Optimizing a Binary Logistic Regression model by Hyperparameter tuning

Arnav Oberoi, Om Sehgal, Chirag Malik

Abstract— Everyone wants to get into the best university they could think of for their master's degree but considered the different requirements considered by the universiy's admission committee, it becomes extremely difficult to predict whether they would get into the university or not. Our model will predict whether the student can get into the university and using this model we will be trying to tune the parameters and get the best Logistic Regression model for ideal results. Given various tuning techinques, we will try to find the best tuning method for our model. Learning the maths behind the tuning techniques and model, we will get a clearer idea about how the model works and how we will be improving its overall performance.

Index Terms— Machine learning, Model Validation and Analysis, Parameter learning, Logistic Regression

DOI: 10.29322/IJSRP.13.07.2023.p13913
http://dx.doi.org/10.29322/IJSRP.13.07.2023.p13913

Paper Received Date: 14th May 2023
Paper Acceptance Date: 25th June 2023
Paper Publication Date: 6th July 2023

1 Introduction

MACHINE Learning is the field of study that imparts the ability to automate learning and allows machines to learn from the data itself. Machine Learning is tools and technology that use data to answer questions. We have used SciKit Learn's Logistic Regression algorithm for our dataset. SciKit Learn is a free software machine learning library which was a project developed by David Cournapeau under Google Summer of Code in 2007. It is a third-party extension to SciPy. Logistic Regression is used to predict the probability of an event. Logistic function or Sigmoid function was developed by statisticians to describe the growth in population. It is like an S shaped curve that takes any numeric value into account and maps it into a value between 0 and 1 both inclusive. The mathematics behind the Logistic Regression algorithm is related to that of Linear Regression. Linear Regression can predict the exam score of a student between 0 to 100 whereas Logistic Regression can predict whether the student passed or failed the exam. There are three types of Logistic Regression: Binary Logistic Regression, Multinomial Logistic Regression, Ordinal Logistic Regression. In this work we will be using Binary Logistic Regression and will use techniques to increase its accuracy, precision, f1-score, recall and overall performance. While making a machine learning model, one can alter the learning rates and contraints to optimize the model's performance and these parameters are called hyperparameters. Hyperparameter are parameters whose values are used during the process of learning. The dataset chosen for our model regards with the prediction of the chance of admit for a student for his/her master's degree.

2 Dataset Overview

The dataset chosen for our model is available on kaggle and has more then 31,000+ downloads and 216,000+ views at the time of this work. The dataset is maintained by Mohan S. Acharya and was used by him and the co-authors in their research work. The dataset consists of 8 columns and 500 instances.

3 Exploratory Data Analysis

To build a model it is necessary to understand the data in the best possible way.

3.1 Understanding the columns

The eight columns excluding the Serial No. column are GRE Score (out of 340), TOEFL Score (out of 120), University Rating (out of 5), Statement of Pupose (out of 5), Letter of Recommendation (out of 5), CGPA (out of 10), Research (0 or 1) and Chance of Admit (between 0 and 1). GRE is an exam to check how well prepared a student is for graduate-level course work. TOEFL is an English proficieny exam. University Rating is the rating of the student's Bachelor university. Statement of Purpose is one of the key highlights of one's application which tells the admission committee why the student has chosen that career path and basis that the score has been given. Letter of Recommendation is provided by the bachelor's university and so the score has been provided. CGPA is the score of the student in the university. In the Research column 1 is means that the student has research experience whereas 0 is for no experience.

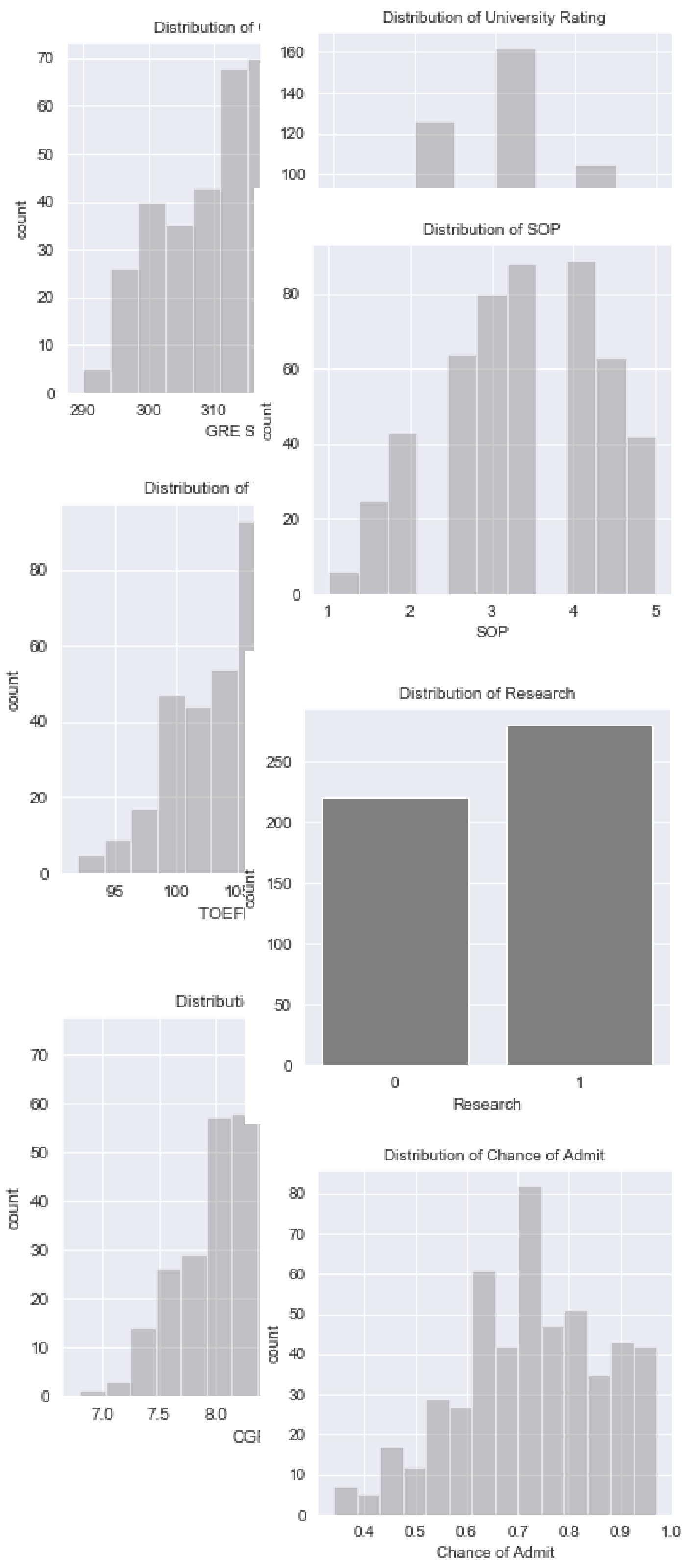
3.2 Classifying the variables

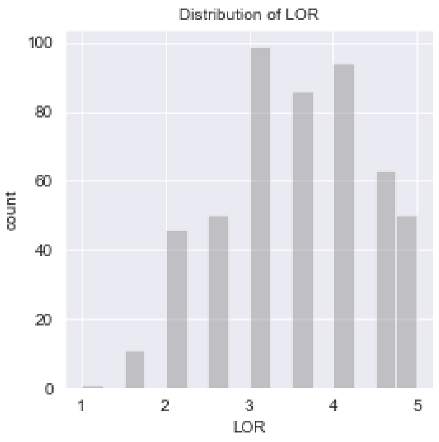
In our dataset we have 7 independent variables (GRE Score, TOEFL Score, University Rating, Statement of Purpose, Letter of Recommendation, CGPA and Research) and 1 dependent variable (Chance of Admit). We will be using the 7 independent variables to predict the dependent variable.

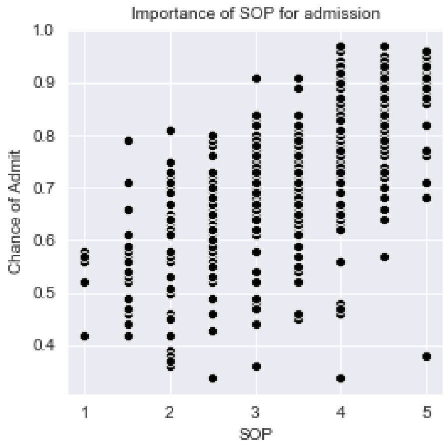
3.3 Variable Analysis

We tried to find the mean of each variable. For Research we have seen how many people have done research and how mant haven't. We then went to visualize the distribution using the seaborn's displot for each variable. Doing this we got a clear idea of how our data has been organized. For Research we used Seaborn's countplot.

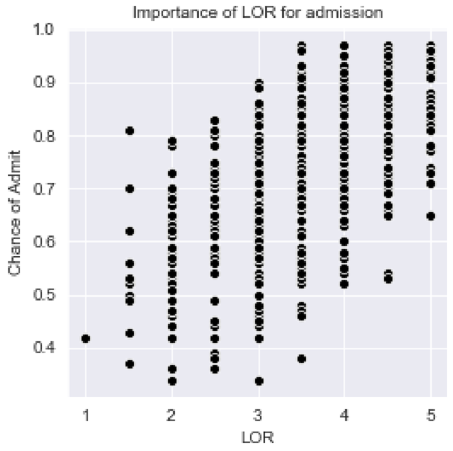
The mean of GRE Score: 316.47/340
The mean of TOEFL Score: 107.19/120
The mean of SOP: 3.37/5
The mean of LOR: 3.48/5
The mean of CGPA: 8.58/10
The mean of University Rating: 3.11/5
The mean of Chance of Admit: 0.72/1

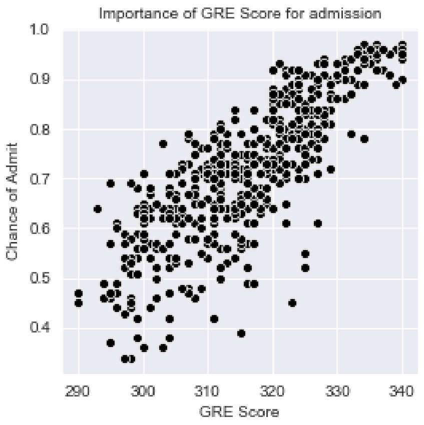


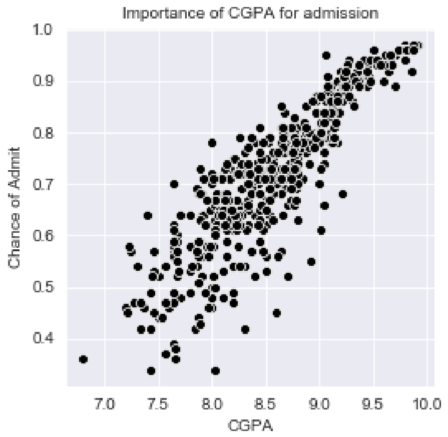


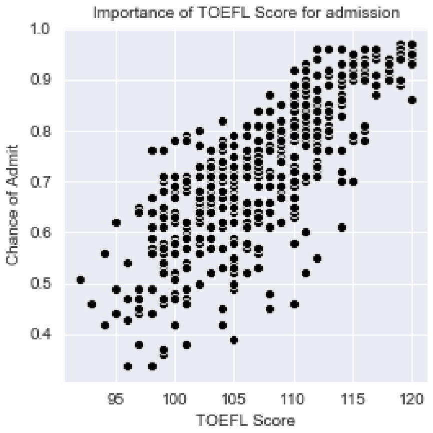


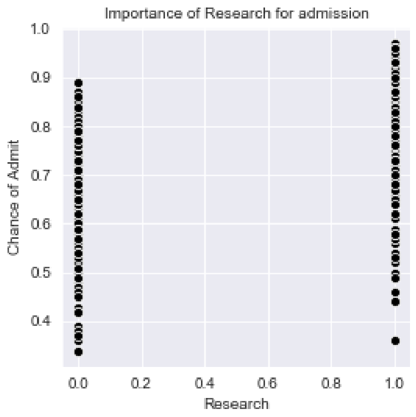
For GRE Score most of the instances seem to be between 310 and 330. For TOEFL Score most of the instances are between 105-115. CGPA has a greater number of instances towards the higher end and so if for Chance of Admit. LOR and SOP have more intances around 3 and 4. Around 270 people have done research while around 210-220 haven’t. The distribution plots give us a clear idea as of how our data has been organized and how it has been distributed. To build a model we must understand the correlation between different variables and Chance of Admit. To find this we have used Seaborn’s scatterplot.

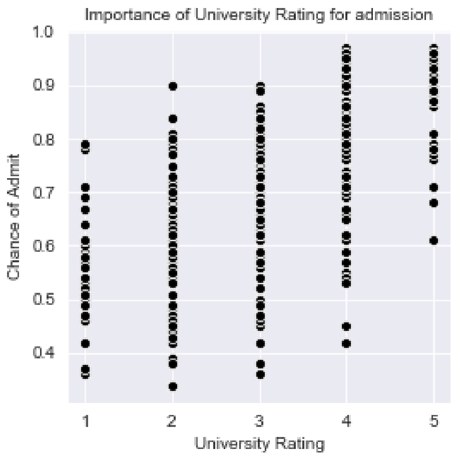












GRE Score and TOEFL Score have a high impact on Chance of Admit as shown in the scatter plot. Whereas SOP, LOR, Research and University Rating don't. CGPA shows the highest correlation with Chance of Admit making it the most important factor for admission into a university for master's degree. Students with higher scores in LOR, SOP and with some research experience

5 Mathematics behind logistic regression

Lets first understand how a Logistic Regression model actually works by looking at the mathematics behind it. The following equation is that of a multiple linear regression model:

$$y = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n$$

Here y is the variable the model is predicting and the other terms are the dependent variables with their coefficients (β). The variable is denoted by x and the coefficient by β. Coefficients are the rate by which the predicted value of y is affected when there is a unit change in the variable corresponding to it.

Now coming to Logistic Regression, lets use the equation of Linear Regression to derive the formula and then relate it to our model. Logistic Regression model uses the logistic function which is:

$$p(X) = (e^{\beta_0+\beta_1 X_1+\cdots+\beta_n X_n})/(1 + e^{\beta_0+\beta_1 X_1+\cdots+\beta_n X_n})$$

Here X is the event our model will be predicting. X in our case will be Chance of Admit. If it is 1 then the person has over 75% chance of getting admitted and if 0, the chances are less than 0. For logistic regression we predict the probability of occurrence of an event which is denoted by p(x). β_0 will be the intercept provided by our model, $\beta_1 - \beta_7$ will be the coefficients provided by our model. $X_{1-} X_7$ are the values of the X test instance which the model will be using for prediction. To help get values close to 0 and 1 for better prediction we manipulate the above equation to:

$$p(X)/(1 - p(X)) = e^{\beta_0+\beta_1 X_1+\cdots+\beta_n X_n}$$

The left side of this equation is called the odds and can take values ranging from 0 to ∞. By taking logarithm at both sides, we get:

$$\log(p(X)/(1 - p(X))) = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n$$

So, we get this as our final equation. The left side is called the logit or log-odds. The relationship between p(X) and $X_{1-} X_7$ is not a straight-line and the rate of change in p(X) per unit change in $X_{1-} X_7$ depends on the current value of $X_{1-} X_7$.

6 Model Prepartation

6.1 Model 1

For model 1 we haven't done any scaling or any hyperparameter tuning. We manipulated the Chance of Admit data by classifying it to 0 and 1 using the Pandas 'cut' attribute. For chance of admit from 0.0 to 0.74, we have set it to 0. For chance of admit from 0.75 to 1.0, we have set it to 1. That is if the chance of admit is 75% or greater, the chance of admit will be 1 and if lower then 0. We then assign x and y values for train test split. To x we assign all the values of the columns except that of Chance of Admit. To y we assign the values of Chance of Admit. Here are how our first 5 instances of x look:

	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	337	118	4	4.5	4.5	9.65	1
1	324	107	4	4.0	4.5	8.87	1
2	316	104	3	3.0	3.5	8.00	1
3	322	110	3	3.5	2.5	8.67	1
4	314	103	2	2.0	3.0	8.21	0

Here are how our first 5 instances of y look:

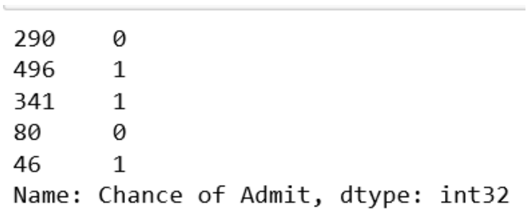
```
0    1
1    1
2    0
3    1
4    0
Name: Chance of Admit, dtype: int32
```

Now to perform train test split we import train_test_split from sklearn's model_selection package. We input x and y to use them for split and assign test size to be 0.25, which means that our test size will be 25% of the randomly chosen data. The random state has been set to 1. Random state is used to initialize internal random number generator. By setting the random state number, one will get the same train test split data over multiple runs which helps provide better analysis of data. Let's have a look at our split data now.

First 5 of x train:

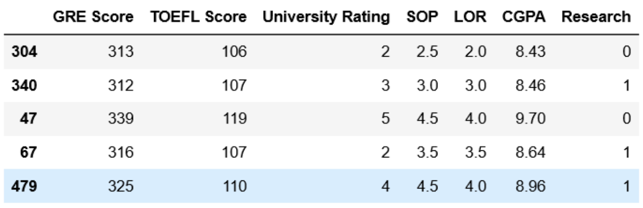
	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
290	307	105	2	2.5	3.0	7.65	0
496	337	117	5	5.0	5.0	9.87	1
341	326	110	3	3.5	3.5	8.76	1
80	312	105	3	2.0	3.0	8.02	1
46	329	114	5	4.0	5.0	9.30	1

First 5 of y train:



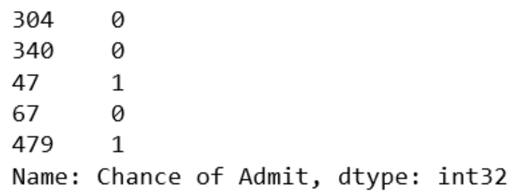
```
290    0
496    1
341    1
80     0
46     1
Name: Chance of Admit, dtype: int32
```

First 5 of x test:



	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
304	313	106	2	2.5	2.0	8.43	0
340	312	107	3	3.0	3.0	8.46	1
47	339	119	5	4.5	4.0	9.70	0
67	316	107	2	3.5	3.5	8.64	1
479	325	110	4	4.5	4.0	8.96	1

First 5 of y test:



```
304    0
340    0
47     1
67     0
479    1
Name: Chance of Admit, dtype: int32
```

Now that our data has been split, we will fit the data into the model to train it using 'fit' method. Once our data is fit, we will see how our model works on the test data by making predictions. After making predictions we will make a confusion matrix using sklearn's metrics package which helps make a confusion matrix.

```
[[66 8]
 [9 42]]
```

The above matrix is our model's confusion matrix. So off our test set, our model has identified 66 instances as True Negatives (TN), 8 as False Positives (FP), 9 as False Negatives(FN) and lastly 42(TP) as True Positives. All these values help us calculate precision score, f1-score, recall score and accuracy. Precision, F1-score, and Recall are different ways to measure how a model has performed.

The formula for precision is:
Precision = TP/ (TP + FP)

The formula for recall is:
Recall = TP/ (TP + FN)

The formula for f1 score is:
*F1 = 2 * [(Precision * Recall)/ (Precision + Recall)]*

The formula for accuracy is:
Accuracy = (TP + TN)/ (TP+FP+FN+TN)

So, if we now calculate all the measures using sklearn's built in function, then we get accuracy score as: 0.864 on test set and a score of 0.8586 on training set. Which is reasonable taking into account that the data isn't scaled, or the parameters haven't been tuned. We achieved a precision score of 0.84, f1 score of 0.8316 and recall as 0.8235. The model's overall performance is decent, but we can improve it. In our next models we will be using the same data and apply some scaling technique to improve its performance.

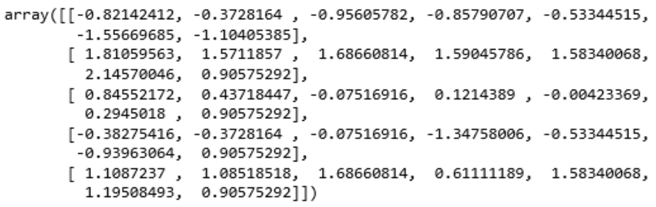
6.2 Model 2 and 3 with Scalers

To understand how scaling improves a model, we must understand why it is used. In our x train and test array, our data varies over a very large scale. For model 2 and 3 we will be taking the same training and testing data and scale the x features. We will be comparing two scaling techniques, StandardScaler and MinMaxScaler. Standard Scaler brings the values in the x train and x test around 0 making it easier for the model to perform. Now to look at MinMaxScaler, it will bring the x test and x train values in between 0 and 1.

The formula followed by standard scaler is:
$(X_i - mean(X))/ stddev(X)$

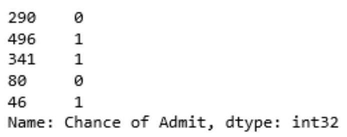
The formula followed by minmaxscaler is:
$(X_i - min(X))/ (max(X) - min(X))$

For model 2 we will be using StandardScaler and for model 3 we will be using MinMaxScaler. For our model 2 we apply the scaler to x train and x test and then make predictions after fitting the model the training data. After fitting the x train to scaler, we get the new x train which looks like (showing only first 5):



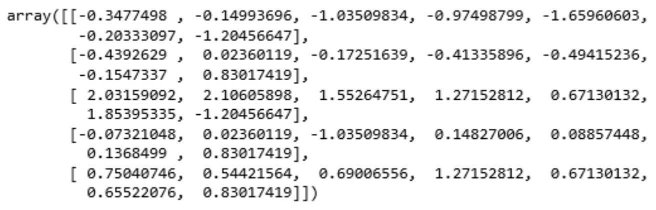
```
array([[-0.82142412, -0.3728164 , -0.95605782, -0.85790707, -0.53344515,
        -1.55669685, -1.10405385],
       [ 1.81059563,  1.5711857 ,  1.68660814,  1.59045786,  1.58340068,
         2.14570046,  0.90575292],
       [ 0.84552172,  0.43718447, -0.07516916,  0.1214389 , -0.00423369,
         0.2945018 ,  0.90575292],
       [-0.38275416, -0.3728164 , -0.07516916, -1.34758006, -0.53344515,
        -0.93963064,  0.90575292],
       [ 1.1087237 ,  1.08518518,  1.68660814,  0.61111189,  1.58340068,
         1.19508493,  0.90575292]])
```

The first 5 of our y train are:



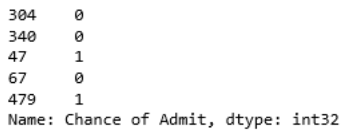
```
290    0
496    1
341    1
80     0
46     1
Name: Chance of Admit, dtype: int32
```

The first 5 of our x test data:



```
array([[-0.3477498 , -0.14993696, -1.03509834, -0.97498799, -1.65960603,
        -0.20333097, -1.20456647],
       [-0.4392629 ,  0.02360119, -0.17251639, -0.41335896, -0.49415236,
        -0.1547337 ,  0.83017419],
       [ 2.03159092,  2.10605898,  1.55264751,  1.27152812,  0.67130132,
         1.85395335, -1.20456647],
       [-0.07321048,  0.02360119, -1.03509834,  0.14827006,  0.08857448,
         0.1368499 ,  0.83017419],
       [ 0.75040746,  0.54421564,  0.69006556,  1.27152812,  0.67130132,
         0.65522076,  0.83017419]])
```

The first 5 of our y test data:



```
304    0
340    0
47     1
67     0
479    1
Name: Chance of Admit, dtype: int32
```

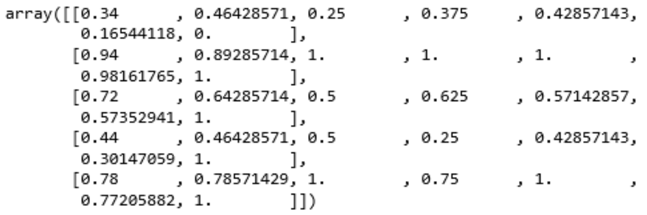
Fitting our new training data into the model we achieve an accuracy of 0.904 on test set and a score of 89.6 on training data. The confusion matrix for the model is like:

```
[[67  7]
 [5  46]]
```

So now our model has predicted 67 True Negatives (TN), 7 False Positives (FP), 5 False Negatives (FN) and 46 True Positives (TP). We achieved a total of 0.8679 precision score, 0.8846 f1 score and 0.9019 recall. So clearly our model's overall performance has improved. It happened since we centralized all the values in x train and x test around 0.

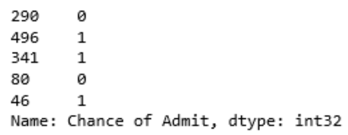
Now to look at model 3, putting the x train and x test data we get the following data.

The first 5 for x train:



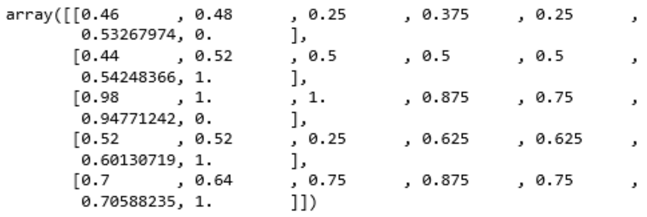
```
array([[0.34      , 0.46428571, 0.25      , 0.375     , 0.42857143,
        0.16544118, 0.        ],
       [0.94      , 0.89285714, 1.        , 1.        , 1.        ,
        0.98161765, 1.        ],
       [0.72      , 0.64285714, 0.5       , 0.625     , 0.57142857,
        0.57352941, 1.        ],
       [0.44      , 0.46428571, 0.5       , 0.25      , 0.42857143,
        0.30147059, 1.        ],
       [0.78      , 0.78571429, 1.        , 0.75      , 1.        ,
        0.77205882, 1.        ]])
```

The first 5 for y train:



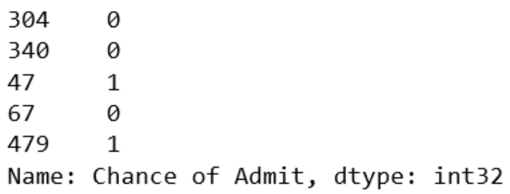
```
290    0
496    1
341    1
80     0
46     1
Name: Chance of Admit, dtype: int32
```

The first 5 for x test:



```
array([[0.46      , 0.48      , 0.25      , 0.375     , 0.25      ,
        0.53267974, 0.        ],
       [0.44      , 0.52      , 0.5       , 0.5       , 0.5       ,
        0.54248366, 1.        ],
       [0.98      , 1.        , 1.        , 0.875     , 0.75      ,
        0.94771242, 0.        ],
       [0.52      , 0.52      , 0.25      , 0.625     , 0.625     ,
        0.60130719, 1.        ],
       [0.7       , 0.64      , 0.75      , 0.875     , 0.75      ,
        0.70588235, 1.        ]])
```

The first 5 for y test:



```
304    0
340    0
47     1
67     0
479    1
Name: Chance of Admit, dtype: int32
```

Fitting our updated data into the model we achieve an accuracy like that of the standard scaler which is 0.904 on test set but a little lesser score than that of standard scaler on training set and that is 0.888. The confusion matrix for this model is like:

```
[[66  8]
 [4  47]]
```

So, our model predicted 66 True Negatives (TN), 8 False Positives (FP), 4 False Positives (FN) and 47 True Positives (TP). We achieved 0.8545 precision, 0.8867 f1 score and a recall of 0.9215. This is because our data has been brought between 0 and 1 or within a limited range.

So overall considering all the parameters of measuring the performance of a model, the MinMaxScaler provides us with a better model.

6.3 Model 4 and 5 with GridSearchCV

Using GridSearchCV we will be tuning hyperparameters like penalty and C. Penalty can be either L1 or L2. If it is L1 then the model is called Lasso Regression and if L2 then it is Ridge Regression. Lasso Regression model will add the absolute value of magnitude of coefficients as penalty term whereas Ridge Regression will add the squared magnitude of coefficients. C in scikit is the inverse of lambda or inverse of regularization strength. We will be splitting our data in 10 parts and fit it in our grid model. For GridSearchCV we will have to pass a param_grid which is the various parameters we ask it to test the models on and get the optimal parameters. Our param grid looks is:

{'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000], 'penalty': ['l1','l2']}

MODEL 4: Given these parameters and setting cv as 10, we will first test on our model with standard scaler which is model 2. The x train, y train, x test and y test are the same as that of model 2. After fitting the training set into the model to train and creating predictions we get a confusion matrix which is:

```
[[67  7]
 [4  47]]
```

The best parameters given by for grid model are:

```
{'C': 0.1, 'penalty': 'l1'}
```

So, our model has identified 67 True Negative (TN) values, 7 False Positives (FP), 4 False Negatives (FN) and 47 True Positives (TP). We achieve an accuracy of 0.912 which shows there is sufficient increase in accuracy using GridSearchCV. The score on training set is 0.904, precision score of 0.8703, f1 of 0.8952 and recall of 0.9215. So, this shows that after the data has been scaled by standard scaler, it gives us better results but if we use GridSearchCV after scaling our model is likely to show more efficiency.

MODEL 5: Given the same parameters and the training and testing data from model 3 we will create a GridSearchCV model. After fitting the training set data to train our model and making predictions we get a confusion matrix like:

```
[[67  7]
 [1  50]]
```

The best parameters given by for model are:

```
{'C': 10, 'penalty': 'l1'}
```

So our model has identified 67 True Negative (TN) values, 7 False Positive (FP) values, 1 False Negative (FN) and 50 True Positive (TP) values. We achieve an accuracy of 0.936 and a score of 0.901 on training set. Precision score achieved was 0.8771, recall was 0.9803 and f1 score was 0.9259. So, we can see how much of a difference hyperparameter tuning can make. Scaling the data and then tuning the parameters helps us provide a better and efficient model.

7 How to make Predictions

To see how predictions are made we will have to use 2-3 inbuilt function and call their values. We will use the intercept method of our model to get the intercept or β_0 and the coefficients method to get the values of $\beta_1 - \beta_7$. For calculation of our predictions, we will take the first set of values of x test and predict y test manually to show how prediction are made.

The first set of x test values are:

```
array([[0.46      , 0.48      , 0.25      , 0.375     , 0.25      ,
        0.53267974, 0.        ]])
```

So X_1(GRE Score) is 0.46, X_2(TOEFL Score) is 0.48, X_3(University Rating) is 0.25, X_4(SOP) is 0.375, X_5(LOR) is 0.25, X_6(CGPA) is 0.53267974 and X_7(Research) is 0.

The first value in y test array is:

```
304    0
Name: Chance of Admit, dtype: int32
```

The predicted value by computer is:

```
array([0])
```

Now to see how this value was predicted let's call the intercept using the intercept method of our model.
Upon calling the intercept, the value provided to us was:

```
array([-12.13066541])
```

Now we know that our intercept or β_0 is -12.13066541.
Let's check the coefficients. Upon calling the coef method, the values provided to us were:

```
array([[ 3.47888443,  2.8373769 ,  1.98895515,  1.00991168,  0.40354878,
        11.40485272,  0.41679281]])
```

So β_1(GRE Score) is 3.4788843, β_2(TOEFL Score) is 2.8373769, β_3(University Rating) is 1.98895515, β_4(SOP) is 1.00991168, β_5(LOR) is 0.40354878, β_6(CGPA) is 11.40485272 and β_7(Research) is 0.41679281.

The formula we will be using is that of logistic function:

$$p(X) = (e^{\beta_0+\beta_1 X_1+\cdots+\beta_n X_n})/(1 + e^{\beta_0+\beta_1 X_1+\cdots+\beta_n X_n})$$

Now that we have all the values needed let's calculate the prediction. Calculating only a sub part first which is:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \beta_4 X_4 + \beta_5 X_5 + \beta_6 X_6 + \beta_7 X_7$$

we get: -2.11640816072107. Now we calculate $e^{-2.11640816072107}$ and get: 0.12045719515702515180394445887843. Putting the values in the formula we get 0.10750718160201006.
Since the value is less than 0.5, we get 0 as the predicted value. So, by seeing this we know how our model used logistic function to predict the values.

7 Conclusion

Everyone knows how to make a machine learning model but to apply it in real life scenarios one must understand how their model works and how it can be made more efficient. To improve its performance, one must tune the hyperparameters like in the case of our model we tuned C and Penalty hyperparaneters by GridSearchCV. With a normal Logistic Regression model, we achieved 0.864 and upon improvising via scaling and hyperparameter tuning we achieved 0.936. So clearly our model has shown better performance.

Acknowledgment

We would like to thank Manvinder Singh Oberoi for his continuous support and guidance and Mohan S. Acharya for his research and dataset.

References

[1] https://www.kaggle.com/mohansacharya/graduate-admissions
[2] 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
[3] https://github.com/SSaishruthi/LogisticRegression_Vectorized_Implementation/blob/master/Logistic_Regression.ipynb
[4] Arnav Oberoi, Github, Kaggle and Other Projects, https://github.com/arnavobero1/Kaggle-and-Other-Projects
[5] Koo Ping Shung, Towards Data Science, Accuracy, Precision, Recall or F1?

[6] Anuja Nagpal, Towards Data Science, L1 and L2 Regularization
[7] Anuja Nagpal, Towards Data Science, Over-fitting, and Regularization.
[8] Wikipedia, Null Hypothesis, and calculation of P value
[9] Scikit-learn.org, GridSearchCV
[10] Tara Boyle, Towards Data Science, Hyperparameter Tuning
[11] Kevin Markham, Data School, Machine Learning and Application
[12] Kirill Ermenko, Hadelin de Ponteves, Udemy, SuperDataScience, Machine learning in R and Python
[13] Jose Portilla, Udemy, Pierian Data, Data Science and Machine learning using Python
[14] Arnav Oberoi, Joshua Sabherwal, Kaggle, Iris ML Kernel

First Author: Arnav Oberoi Department of Computer Science and Engineering, The NorthCap University, Near Rotary School, Gurugram, Haryana. E-mail: arnav@oberoi.co.in

Second Author: Om Sehgal Department of Computer Science and Engineering, The NorthCap University, Near Rotary School, Gurugram, Haryana. E-mail: omsehgal9211@gmail.com

Third Author: Chirag Malik Department of Computer Science and Engineering, The NorthCap University, Near Rotary School, Gurugram, Haryana. E-mail: chiragmalik1301@gmail.com