Predicting Business Major’s G.P.A. with Machine Learning

Daniel Z. Carter

Bushnell University

Author Note

This is my senior capstone project. My faculty mentor was Ernest Bonat, Ph.D. Computer Engineering Faculty.

Abstract

This is a write up detailing the process in which a machine learning model was used on data from Bushnell University to predict G.P.A. of Business Majors based on select features.

Predicting Business Major’s G.P.A. with Machine Learning

Research Question: Can a machine learning linear regression model successfully predict a Business Major’s G.P.A. based on credits attempted and credits earned, classes, gender, whether the student is an athlete, style of class (online or in-person), and the grade received for each class?

# Preprocessing

The first step was to gather and clean the data. The data that was used for this project was gathered from Bushnell University. The data consists of student data dating back to the 90’s and came in the form of two separate CSV files. In order to begin creating the model the data needed to be cleaned so it can be plugged into the model and not cause an error. The very first step is to import the initial libraries that will be used for the cleanup process.

import sklearn

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

The next step was to read the datasets into PANDAS.

data = pd.read\_csv('DC\_Course\_List\_Clean\_2000.csv')

gpa\_data = pd.read\_csv('DC\_Senior\_Research\_Data (1).csv')

After reading in the two separate CSV files, they needed to be merged into one dataframe.

complete\_data = data.merge(gpa\_data, on='ID')

The next step in the preprocessing process was to create a column counting the total credits for each student to use as a feature. This was done with the following code:

count = complete\_data.groupby('ID')

count['HRS\_EARNED'].count()

df = complete\_data.merge(complete\_data.groupby('ID')['HRS\_EARNED']

           .agg(['sum']),

         left\_on='ID',

         right\_index=True)

The data used in this project was provided by Bushnell University. The credit requirement to graduate is to have at least 124 credits. With this in mind, the data was sorted to only use the students who completed enough credits to graduate from the university. This was done by locating all the data in the total credits column that were greater than or equal to 124:

clean = df.loc[df['sum'] >= 124]

Another feature was created by dividing the total credits column by 4 to take an average amount of credits taken each year for each student. Another feature column was created using the same method as finding the total credits to calculate the total amount of classes that each student took. The last initial steps were to remove any rows that contained empty values and remove any column that were not going to be used as features.

The next step in the preprocessing process was to encode the data for the model. This was done using Label Encoder:

from sklearn import preprocessing

from sklearn.preprocessing import LabelEncoder

# label encoder categorical columns: four, five, eight

label\_encoder\_four = LabelEncoder()

final\_df['GENDER'] = label\_encoder\_four.fit\_transform(final\_df['GENDER'])

label\_encoder\_five = LabelEncoder()

final\_df['TRM\_CDE'] = label\_encoder\_five.fit\_transform(final\_df['TRM\_CDE'])

label\_encoder\_eight= LabelEncoder()

final\_df['Dept'] = label\_encoder\_eight.fit\_transform(final\_df['Dept'])

label\_encoder\_nine= LabelEncoder()

final\_df['Delivery'] = label\_encoder\_nine.fit\_transform(final\_df['Delivery'])

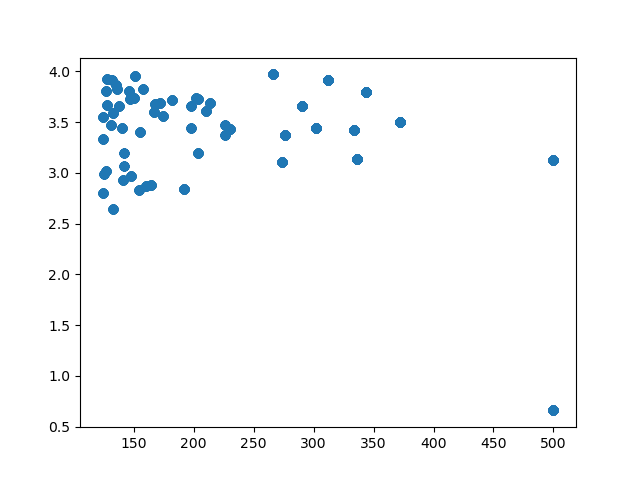
label\_encoder\_six= LabelEncoder()

final\_df['ATH\_TEAM\_MEMBR'] = label\_encoder\_six.fit\_transform(final\_df['ATH\_TEAM\_MEMBR'])

label\_encoder\_two= LabelEncoder()

final\_df['Grade'] = label\_encoder\_two.fit\_transform(final\_df['Grade'])

After encoding the data a few tests were done to determine if there were any outliers in the data that could produce misleading results. This was done by looking at quartiles and box and whisker plots as well as scatter plots:



An example of one of the scatter plots (total credits vs gpa) shows evident outliers in the data. The outliers where then removed to make the model more accurate:

#remove outliers

outlier = final\_df.loc[final\_df['sum'] <= 300]

outlier = outlier.loc[outlier['count'] <= 90]

outlier = outlier.loc[outlier['career\_gpa'] >= 2.5]

The preprocessing phase is almost complete. The last thing that was done was convert the data into the appropriate data types so that the model would accept all of the data and not error out:

outlier["HRS\_ATTEMPTED"]=pd.to\_numeric(outlier["HRS\_ATTEMPTED"],errors='coerce')

outlier = outlier.astype("float64")

## The Model

The first step in creating the model was to call in all of the appropriate libraries and packages in order to create the model:

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestRegressor

from sklearn.linear\_model import LinearRegression

from sklearn.tree import DecisionTreeRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from sklearn.metrics import r2\_score, mean\_squared\_error, mean\_absolute\_error

from scipy.stats import sem

from sklearn.model\_selection import KFold, RepeatedKFold, cross\_val\_score

The next step in creating the model was to create a function that will split our data into the training, validation, and the testing sets:

def train\_validation\_test\_split(X, y, test\_size=0.2, valid\_size=0.5, random\_state\_initial=50):

    try:

        X\_train, X\_test\_valid, y\_train, y\_test\_valid = train\_test\_split(X, y, test\_size=test\_size, random\_state=random\_state\_initial)

        X\_valid, X\_test, y\_valid, y\_test = train\_test\_split(X\_test\_valid, y\_test\_valid, test\_size=valid\_size, random\_state=random\_state\_initial)

    except:

        exception\_message = sys.exc\_info()[0]

        print("An error occurred. {}".format(exception\_message))

    return X\_train, y\_train, X\_valid, y\_valid, X\_test, y\_test

This function is very important. What is happening in this block of code is setting up the model for future use. The first line of code under the try is splitting 80% of the data into the training set. The next line under the try is then splitting the remaining 20% in half with 10% going into the validation set to check for overfitting and the last 10% going to the testing split for running through the model for final results. The next step is to split the data into your X (features) and y (target) variable. For this model the gpa was our target and the rest of the columns were used as features. The code for this is shown below:

 X = outlier.drop(labels="career\_gpa", axis=1)

y = outlier["career\_gpa"]

The following step is calling the function that was created in order to split the data so that the data is split into the appropriate data splits:

X\_train, y\_train, X\_valid, y\_valid, X\_test, y\_test = train\_validation\_test\_split(X, y, test\_size=test\_size, random\_state\_initial=1)

The data is then scaled to fit the model. This is only done on the features because the target is only one value. If y was multiple values y would also need to be scaled to fit. The code for this is:

 scaler = StandardScaler()

 scaler.fit(X\_train)

 X\_train = scaler.transform(X\_train)

 X\_valid = scaler.transform(X\_valid)

The next part of the process is defining the model to be used. Many models were tested to determine which was the most accurate. The following are the different models that were tested:

model\_regressor = RandomForestRegressor(n\_jobs=-1, random\_state=50)

 model\_regressor = SVR(kernel='rbf', C=100, gamma=0.1, epsilon=.1)

model\_regressor = SVR(kernel='linear', C=100, gamma='auto')

model\_regressor = SVR(kernel='poly', C=100, gamma='auto', degree=3, epsilon=.1, coef0=1)

 model\_regressor = DecisionTreeRegressor(random\_state=0)

 model\_regressor = MLPRegressor(random\_state=1)

 model\_regressor = xgb.XGBRegressor(n\_jobs=-1, random\_state=1, objective="reg:squarederror")

model\_regressor = LinearRegression()

The next step was to fit the model and run the predictions:

 model\_regressor.fit(X\_train, y\_train)

 y\_predicted = model\_regressor.predict(X\_valid)

The last part of the model is getting the results and calculating the errors. This is done with:

    # R2 can take values from 0 to 1. A value of 1 indicates that the regression predictions perfectly fit the data.

    r2\_score\_value = r2\_score(y\_valid, y\_predicted)

    print("valid r2 score value:")

    print(r2\_score\_value)

    print()

    # small MSE suggests the model is great at prediction

    mean\_squared\_error\_result = mean\_squared\_error(y\_valid, y\_predicted)

    # mean\_squared\_error\_result = np.sqrt(mean\_squared\_error\_result)

    print("valid mean squared error:")

    print(mean\_squared\_error\_result)

    print()

    # small MAE suggests the model is great at prediction

    mean\_absolute\_error\_result = mean\_absolute\_error(y\_valid, y\_predicted)

    print("valid mean absolute error:")

    print(mean\_absolute\_error\_result)

    print()

    #exit()

    X\_test = scaler.transform(X\_test)

    y\_predicted = model\_regressor.predict(X\_test)

    # R2 can take values from 0 to 1. A value of 1 indicates that the regression predictions perfectly fit the data.

    r2\_score\_value = r2\_score(y\_test, y\_predicted)

    print("test r2 score value:")

    print(r2\_score\_value)

    print()

    # small MSE suggests the model is great at prediction

    mean\_squared\_error\_result = mean\_squared\_error(y\_test, y\_predicted)

    # mean\_squared\_error\_result = np.sqrt(mean\_squared\_error\_result)

    print("test mean squared error:")

    print(mean\_squared\_error\_result)

    print()

    # small MAE suggests the model is great at prediction

    mean\_absolute\_error\_result = mean\_absolute\_error(y\_test, y\_predicted)

    print("test mean absolute error:")

    print(mean\_absolute\_error\_result)

    print()

Which resulted in these results:



### **Conclusion**

This was a very long and tedious project. The code shown throughout this project is from the final python file that was written for this model. Several attempts were made until the final product was created. The main issue that was found initially was that the data was overfitting to the training set causing the results to be misleading. A cross validation test was conducted as a first attempt to solve this issue. The final resolution was adding in the validation split to negate overfitting. This was not the case for every model as half of the models still overfit the data. This is significant improvement as all of the models were overfitting previously. From the results it is evident that the XGBRegressor model was the most accurate with a tested R^2 value at 0.96. This means that we can expect that roughly 96% of the predictions made from the model are being predicted correctly.

References

https://machinelearningmastery.com/repeated-k-fold-cross-validation-with-python/

Footnotes

1Add footnotes, if any, on their own page following references. The body of a footnote, such as this example, uses the Normal text style. (Note: If you delete this sample footnote, don’t forget to delete its in-text reference as well. That’s at the end of the sample Heading 2 paragraph on the first page of body content in this template.)

Tables

Table 1

Table Title

| Column Head | Column Head | Column Head | Column Head | Column Head |
| --- | --- | --- | --- | --- |
| Row Head | 123 | 123 | 123 | 123 |
| Row Head | 456 | 456 | 456 | 456 |
| Row Head | 789 | 789 | 789 | 789 |
| Row Head | 123 | 123 | 123 | 123 |
| Row Head | 456 | 456 | 456 | 456 |
| Row Head | 789 | 789 | 789 | 789 |

Note: Place all tables for your paper in a tables section, following references (and, if applicable, footnotes). Start a new page for each table, include a table number and table title for each, as shown on this page. All explanatory text appears in a table note that follows the table, such as this one. Use the Table/Figure style, available on the Home tab, in the Styles gallery, to get the spacing between table and note. Tables in APA format can use single or 1.5-line spacing. Include a heading for every row and column, even if the content seems obvious. A table style has been setup for this template that fits APA guidelines. To insert a table, on the Insert tab, click Table.

Figures



Figure 1. Include all figures in their own section, following references (and footnotes and tables, if applicable). Include a numbered caption for each figure. Use the Table/Figure style for easy spacing between figure and caption.

For more information about all elements of APA formatting, please consult the APA Style Manual, 6th Edition.