Image Source: College Admissions Requirements

Predict College Admissions

CAPSTONE PROJECT REPORT

Piyush Agarwal | Udacity Machine Learning Nanodegree | November 25, 2018

Table of Contents

Definition

Project Overview

We all have been in the situation where we are searching for colleges to apply for. These days, there are a lot of colleges around and each of them has a different acceptance criterion for admissions. While filling out our application we want to make sure that we choose a college with a higher probability of accepting our application - one, because the application costs are high and second, because we want to apply to a college which admits in the range of our test scores.

Supervised Machine Learning has proven its capabilities repeatedly. It is being applied in many domains these days to find answers to complex questions based on the historical data. Many real-world applications, which we use on a day-to-day basis, are using supervised learning in the background - think catching spam emails, predicting stock price movements, financial fraud detection, etc.

In this project, I have used Supervised Machine Learning to predict the number of admissions in a college based on the number of applicants providing test scores (SAT/ACT) and the scores itself.

Problem Statement

Most of the information these days can easily be found by "Google-ing" but, think about a scenario in which you are applying to a college whose admissions rate is unknown or not available online yet (maybe because it's a new college). Main questions to ask are:

- *Can we build a model which can predict the admissions to a new college based on other similar colleges whose data is available?*
- *How much do the test scores affect the admissions?*

This can be considered as a Regression Problem which can be solved using Supervised Machine Learning techniques - where the input variables (in this case number of students submitting their test scores and their test scores itself) will be used to predict the continuous dependent variable (admissions number).

After data gathering and cleaning (described in detail in the Data Preprocessing section), I am planning on using few proven regression algorithms – Linear, Random Forest, Gradient Boosting and Neural Networks to work towards a viable solution for this problem. The output of the Linear Regression model will be used as a baseline/benchmark for the other models. I'm then planning on comparing the statistical measure of *coefficient of determination* (r^2), which is widely used as a metrics for regression models, and pick the model which gives the best value.

Evaluation Metrics

Evaluating the performance of a model is as necessary as building the model itself. For evaluating the performance of this regression model, I used the statistical measure of coefficient of determination (denoted as R^2 or r^2).

It is defined as the proportion of the variance in the dependent variable as predicted by the independent variables(s). Wikipedia defines coefficient of determination mathematically as:

A data set has n values marked $y_1,...,y_n$ (collectively known as y_i or as a vector $y = [y_1,...,y_n]^T$), each associated with a predicted (or modeled) value $f_1,...,f_n$ (known as f_i, or sometimes $\hat{y}_i$, as a vector f).

Define the residuals as $e_i = y_i - f_i$ (forming a vector e).

If $\bar{y}$ is the mean of the observed data:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$$

then the variability of the data set can be measured using three sums of squares formulas:

- The total sum of squares (proportional to the variance of the data):

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2,$$

- The regression sum of squares, also called the explained sum of squares:

$$SS_{\text{reg}} = \sum_i (f_i - \bar{y})^2,$$

- The sum of squares of residuals, also called the residual sum of squares:

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2$$

The most general definition of the coefficient of determination is

$$R^2 \equiv 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}}$$

Put in simple words -

coefficient of determination shows the effect of variation in input features on the variation of the predicted feature. The best r^2 value a model can have is 1.0 - higher this value, the better our model. A negative r^2 value shows that the model is performing arbitrarily worse and needs some major fixing.

Analysis

Data Exploration

I used the admissions data from *Integrated Postsecondary Education Data System (IPEDS)*, which is established as the core postsecondary education data collection program for *National Center for Education Statistics (NCES)* and is a system of surveys designed to collect data from all primary providers of postsecondary education.

The NCES website has the IPEDS data for many years but I ended up using the data for the period of 2010 – 2016. Due to the fast dynamics of the education requirements, taking a somewhat recent set seemed like a good idea to use for model building as the older dataset is not much useful. NCES also provides a dictionary for the variables in the data which is very helpful resource in understanding the data. After going through the whole dictionary and understanding the variables, I ended up with a subset of 16 variables (15 original and 1 to represent the YEAR of the data). These are described as below:

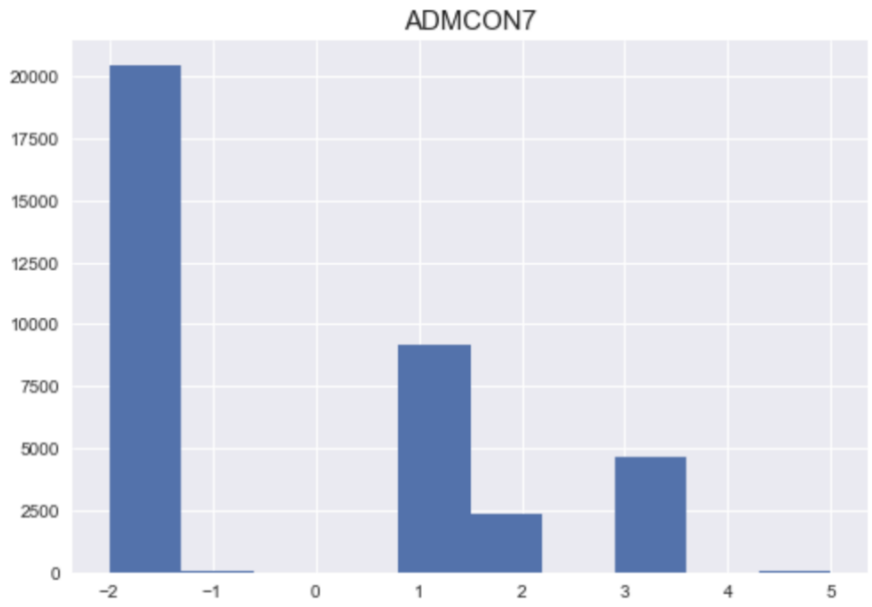
Variable	Description	Values
UNITID	Id assigned to a college	Numeric
YEAR	Year which the data is from	2010 to 2016
ADMCON7	Suggests if a college considers SAT/ACT scores or not	1 – Required 2 – Recommended 3 – Neither required nor recommended 4 – Do not know -1 – Not reported -2 – Not Applicable
APPLCN	Number of applications	Numeric
ADMSSN	Number of admissions	Numeric
ENRLT	Number of enrollments	Numeric
ENRLFT	Number of full-time enrollments	Numeric
ENRLPT	Number of part-time enrollments	Numeric
SATNUM	Number of applicants who reported SAT scores	Numeric
ACTNUM	Number of applicants who reported ACT scores	Numeric
SATVR25	SAT Verbal 25th percentile score	Numeric (1 – 800)
SATVR75	SAT Verbal 75th percentile score	Numeric (1 – 800)
SATMT25	SAT Math 25th percentile score	Numeric (1 – 800)
SATMT75	SAT Math 75th percentile score	Numeric (1 – 800)
ACTCM25	ACT Composite 25th percentile score	Numeric (1 – 36)
ACTCM75	ACT Composite 75th percentile score	Numeric (1 – 36)

There were a few challenges with the data like 1 college having many records for the same year and lot of colleges not reporting or not considering the test scores. I had to tackle with these challenges which is explained in detail in the Data Preprocessing section. Also, there were quite a few of the missing values (as with any other dataset available online), which I had to deal with.

In total, after combining the records from all the years (2010 – 2016), there were *about 36000 records* in total to work with.

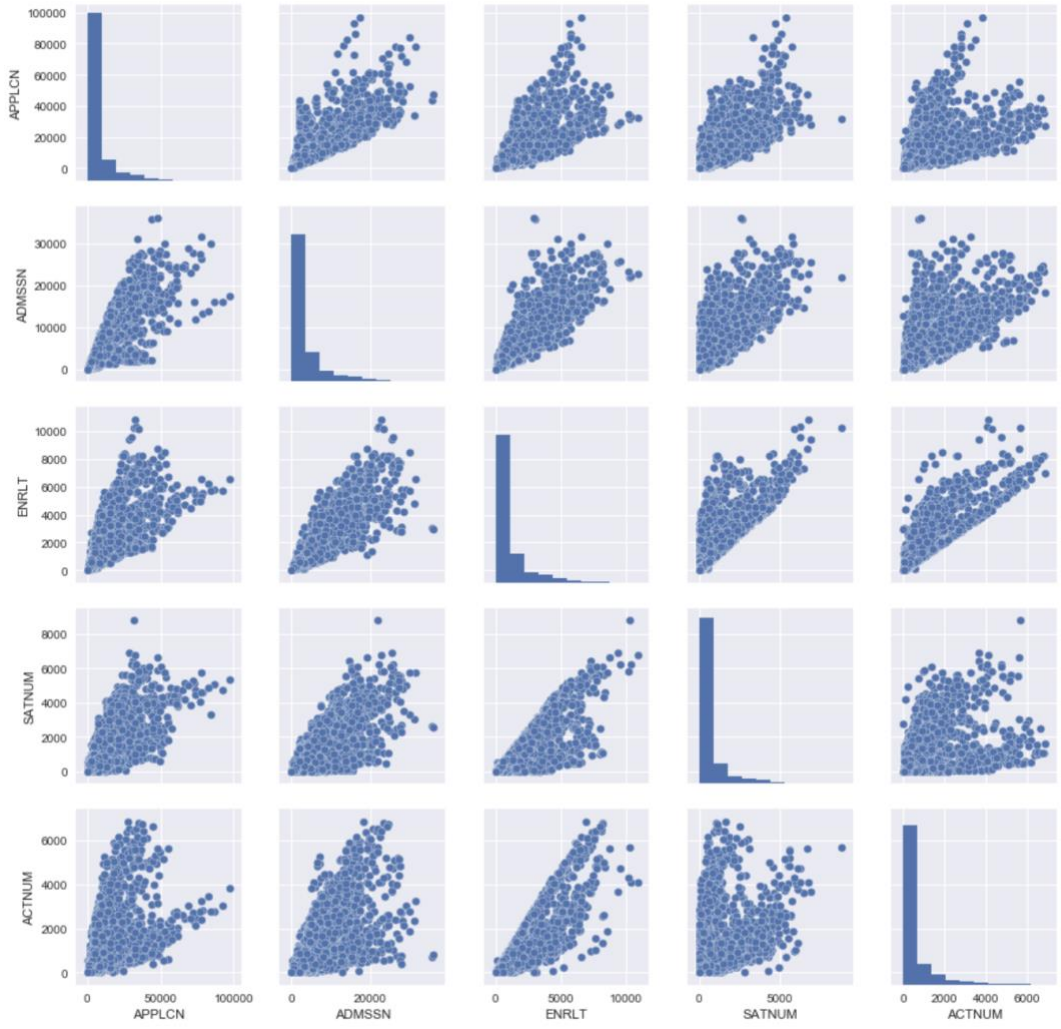
Exploratory Visualization

The first step was to visualize the ADMCON7 variable, which specifies how many colleges consider test scores as a part of their admission decision.



This plot told me that about 20000 records were useless as they were either "Not Reported" or "Not Applicable", which was discouraging as already my dataset was small in size and now it was being reduced to less than half.

I was also curious to see correlation between the number of applicants submitting their scores and the admissions so I thought of making a pair plot of these variables.



This plot shows that the number of applicants reporting scores have a direct correlation with the number of admissions which is definitely a good insight for building this model.

Algorithms and Techniques

Predicting continuous values is considered as a Regression Problem in Machine Learning as there is no discrete set of values to choose from, which is the case for Classification Problem. Regression is a statistical approach to find the relationship between variables. It is used to predict the outcome of an event, based on the relationship between the variables obtained from the data-set. There are various types of regression algorithms available in Python – some simplistic and some very sophisticated. The algorithms I tried working with before finalizing the model are:

Linear Regression: It is a linear approach to modelling the relationship between a scalar response (or dependent variable) and one or more explanatory variables (or independent variables). The case of one explanatory variable is called simple linear regression. For more than one explanatory variable, the process is called multiple linear regression. For a deeper understanding of Linear Regression, refer to its Wikipedia page.

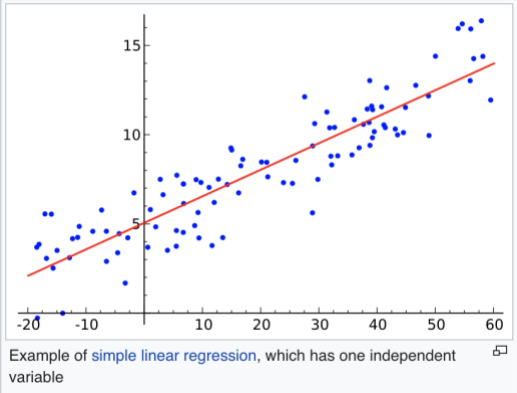


Image Source: Linear Regression Wiki

Random Forest Regression: It is a Decision Tree based approach to solving regression problems. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. Checkout this cool intro to Random Forests on DataScience.com

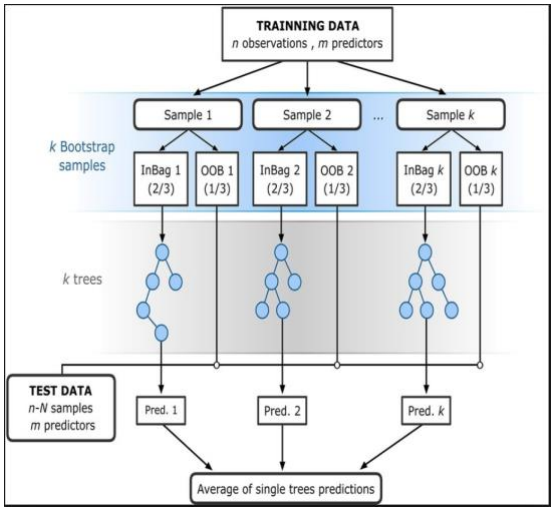


Image Source: Research Gate on RF for regression

Gradient Boosting Regression: It is a machine learning technique for regression problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function. Refer to its Wikipedia page and other resources in the Reference section to learn more on gradient boosting.

Neural Networks: They are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated. Neural Networks are super powerful algorithm and very data intensive as they need a lot of data to make necessary connections and learning. Check out this Wiki on Neural Nets for more reference.

Baseline

I used the Linear Regression model as a baseline. The output of the Linear Regression model gave an *r^2 value of 0.82*, which is a great baseline. My goal was to improve this value, using other more sophisticated algorithms as described above and increase this value to >= ***0.90***.

I was unable to find any previous works done on predicting the admission rates using the IPEDS data, so used the output of my linear regression model as a benchmark. The hardware used was same for all the models trained throughout this project.

Hardware Component	Specifications
Processor	2.3 GHz Intel Core i5
Memory	8 GB
Hard Disk	256 GB

Methodology

Data Preprocessing

Data preprocessing was an important part of this project as I had to deal with few complications with the data and how it was populated.

Using relevant data only

ADMCON7 variable suggests whether a college considers SAT/ACT scores for making an admission decision or not. The allowed values are:

Code	Description
1	Required
2	Recommended
3	Neither Required nor Recommended
4	Do not know
-1	Not reported
-2	Not applicable

And as you can see in the Exploratory Visualization section there are about 20000 records which have either -1 or -2 values and cannot be used, so I had to take them out, leaving me with only 16274 records

Removing records with all Null scores

Test scores are the main features for this model development so if there are all Null values for a record, it didn't made sense to include it. First, the data had "." Instead of NaN so had to replace them with NaN to avoid confusions. Then I dropped the records which have all NaNs for the 8 score features – SATNUM, ACTNUM, SATVR25, SATVR75, SATMT25, SATMT75, ACTCM25 and ACTCM75. Remember, only those records where all these 8 features were NaN were dropped from the dataset. This left me with 9716 records only.

Consolidating college with multiple records for same year

While investigating, I saw a college id which had multiple records for the year 2012. This was surprising because as per my understanding, each college should have only one record per year. I was curious to see if there are any more of these UNITID-YEAR pairs which share the same behavior, so I did some quick analysis which resulted in only one college – the one I found in my earlier investigation.

This was good news, as now I can just consolidate all the records for this college into one record. I followed the simple process of taking the mean of all features for the multiple records of this college data and combining it to one record. Then I replaced those multiple records (1445 records) with just 1 record (one with the means). This left me with final 8272 records for modeling.

Imputing Null with Means

Of these 8272 records, some features had Null values which would not have been good for our model so to fix this I ended up imputing the null values in a feature with the mean of the remaining values of that feature.

Removing records with 0 SAT/ACT Numbers

I also noticed that there were few records which had 0 values for SATNUM and ACTNUM features which suggests that there were no applications to that college which reported their SAT/ACT scores. These records are of no use for model development as they don't have any scores at all, so had to take them out as well which left me with 7431 records.

Feature Engineering

I also ended up creating new features from our initial set of features to help in model development and data consistency. The 2 new features which were created are SATCM25 and SATCM75 which represent the composite scores for SAT calculated by simply taking an average of the scores of 2 test sections – Verbal and Math.

```
SATCM25 = ROUND((SATVR25 + SATMT25)/2)
SATCM75 = ROUND((SATVR75 + SATMT75)/2)
```

I added these new features to the dataset and removed the old features of SAT verbal and math section scores.

Normalize

As both SAT and ACT scores are on a different scale – SAT scores are out of 800, per section and ACT scores are out of 36 per section, an intuitive idea was to normalize these values so that they have an equivalent effect on the output of the model. The normalization was simply done by dividing the composite scores with the max score for each feature –

```
ACTCM25_NORM = ACTCM25 / 36
ACTCM75_NORM = ACTCM75 / 36

SATCM25_NORM = SATCM25 / 800
SATCM75_NORM = SATCM75 / 800
```

That's it, finally all the preprocessing is over and now I could begin with the modeling exercise.

Implementation

- The final dataset after all preprocessing steps was reduced to include the following features for the model training set –
  APPLCN,
  SATNUM,
  ACTNUM,
  YEAR,
  ACTCM25_NORM,
  ACTCM75_NORM,
  SATCM25_NORM,
  SATCM75_NORM

- The ADMSSN variable was kept out as the Labels set

- For developing this model, I chose Python as my language of choice as it has a lot of cool Machine Learning (ML) packages to work with and also being open-source, it has gained a lot of popularity in the ML community

- I also used Keras which is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano (all Neural Network Libraries).

- Other major python packages I used are:

  scikit-learn (for all machine learning algorithms);
  pandas (for data processing);
  numpy (for mathematical calculations and also needed by pandas);
  matplotlib (for plotting);
  seaborn (for cleaner visualizations);

- The whole dataset was split into train and test datasets using scikit-learn's *train_test_split* method. The *test set* was kept as 20% and remaining 80% as the *train set*. Using this technique, we can make sure that we have at least one set of data which the model has never seen and then can use it to validate our model

- First, Linear Regression was used to train a model on the *train set* using the LinearRegression algorithm from the scikit-learn library of python. The *test set* was then scored with this model and the output metrics (r^2 score in this case) used as a baseline for the remaining models

- Subsequently, other models, i.e., RandomForestRegressor (RFR), GradientBoostingRegressor(GBR) and Deep Neural Nets were also trained on the same training set

- Multiple parameter values were tested, using GridSearchCV, to find the best parameters for the RFR and GBR models. The test parameters grid was kept same for both RFR and GBR to do an equal comparison.

```
test_params = {'n_estimators': [100, 500, 1000], 'max_depth': [2, 3, 4, 6], 'random_state': [10]}
```

Which suggests that 3 values (100, 500 and 1000) was tested for the number of estimators and 4 values (2, 3, 4 and 6) was tested for the maximum depth of Decision Trees in both the models. Random state was kept the same as 10.

Below is the best estimator for RFR:

```
Best Random Forest Regressionr estimator from GridSearch:
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=6,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_split=1e-07, min_samples_leaf=1,
           min_samples_split=2, min_weight_fraction_leaf=0.0,
           n_estimators=1000, n_jobs=1, oob_score=False, random_state=10,
           verbose=0, warm_start=False)
```

And this is the best estimator for GBR:

```
Best Gradient Boosting Regression estimator from GridSearch:
GradientBoostingRegressor(alpha=0.9, criterion='friedman_mse', init=None,
             learning_rate=0.1, loss='ls', max_depth=6, max_features=None,
             max_leaf_nodes=None, min_impurity_split=1e-07,
             min_samples_leaf=1, min_samples_split=2,
             min_weight_fraction_leaf=0.0, n_estimators=500,
             presort='auto', random_state=10, subsample=1.0, verbose=0,
             warm_start=False)
```

- For Deep Neural Nets, the model had the following architecture:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 32)                288
_________________________________________________________________
dense_2 (Dense)              (None, 64)                2112
_________________________________________________________________
dense_3 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 65
=================================================================
Total params: 6,625.0
Trainable params: 6,625
Non-trainable params: 0.0
_________________________________________________________________
```

- All the trained models were used to score the *test set* and their r^2 values were compared to each other. The model with the highest value was chosen as per the assumption that higher the r^2 value, the better the model

Refinement

For the benchmarking, I just used the plain LinearRegression model provided by python's scikit-learn library, without tuning any parameters as this was only supposed to serve as the benchmark.

For the RandomForestRegressor(RFR) and GradientBoostingRegressor(GBR), I ended up finding the best parameters from a set of selected parameters using GridSearchCV as described above. *For RFR the best model (model with the best parameters selected) had **1000 trees with a max. depth of 6**. For GBR the best model had **500 trees with a max. depth same as 6**.*

For Neural Nets, no r^2 is provided by default as a metrics by the Keras library, so I had to use a custom metric function for r^2. I found a very helpful article after some research which also provided its implementation.

- The Neural Net uses Tensorflow as the backend
- It's a Sequential Model with 1 input layer, 2 hidden layers and 1 output layer
- The kernel was initialized with 'RandomNormal' so as to use tensors with a normal distribution
- The input and hidden layers all use 'relu' as their activation function whereas the output uses 'linear'
- 'mean_absolute_error' was used as the loss function with 'adam' optimizer.
- The model was trained for 50 epochs and a batch size of 16, with saving checkpoints which reduced losses
- The best checkpoint was picked at the end to compile the final model
- The final model was then used to predict the test set values

---

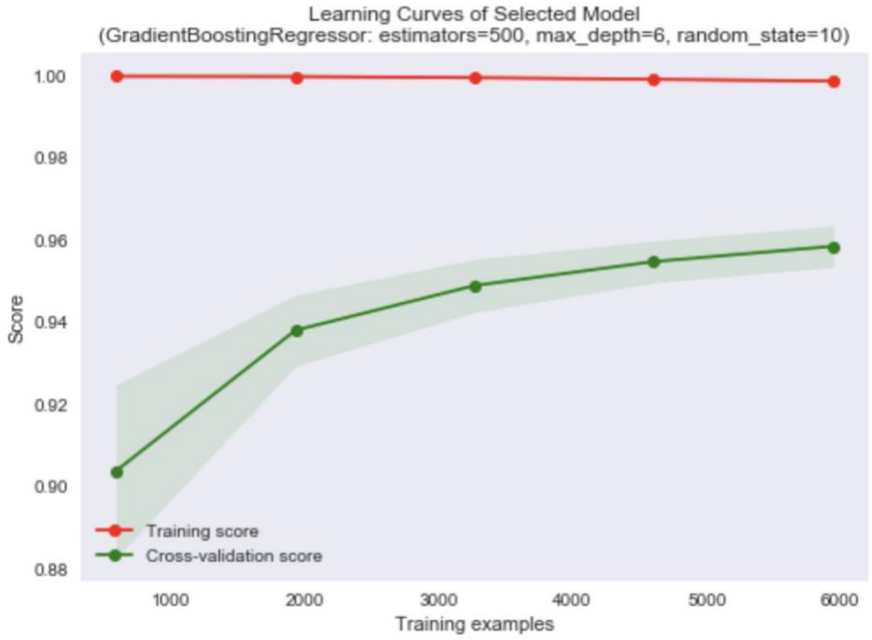
Results

Model Evaluation and Validation

As mentioned above, the r^2 value of the LinearRegression model was selected as the benchmark for the final model. The r^2 values I got from the different model based on training on the same 80% of data and testing on the same 20% of data are summarized in the table below

Model Algorithm	Coefficient of Determination(r^2)
LinearRegression (*baseline*)	0.82
RandomForestRegressor	0.93
GradientBoostingRegressor	0.95
Deep Neural Network	0.84

So basically, all models performed better than the initial baseline model, which is good. With the available training data, Gradient Boosting Regression model gave the best value of r^2 which is way above my goal of 0.90. This maybe because of how Gradient Boosting works in general as it averages the prediction made by all weak-learner trees to predict the final outcome of the model. The train-test learning curve for the final selected Gradient Boosting Regressor model is shown below:



From the learning curve plot, we can see how the validation error is decreasing with the increase in the training sample size. This curve has still not reached a stable value which suggests that if I had more data, this model could have learnt better and made better predictions on the admissions rate.

Gradient Boosting has recently got much attention in the Machine Learning community because of how well it utilizes the weak learners as compared to some other algorithms while also being immune to the effects of overfitting. Check out this great article from a Kaggle Master on Gradient Boosting.

Reflection

This whole process of gathering the data, cleaning it up, putting it in the required format for modeling, finding the best modeling algorithm to work with, tuning the model parameters, etc. gave me a much better understanding of how Machine Learning works and also how I can use it to build better models, given any kind of dataset in the future.

In the beginning of this project, I started with a goal to build a model which can predict admission numbers for a given college based on data from other colleges and their accepted test scores and now that I reflect back on it, I can see how I achieved it - going through so many hours of research and understanding of new and amazing concepts of machine learning and then finalizing on the best algorithm for this model.

I was assuming that Neural Networks will give the best performance but to my surprise Gradient Boosting & Random Forests both performed much better than it. Giving it some more thought I figured that this is probably because of the low amount of data I have for developing this model. The actual benefits of Neural Networks and Deep Learning is visible when there is tons of data for it to work on. With such small amount of data, the Neurons cannot learn enough to make better predictions whereas it's easier for the Decision Trees in the other algorithms to make those connections.

Improvement

- Definitely getting more data would help. This whole project was based on just about 7500 records which is very small. Small enough that a Neural Network was not even able to give its best performance.

- Understanding and tuning of hyper-parameters is also essential. I looked at few parameters of the scikit-learn models only but I could have invested a lot more time in understanding them and seeing how all of them affect the model training.

- The IPEDS data, although clean and presented in a nice way, is still very confusing. It took me a lot of time understanding the dataset and coming up with the list of variables which could be used for model development. Also, lot of data was unusable and cannot be used for model development. As part of improvement, I could look into other sources of getting college admissions data and maybe combine them altogether to give a much bigger dataset to work with.

- Better understanding of the data would also lead to coming up with more useful features, which could significantly help in improving the model.

References

- National Center for Education Statistics
- Integrated Postsecondary Education Data System
- Strengths and Weaknesses of the IPEDS Data
- Coefficient of Determination Wiki
- Scikit-Learn r2_score
- Linear Regression Wiki
- Random Forest Regression Wiki
- Gradient Boosting Regression Wiki
- Neural Networks Wiki
- Custom metrics in Keras: Coefficient of Determination
- Scikit-Learn LinearRegression
- Scikit-Learn RandomForestRegressor
- Scikit-Learn GradientBoostingRegressor
- Scikit-Learn Learning Curves
- Keras Documentation
- Intro to Gradient Boosting
- Gradient Boosting for Predicting House Prices
- Kaggle Master on Gradient Boosting
- Neural Networks for Regression
- Learning Curves
- California House Price predictions with Gradient Boosted Regression Trees