20/12/2021
Machine Learning

Build a classifier

Daniel Krasovski
C18357323

Table of Contents

How I solved the problem......3
- Linear Regression......3
- Decision Tree......3
- Naïve Bayes......4
- KNN......4

Decisions Made......5

Why I chose Naïve Bayes......6

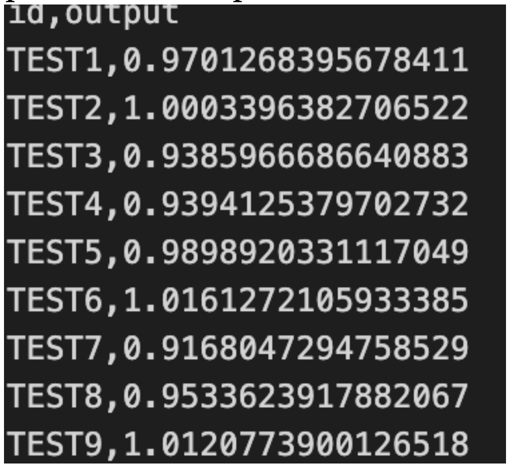
Issues with Data......6

How testing was performed......6

How I solved the problem

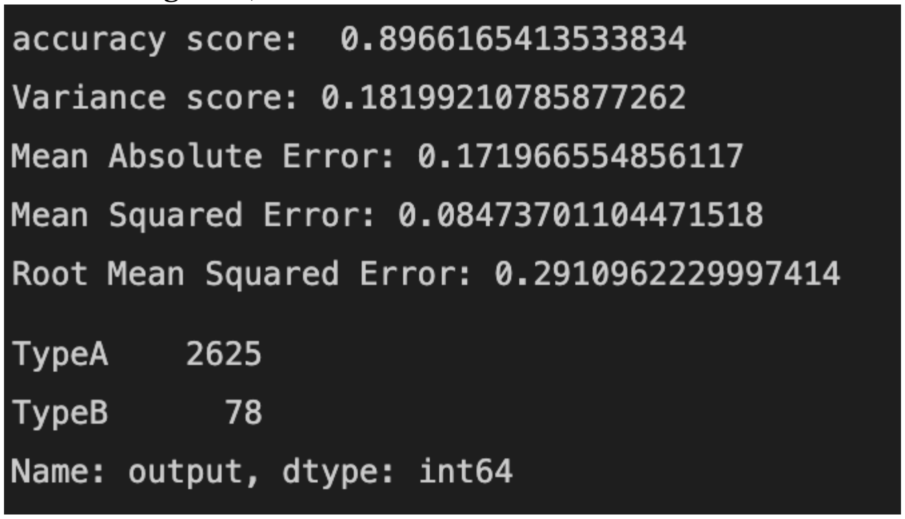
Linear Regression

To solve the problem, I went with the approach of making all the classifiers and then testing to see which classifier gave me the best option. I started off with Linear Regression however I realized early on that it was not the best classifier as it does not work well with predicting binary outputs. Instead of predicting either 1 or 0, it would predict decimal point numbers. So I would get outputs like this:



```
id,output
TEST1,0.9701268395678411
TEST2,1.0003396382706522
TEST3,0.9385966686640883
TEST4,0.9394125379702732
TEST5,0.9898920331117049
TEST6,1.0161272105933385
TEST7,0.9168047294758529
TEST8,0.9533623917882067
TEST9,1.0120773900126518
```

To solve this, I just rounded the numbers however I was not satisfied with this, even though with the rounding, the accuracy score was 89%, the variance score was 18%. There was also 78 type B's, which is 2% of all the prediction data, whereas in the testing and training data, it was 11%.

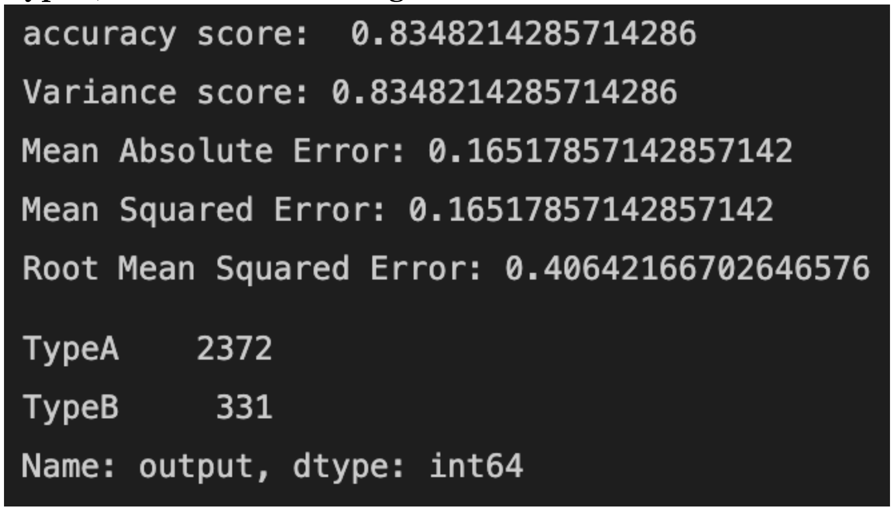


```
accuracy score:  0.8966165413533834
Variance score: 0.18199210785877262
Mean Absolute Error: 0.171966554856117
Mean Squared Error: 0.08473701104471518
Root Mean Squared Error: 0.2910962229997414

TypeA    2625
TypeB      78
Name: output, dtype: int64
```

Decision Tree

Because of these outputs, I was not satisfied with Linear Regression, so I went on to do Decision tree. This gave a more expected output with 83% accuracy and variance score, with low error scores. However, the accuracy score of 83% was too low for me, so I moved on. I was more satisfied with the amount of typeB with 12% of the outputs being TypeB, closer to the training dataset.

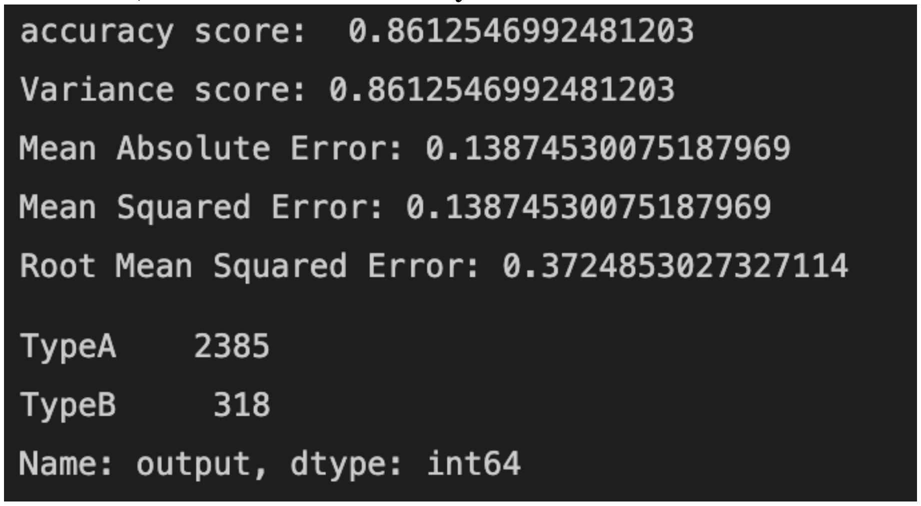


```
accuracy score:  0.8348214285714286
Variance score: 0.8348214285714286
Mean Absolute Error: 0.16517857142857142
Mean Squared Error: 0.16517857142857142
Root Mean Squared Error: 0.40642166702646576

TypeA    2372
TypeB     331
Name: output, dtype: int64
```

Naïve Bayes

After this, I went on to Naïve Bayes. This was the one I was the most satisfied with.



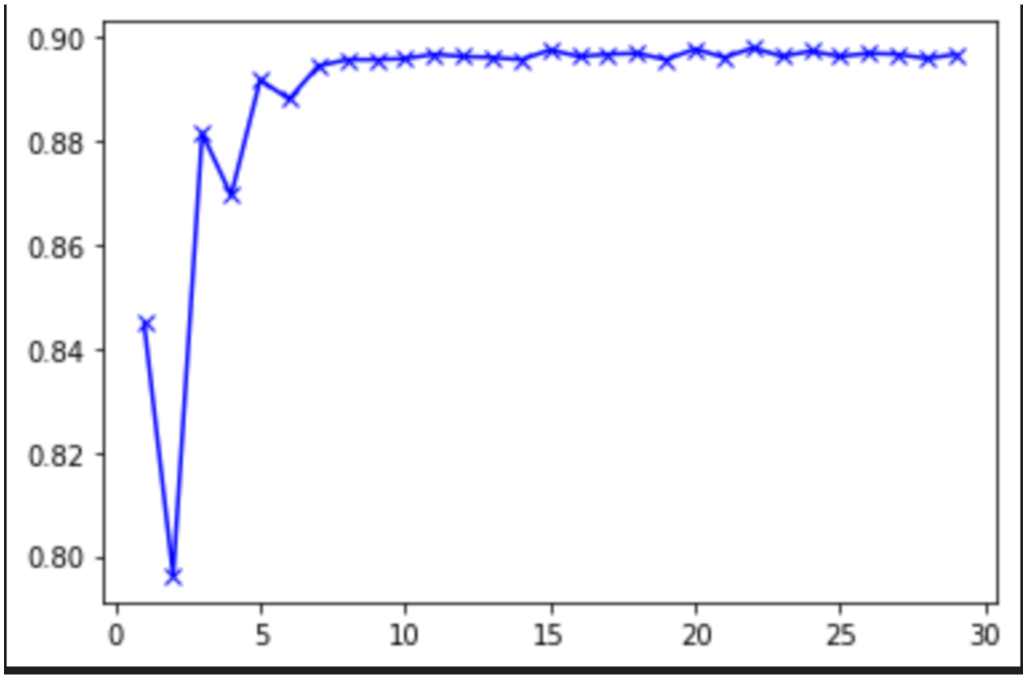
```
accuracy score:  0.8612546992481203
Variance score: 0.8612546992481203
Mean Absolute Error: 0.13874530075187969
Mean Squared Error: 0.13874530075187969
Root Mean Squared Error: 0.3724853027327114

TypeA    2385
TypeB     318
Name: output, dtype: int64
```

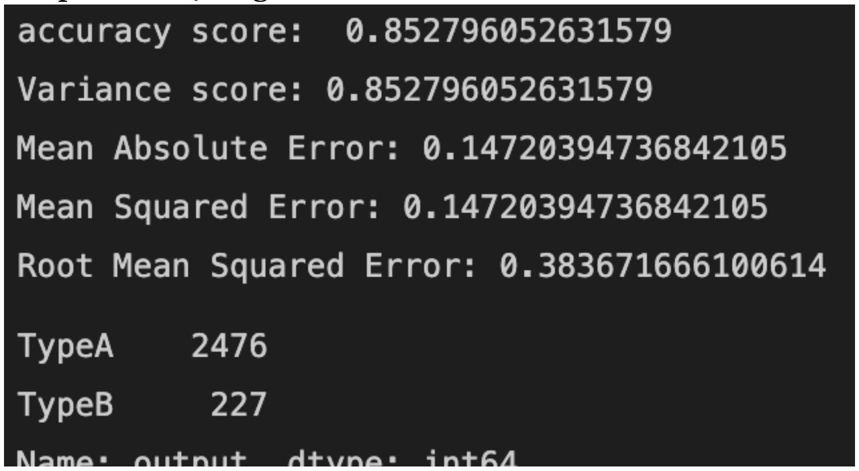
With accuracy score of 86% and lower error scores, and giving closer to 11% TypeB outputs. Everything just made perfect sense with all the outputs.

KNN

But as I only had one more to test out, I went for it, with KNN, this one was the most interesting for me because of what I learned. Going using the method I used with all the others. I started off with 5 neighbors and got a score of 89%. I then created a function to compare the most optimal number of neighbors and found that 7 was the most optimal, with the scores leveling out after around 10-20 neighbors.



However, the higher the neighbors, the lower of TypeB's I got, however the accuracy stayed the same. This was a warning sign for me and why I ended up not going with it. Output with 4 neighbors:



```
accuracy score:  0.852796052631579
Variance score: 0.852796052631579
Mean Absolute Error: 0.14720394736842105
Mean Squared Error: 0.14720394736842105
Root Mean Squared Error: 0.383671666100614

TypeA    2476
TypeB     227
Name: output, dtype: int64
```

Decisions Made

Decision making was easy for me, for choosing the classifier as I mentioned above. The other decision that I had to make was on the data preprocessing. I decided to assign relevant datatypes to all the column types, depending on the data, either float, int, binary or categorical. And then for the categorical datatypes, I converted them to dummy variables. This was all I needed to do for the data preprocessing.

There was also the decision on whether to drop the output column from the training and testing. If it was not dropped the accuracy score would shoot up to 97-100%. However, I found this to be misleading and it did not give the desired predictions for the actual data that needed to be predicted. ID was also dropped, but that is self-explanatory as it is does not contribute value to the predictions.

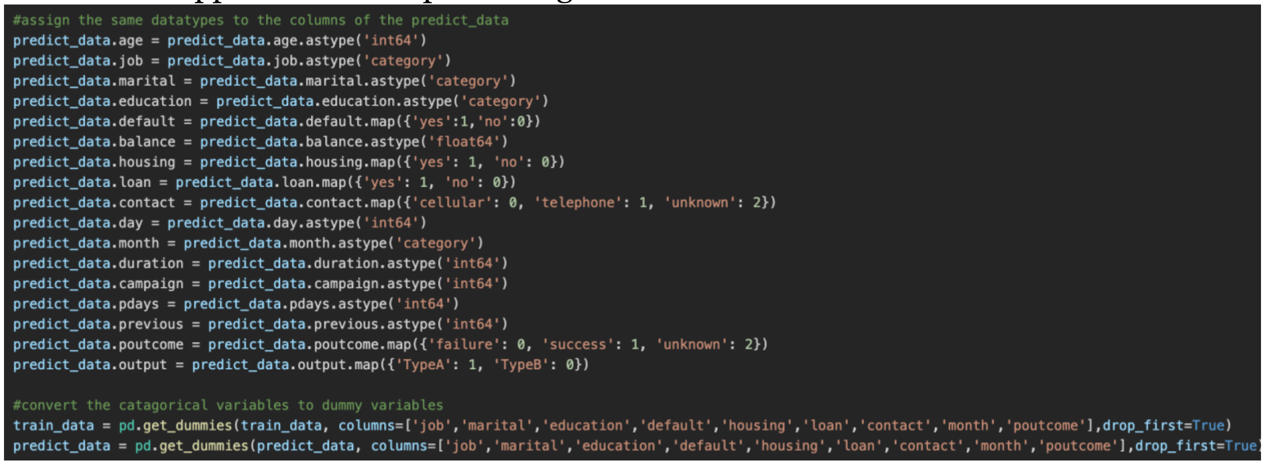
There was also decision made about the test size and random state in the train_test_split function. I played around with the numbers and found that test size of 0.2 and random state of 2 worked the best, however test size of 0.35 and random state of 4 was also very good

Why I chose Naïve Bayes

As mentioned before, the reason why I chose Naïve Bayes, was because it clicked all the check boxes for me. Good Accuracy score, Lower error scores and in around the 11% of typeB's. There was also nothing that stood out to me that seemed out of place, which there was with other classifiers

Issues with Data

There were no major issues with data. The only decisions made was how to turn all the datatypes into numerical values, which was completed using mapping and using dummy variables. Snippet of the data processing code:

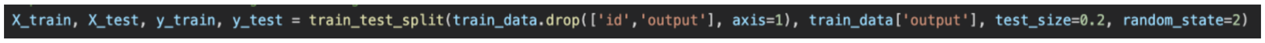


```
#assign the same datatypes to the columns of the predict_data
predict_data.age = predict_data.age.astype('int64')
predict_data.job = predict_data.job.astype('category')
predict_data.marital = predict_data.marital.astype('category')
predict_data.education = predict_data.education.astype('category')
predict_data.default = predict_data.default.map({'yes':1,'no':0})
predict_data.balance = predict_data.balance.astype('float64')
predict_data.housing = predict_data.housing.map({'yes': 1, 'no': 0})
predict_data.loan = predict_data.loan.map({'yes': 1, 'no': 0})
predict_data.contact = predict_data.contact.map({'cellular': 0, 'telephone': 1, 'unknown': 2})
predict_data.day = predict_data.day.astype('int64')
predict_data.month = predict_data.month.astype('category')
predict_data.duration = predict_data.duration.astype('int64')
predict_data.campaign = predict_data.campaign.astype('int64')
predict_data.pdays = predict_data.pdays.astype('int64')
predict_data.previous = predict_data.previous.astype('int64')
predict_data.poutcome = predict_data.poutcome.map({'failure': 0, 'success': 1, 'unknown': 2})
predict_data.output = predict_data.output.map({'TypeA': 1, 'TypeB': 0})

#convert the catagorical variables to dummy variables
train_data = pd.get_dummies(train_data, columns=['job','marital','education','default','housing','loan','contact','month','poutcome'],drop_first=True)
predict_data = pd.get_dummies(predict_data, columns=['job','marital','education','default','housing','loan','contact','month','poutcome'],drop_first=True
```

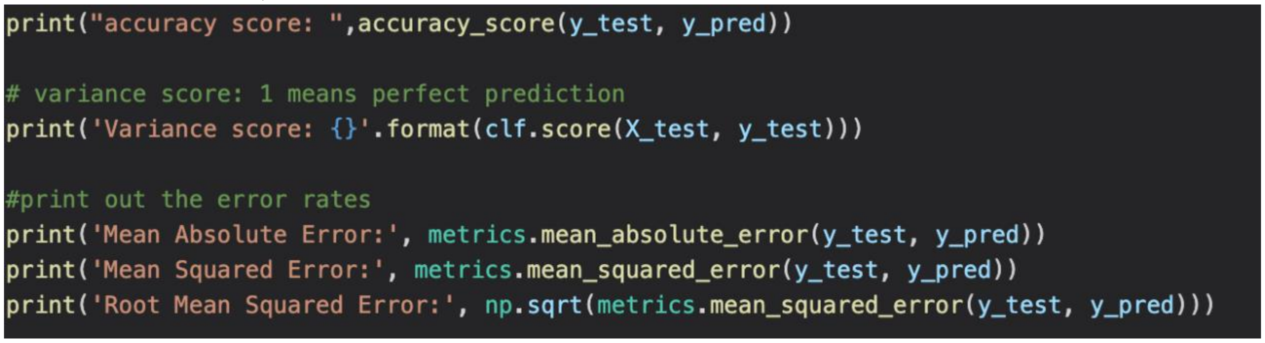
How testing was performed

Testing was performed using accuracy score and error rates. Using the inbuilt sklearn functions. For the training and testing, I used train_test_split to split up the data.



```
X_train, X_test, y_train, y_test = train_test_split(train_data.drop(['id','output'], axis=1), train_data['output'], test_size=0.2, random_state=2)
```

And for the scores, these were the functions used:



```
print("accuracy score: ",accuracy_score(y_test, y_pred))

# variance score: 1 means perfect prediction
print('Variance score: {}'.format(clf.score(X_test, y_test)))

#print out the error rates
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

Depending on the scores given, I decided what classifier I was going to use