en2 <- (pred_rf+pred_svm_radial)/2
rmse_en2 <- RMSE(test_set$`Chance of Admit`,pred_en2)
rmse_results <- bind_rows(rmse_results,
                          tibble(method="ensemble rf + svm radial",
                                 RMSE = rmse_en2 ))
```

method	RMSE
lm	0.0631181
knn	0.0668677
rf	0.0564256
nnet	0.0637076
ensemble: lm + rf	0.0587080
gbm	0.0637076
svm linear	0.0632642
svm non linear - radial	0.0579285
ensemble rf + svm radial	0.0562957

Our new model clearly outperformed all our previous model. With RMSE of 0.0562957, it performed slightly better than Random Forests.

Results

The results of all the models implemented in this project are shown below

method	RMSE
lm	0.0631181
knn	0.0668677
rf	0.0564256
nnet	0.0637076
ensemble: lm + rf	0.0587080
gbm	0.0637076
svm linear	0.0632642
svm non linear - radial	0.0579285
ensemble rf + svm radial	0.0562957

We can see that our new model and Random forests performed very well with this dataset compared to other models with most of them having RMSE of above 6.

Inspecting rows of true value and predicted value.

```
## [1] 0.6019692 0.5883987 0.6068549 0.9394004 0.5874137 0.8475858
```

```
## [1] 0.65 0.68 0.62 0.94 0.48 0.88
```

Our predicted value here is quite close to the true value and is good enough to be used in real-life scenarios.

I have tried testing the model with multiple seeds and found out that our new ensembled model and random forests outperformed all models with some of the time random forest performing better and vice versa.

conclusion

Finally, I would conclude that the new model performed quite well in this scenario and can be implemented in the real world for helping students estimate their chance of admission to college. The limitation I found is that the data set is too small to be used on most models. In the future, I will try to expand the dataset by combining data from other sources online as well as trying out other different models and try to improve upon it.