# Machine Learning Engineer Nanodegree

## Capstone Project

June Yim

May 15th, 2019

# I. Definition

## Project Overview

When graduate schools decide to get new entrants, they consider many records of the applicants including undergraduate GPA, GRE, TOEFL, letters of recommendation and so on. From the Data Science and Machine Learning point of view, by analyzing the admission data and training that, we can make a prediction model to help applicants to concentrate on more important factors which lead them to get the acceptance.

## Problem Statement

The problem is : “How much I can have confidence that I can get the acceptance from graduate school with my record?” The record includes GPA, GRE score, TOEFL score, and so on.
The confidence is expressed as a number in range [0, 1]. That is, we can

think the confidence as the probability of getting the acceptance.
The solution to the problem is the confidence level that the applicants can expect for the graduate admission. The confidence level is a number between 0 and 1 (inclusive) so the applicants can think of this as the probability that he or she can get the admission with his or her records(GRE score, TOEFL score,...). I will attempt to solve this problem with various regression models including linear regression, decistion tree, random forest and so on. Furthurmore, I will also try to apply classification algorithms like Gaussian Naive Bayes.

## Metrics

There are many metrics to measure performance of regression problem including RMSE, R squared and so on:

- RMSE(Root Mean Square Error): This measures the standard deviation of the errors the system makes in its predictions. Smaller is better.
- R squared(Coefficient of Determination): This is the proportion of the variance in the dependent variable that is predictable from the independent variables. Larger is better.

I will use RMSE for performance measurement.

# II. Analysis

## Data Exploration

I used Graduate Admission dataset from kaggle.
(https://www.kaggle.com/mohansacharya/graduate-admissions)
This dataset contains 9 parameters:

- Serial No. (1 to 500)
- GRE Scores (290 to 340)
- TOEFL Scores (92 to 120)
- University Rating (1 to 5)
- Statement of Purpose (1 to 5)
- Letter of Recommendation Strength (1 to 5)
- Undergraduate CGPA (6.8 to 9.92)
- Research Experience (0 or 1)
- Chance of Admit (0.34 to 0.97)

I'll set 'Chance of Admit' parameter as the label and the remaining parameters as inputs to make a prediction model. Serial No. will be dropped in making the model because it is simply used for indexing purpose.
As I showed in the jupyter notebook ("capstone.ipynb") this set has 500 entries, all variables are numeric (there is no categorical variable) and missing values or abnormalities about data were not found.

# Exploratory Visualization

At first, it is worth seeing data summary statistics as to get overall picture of data:


We can see every variable has 500 entries so there is no missing value

in the dataset. And from the mean, standard deviation and the other statistics we can expect there are not abnomalities we should take care of before building up the prediction model.

Next, by using panda’s corr() function we can determine the most important variables which are highly related to the target variable (‘Chance of Admit’) are CGPA, GRE Score, and TOEFL Score. (See the source code in “capstone.ipynb”)
We can see the correlations among them by plotting the correlation matrix:

# Algorithms and Techniques

For the regression problem like this, there are many algorithms which show good performance, Linear Regression, Decision Tree, Random Forest and so on. Though choosing anyone would be good I chose Linear Regression because it is simple, easy to implement and understand. For the other algorithms I used them as benchmarks.
When I used Linear Regression algorithm, I used “Chance of Admit” as a dependent(target) variable, and the others as independent(input) variables. I implemented all these in the jupyter notebook. (“capstone.ipynb”)

# Benchmark

I benchmarked several algrithms not only in regression category but

also in classification category:

- Linear Regression
- Decision Tree
- Random Forest
- SVM(Support Vector Machine)
- Gaussian Naive Bayes
- Decision Tree Classifier
- Random Forest Classifier
- Logistic Regression

Each benchmark was performed on train data set at first, and then on test set.
Thinking of overfitting due to small data size, I used Scikit Learn's k-fold cross-validation for refinement. It randomly splits the training set into k distinct subsets called folds, then it trains and evaluates the model k times, picking a different fold for evaluation every time and training on the other k-1 folds. (I can decide and give the number k to the cross-validation function as a parameter)
The results are in the following Implementation section.

# III. Methodology

## Data Preprocessing

As I mentioned above, any abnormalities in data set did not found. Furthermore, as there is no categorical variable, we can focus our consideration on preprocessing for numerical variables.

When we look at the data statistics (see Data Exploration section above) the value scale ranges are different among variables, we need to adjust scale to get more accurate results.
Scikit learn provides some scaling methods including Standard scaler, MinMax Scaler, and so on.
I used MinMaxScaler which scales every numerical data into (0, 1).

# Implementation

1. Define Input & Output : As a first step, I defined output(target) variable and input variables for a prediction model.
   Output is “Chance of Admit” and the remaining variables are inputs to be used for the prediction. Among the variables, “Serial No.” was dropped.

2. Split Train & Test set : Next, I splitted the data set into train and test set. Train set will be used for model implementation and test set will be used for measuring the implemented model’s prediction performance for unseen data.
   For splitting, I used Scikit learn’s train_test_split function with proper argument settings ( test set size to be 20% of data set and random state to be 42 which will be used as a seed by random number generator)

3. Feature Scaling : To adjust input data scale, I used MinMaxScaler of Scikit Learn.
   This scaler converts every input data (all numeric) into (0, 1)

scale. To do this, I applied fit_transform method to train data and transform method to test data.

4. Now, It's time to fit our train data. I chose Linear Regression algorithm and fit the train data which I splitted in step 2.
   Scikit learn provides many machine learning algorithms. To use linear regression algorithm, I imported LinearRegression from sklearn.linear_model. There are not much thing to do after importing. Just calling 'fit' with our train input and label variable is enough to fit our train data. Lastly, I performed prediction with train data input using predict method of LinearRegression.

5. Measure Performance : To measure our prediction model's performance, some metrics are needed. I chose RMSE for measuring my regression model's performance.
   To use that, I imported mean_squared_error from sckit learn and gave train data label and the prediction result from step 4 as function arguments. And then applied numpy's square root function to get RMSE.

6. Compare with test data set : After prediction and measurement with train data, to see the performance of my prediction model on unseen data, I performed prediction and measurement (wth RMSE) again on test data set. If the RMSE of train set is far lower than that of test set, this means overfitting happens. My result was train RMSE: 0.0593 and test RMSE: 0.0608, nearly same two numbers, which means overfitting did not happen in my linear regression model.

7. Benchmark : To benchmark, I tried some other regression algorithms including Decision Tree, Random Forest. Also, for wider benchmarking I tried some classification algorithms including SVM(Support Vector Machine), Gaussian Naive Bayes, Decision Tree Classifier, Random Forest Classifier and Logistic Regression. To use these classification algorithms I needed to define binary classification label. I divided the target variable - Chance of Admit - into two classes, 1 and 0. When the Chance of Admit is greater than 80%, the data instance is labelled with '1' or else, it is labelled with '0'. For metric I used Sckit learn's score method(the mean accuracy on the given test data and labels) as well as f1 Score(weighted average of the precision and recall)

8. Benchmark Results :

- Regression Algorithms Benchmark Results

| Algorithm | RMSE(train) | RMSE(test) |
|---|---|---|
| Linear Regression | 0.0593 | 0.0608 |
| Decision Tree | 1.6653e-17 | 0.0929 |
| Random Forest | 0.0250 | 0.0651 |

- Classification Algorithms Benchmark Results

| Algorithm | Score | F1 Score |
|---|---|---|
| SVM | 0.97 | 0.94 |

| | | |
|---|---|---|
| Gaussian Naive Bayes | 0.93 | 0.88 |
| Decision Tree Classifier | 0.95 | 0.91 |
| Random Forest Classifier | 0.96 | 0.92 |
| Logistic Regression | 0.96 | 0.92 |

# Refinement

I divided the original data set into two groups - train, test set - to train(fit)the model with train set and test the model for unseen data. But overfit problem occurred for some models especially it was serious on decision tree algorithm (RMSE for train set was nearly zero!)
There can be many reasons that this overfitting problem occurs but the small size of data set is major reason, I think.
To address this, I tried cross validation technique which divides the data set into smaller groups and uses one by one as the validation data set. Scikit learn privides cross_val_score function so I imported it and made 10 subgroups (by setting ‘cv’ parameter to be 10). I used ‘negative mean squared error’ as a ‘scoring’ parameter for cross_val_score function. Scikit learn cross-validation features expect a utility function (greater is beter) rather than a cost function (lower is better), so the scoring function is actually the opposite of the MSE(i.e., a negative value), which is why I used ‘negatieve mean squared error’ and computed ‘-scores’ before calculating the square root in the code file(‘capstone.ipynb’).
As cross validation made 10 RMSE values, so I calculated mean and standard deviation of them and I repeated this process for three regression algorithms: Linear Regression, Decision Tree, Random

Forest.

The cross validation results are as follows: (RMSE Mean for test data set)

- [Linear Regression] 0.059
- [Decision Tree] 0.085
- [Random Forest] 0.064
- Comparing to the benchmark results in Implementation section, we can see that they are considerably improved.

# IV. Results

## Model Evaluation and Validation

To test my final model, I tried some test data and compared the predicted outputs with labels.(see source code in ‘capstone.ipynb’)

The part of results are as follows:

| Index | Prediction | Real |
|---|---|---|
| 0 | 0.91 | 0.93 |
| 10 | 0.72 | 0.45 |
| 20 | 0.67 | 0.67 |
| 30 | 0.56 | 0.59 |
| 40 | 0.97 | 0.96 |
| 50 | 0.89 | 0.93 |
| 60 | 0.63 | 0.68 |

| | | |
|---|---|---|
| 70 | 0.62 | 0.64 |
| 80 | 0.77 | 0.80 |
| 90 | 0.74 | 0.73 |

My final model seems to work well and is making reasonable outputs for test(unseen) data.

## Justification

To compare my final model with benchmarks, I repeated the above process: I entered the same input data into the benchmark models and compared the results. (see the source code in ‘capstone.ipynb’)
I attached the resulting plot here:

I plotted 3 predictions (LR,DT,RF) and real values(labels) for 10 test data.(Index 0, 10, 20, 30, 40, 50, 60, 70, 80, 90)
Except in index 10, Linear Regression model’s prediction is closer to the real value or at least nearly same as the other model’s predictions.

# V. Conclusion

## Free-Form Visualization

To see feature importance, plotting heatmap is another good choice. Heatmap shows us the correlations among features, so we can determine the most important features by seeing the correlations between the target(“Chance of Admit”) and the other features(input

variables)

From the heatmap, we can see the top three important variables which are highly related to the target variable (Chance of Admit) are CGPA, GRE Score and TOEFL Score, which coincides with what we saw above.

# Reflection

In this project I made a prediction model to predict the ‘chance of admit’ for a graduate school with some input features including CGPA, GRE score, TOEFL score and so on.
To make the model I benchmarked many algorithms in regression as well as classification models. For a metric I used RMSE (Root Mean Square Error) and used it to meaure the performance of algorithms. After benchmarking I got my final model (Linear Regression model) and tested it with some input data.
One interesting thing occurred in fitting the train data with Decision Tree algorithm. The metric - RMSE - was nearly zero. At first, I thought there must be some calculation error or parameter setting error, but I came to know it was significant overfitting. To address this overfitting problem, I used Cross Validation which divides the original data set into small groups and use them as validation data. After benchmarks and refinement I came to get my final model.

# Improvement

The target variable I used in this project was the interviewees’ reply

(how much possibility(0% ~ 100% range) they expect they got the admission acceptance from the graduate school with there track records (i.e. the features including CGPA, GRE, TOEFL, and so on))
In other words, the label in this supervised learning problem was not the real results but the interviewees' expectations.
If I can collect the sufficiently many real acceptance data (Yes or No), and use them as labels I can make more useful prediction model.

---