PREDICT BLOOD DONATION

PROJECT - REPORT

Vineet Singh

11/13/20

Data Analyst Project

ACKNOWLEDGEMENT

The internship opportunity I had with MedTourEasy was a great chance for learning and professional development. Therefore, I consider myself as a very lucky individual as I was provided with an opportunity to be a part of it. I am also grateful for having a chance to work with so many new tools and technologies and professionals who led me though this internship period.

I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and I will continue to work on their improvement, in order to attain desired career objectives. Hope to continue cooperation with all of you in the future. Sincerely,

Vineet Singh

13/11/2020

Bangalore

TABLE OF CONTENT

CHAPTER - 1

INTRODUCTION

Blood transfusion saves lives - from replacing lost blood during major surgery or a serious injury to treating various illnesses and blood disorders. Ensuring that there's enough blood in supply whenever needed is a serious challenge for the health professionals. According to WebMD, "about 5 million Americans need a blood transfusion every year".

This project is about predicting that if a donor has donated blood in March 2007. We want to predict whether or not a donor will give blood the next time the vehicle comes to campus. Forecasting blood supply is a serious and recurrent problem for blood collection managers. In this Project, we have worked with data collected from the donor database of Blood Transfusion Service Center. The dataset, obtained from the Machine Learning Repository, consists of a random sample of 748 donors.

For this process we have used Automated Machine Learning Library **TPOT,** it will create a model, and fit our dataset onto it, which will predict the target variable, which will tell us that if the doner will donate blood again.

CHAPTER - 2

ABOUT THE DATASET

The data that is being used is the transfusion data, it has been taken from the donor database of Blood Transfusion Service Center. Our dataset is from a mobile blood donation vehicle in Taiwan. The Blood Transfusion Service Center drives to different universities and collects blood as part of a blood drive.

The data is stored in Transfusion.data file and it is structured according to RFMTC marketing model (a variation of RFM). RFM stands for Recency, Frequency and Monetary Value and it is commonly used in marketing for identifying your best customers. In our case, our customers are blood donors.

RFMTC is a variation of the RFM model. Below is a description of what each column means in our dataset:

- **R** (Recency - months since the last donation)
- **F** (Frequency - total number of donation)
- **M** (Monetary - total blood donated in c.c.)
- **T** (Time - months since the first donation)
- a binary variable representing whether he/she donated blood in March 2007

The dataset contains 5 columns, as mentioned above respectively, and 748 data entries, starting from 0 to 747. The data has been imported in the Python notebook using the **pd.read_csv** function in the Pandas Library, in a variable called transfusion. The last column of the data contains a binary distribution, in which **1** represents that a person has donated blood after March 2007, and **0** represents that a person has not donated blood after March 2007.

We can get a basic overview of the data set, a sort of information about it, through the **transfusion.info()** function. This gives the following result which describes about

the columns, it tells about the null values and the data type of the columns. In our database we don't have any null values and the datatype of every column is int64. The result can be seen below,

```
transfusion.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 748 entries, 0 to 747
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Recency (months)        748 non-null    int64
 1   Frequency (times)       748 non-null    int64
 2   Monetary (c.c. blood)   748 non-null    int64
 3   Time (months)           748 non-null    int64
 4   target                  748 non-null    int64
dtypes: int64(5)
memory usage: 29.3 KB
```

Fig 1.1: Information about Dataset

The data can be viewed using **transfusion.head()** this will give the first five rows of the dataset, which is given below,

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	whether he/she donated blood in March 2007
0	2	50	12500	98	1
1	0	13	3250	28	1
2	1	16	4000	35	1
3	2	20	5000	45	1
4	1	24	6000	77	0

Fig 1.2: Dataset Representation

CHAPTER - 3

VISUALIZATION AND STATISTICS

Visualization and Statistical Analysis are a major part of any data analysis project, the initial step is to understand about the dataset, and the problem we are dealing with. Visualizing the data and the results at every step gives us a good idea about the solution.

In the first step we have used the countplot method of the seaborn library, we have counted the number of 1's and 0's in the column 'weather he/she donated blood in March 2007', this column in further changed to 'target'. The plot shows us the distribution of the people who donated blood and who didn't in the month of March. We can see clearly maximum people did not donate blood in the month of march.

```
transfusion.target.value_counts()
```

```
0    570
1    178
Name: target, dtype: int64
```

Fig 2.1: Values in target column

The above code snippet represents the count of the values in the target column, it gives a count of 0's & 1's. The Same thing is represented below through the countplot method. This distribution shows that there is a vast difference between the values of the people who donated blood in March to the people who did not. It shows that majority of the people did not donate blood in March 2007.

```
sns.countplot(data = transfusion, x = 'whether he/she donated blood in March 2007')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f12c0f6c550>
```

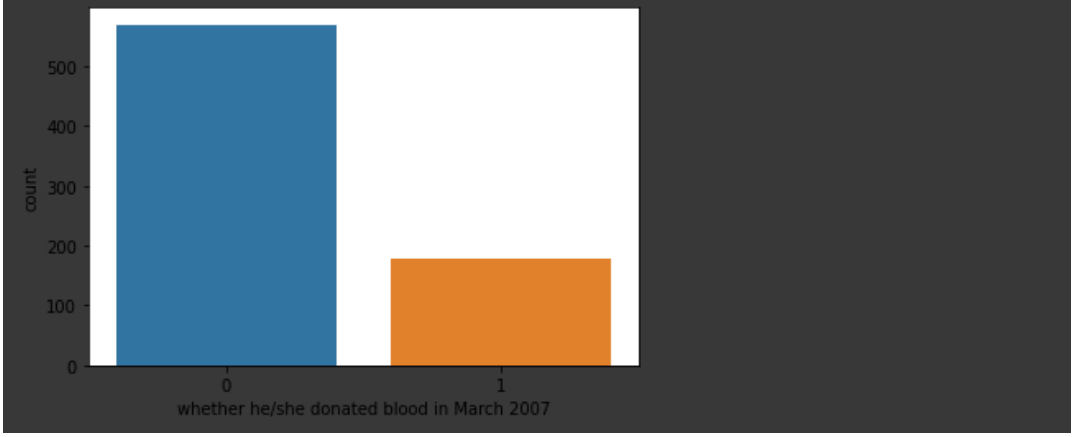

Fig 2.2: Countplot of target column

```
transfusion.describe()
```

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	target
count	748.000000	748.000000	748.000000	748.000000	748.000000
mean	9.506684	5.514706	1378.676471	34.282086	0.237968
std	8.095396	5.839307	1459.826781	24.376714	0.426124
min	0.000000	1.000000	250.000000	2.000000	0.000000
25%	2.750000	2.000000	500.000000	16.000000	0.000000
50%	7.000000	4.000000	1000.000000	28.000000	0.000000
75%	14.000000	7.000000	1750.000000	50.000000	0.000000
max	74.000000	50.000000	12500.000000	98.000000	1.000000

Fig 2.3: Describe function

We can get a basic summary of the dataset at the initial stage of data analysis, through the describe command **transfusion.describe(),** it gives a count of the data in every row, the mean of the data and many such functions like standard deviation, minimum and maximum, etc.

In this we can see that every row count is 748, which means that there are no missing values in the dataset. We can also see that on an average everyone has donated blood 5 times and in the time span of 9 months.

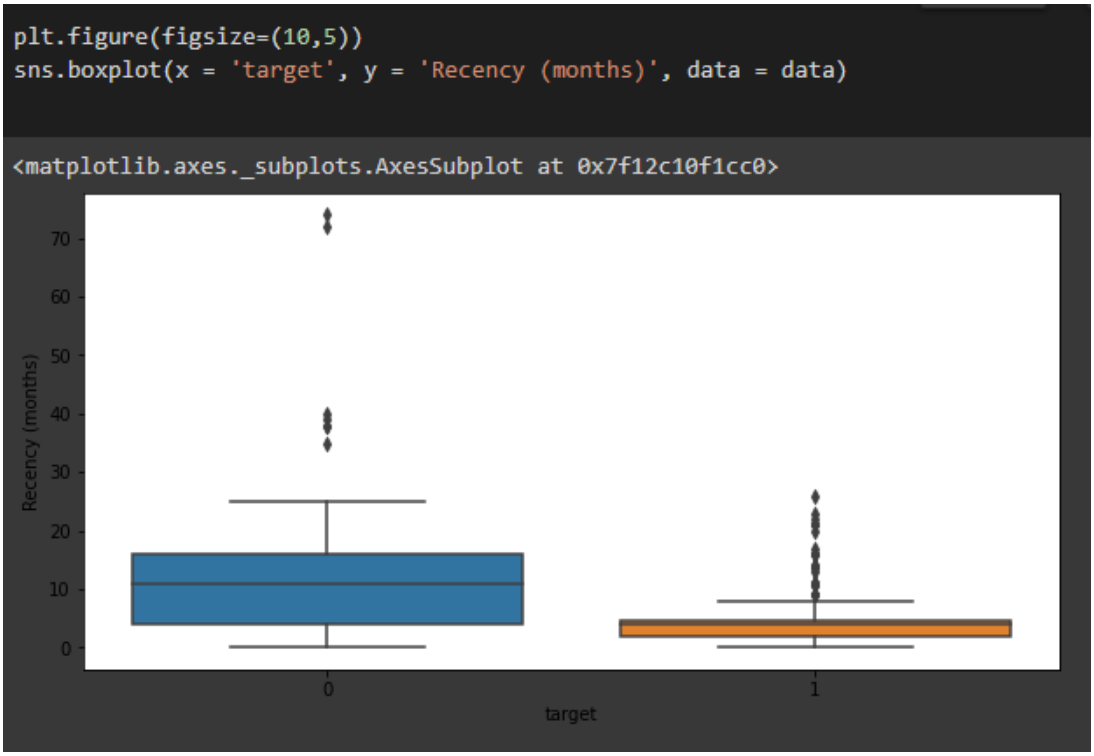

Fig 2.4: Boxplot of target against Recency

In the above figure we can see a comparison between the target variable and the Recency from the dataset. We have these to variable from a boxplot in the seaborn library. From the above graph we can see that the people who donated blood in March 2007, there recency in lesser than the people who did not donate blood in March 2007.

Data Correlation: Is a way to understand the relationship between multiple variables and attributes in our dataset. Using Correlation, we can get some insights such as:

- One or multiple attributes depend on another attribute or a cause for another attribute.
- One or multiple attributes are associated with other attributes.

The below table is the correlation table of our dataset, it is generated by the **corr()** function. As we can see in the above table, we have both positive and negative values. Positive Correlation: means that if feature A increases then feature B also increases or if feature A decreases then feature B also decreases. Both features move in tandem and they have a linear relationship.

Negative Correlation: means that if feature A increases then feature B decreases and vice versa.

```
transfusion.corr()
```

	Recency (months)	Frequency (times)	Monetary (c.c. blood)	Time (months)	target
Recency (months)	1.000000	-0.182745	-0.182745	0.160618	-0.279869
Frequency (times)	-0.182745	1.000000	1.000000	0.634940	0.218633
Monetary (c.c. blood)	-0.182745	1.000000	1.000000	0.634940	0.218633
Time (months)	0.160618	0.634940	0.634940	1.000000	-0.035854
target	-0.279869	0.218633	0.218633	-0.035854	1.000000

Fig 2.5: Correlation table of the data

We can also represent the correlation through a heatmap in the seaborn library. It represents the correlation in a color-coded format, and gives a legend of the colors at the side of the graph. We can see that we have a positive correlation between Frequency and Time. We can see the values of the correlation in each cell, by putting the value of annot = True.

```
sns.heatmap(transfusion.corr(), annot=True, linewidths=0.3, cmap="YlGnBu")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f574d3b4eb8>
```

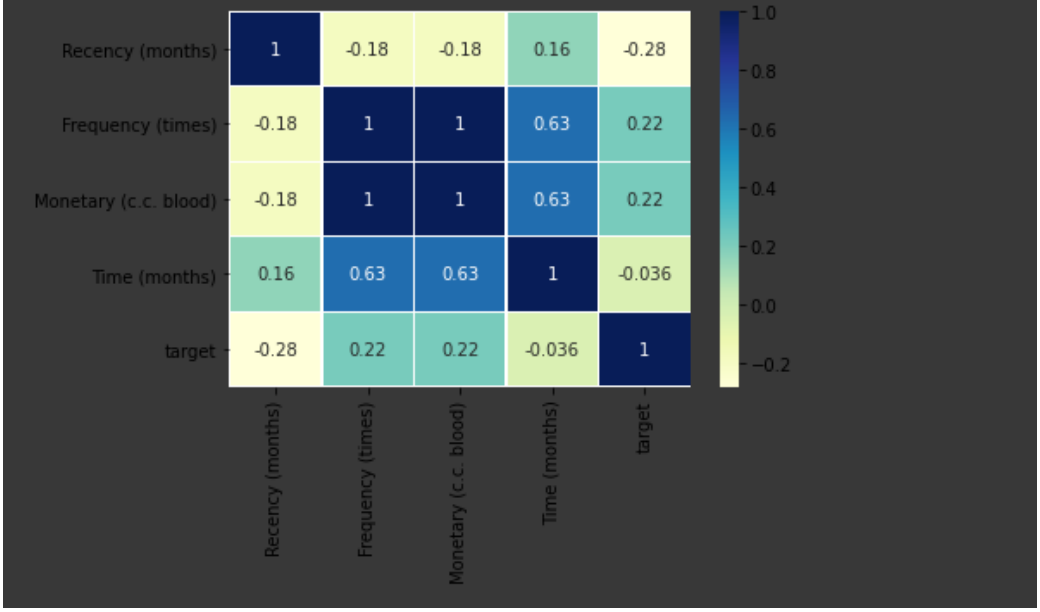

Fig 2.6: Heatmap of the correlation

CHAPTER - 4

PROCESS EXPLANATION

We were assigned the project of Blood Donation Prediction, where we needed to predict weather that person will donate blood in March 2007. This project file was given in a Python Notebook format, along with a transfusion data file. The complete project was divided into tasks, from adding the data to the python notebook, to running the TPOT library and Logistic Regression on the training data to predict the values and getting the accuracy through the AUC Score.

The first few initial steps were importing the required libraries such as pandas, seaborn and reading the data into the notebook though the pandas library and basic Visualization. After all the basic exploration, the next step was to create the target column, which is the value we want to predict. The last column in our data is 'whether he/she donated blood in March 2007', we converted it to 'target' column by the rename function, and putting inplace as True to change the value in the original database.

```
transfusion.rename(
    columns={'whether he/she donated blood in March 2007': 'target'},
    inplace=True
)
```

Fig 4.1: Rename the target column

After these steps we can start the Machine Learning process, the first step is to start the Test and Train split which is splitting the data into 4 parts, X_train, X_test, Y_train, Y_test, In this we split our dataset, in the testing and training part, we have divided our dataset as 75% Train, and 25% Test. We have used the **test_train_split** function from **sklearn_library**. We have initialized the random_state variable as 42 and test_size as 0.25, keeping the random state fixed makes the test and train data fixed.

```
# Import train_test_split method
from sklearn.model_selection import train_test_split

# Split transfusion DataFrame into
# X_train, X_test, y_train and y_test datasets,
# stratifying on the `target` column
X_train, X_test, y_train, y_test = train_test_split(
    transfusion.drop(columns='target'),
    transfusion.target,
    test_size=0.25,
    random_state=42,
    stratify=transfusion.target
)
```

Fig 4.2: Splitting the data

The next task is to install and the import the TPOT library for further processing of the testing dataset. First, we installed TPOT library through **!pip install topt**, then we import the TPOT library and the **roc_auc_score** to measure the accuracy of the model.

```
from tpot import TPOTClassifier
from sklearn.metrics import roc_auc_score
```

Fig 4.3: Importing TPOT and AUC

The next step we initialize the TPOTClassifier, by defining the basic attributes of the classifier function, we have kept the scoring method as roc_auc, defined a random state, generation & population size.

```
# Instantiate TPOTClassifier
tpot = TPOTClassifier(
    generations=5,
    population_size=20,
    verbosity=2,
    scoring='roc_auc',
    random_state=42,
    disable_update_check=True,
    config_dict='TPOT light'
)
```

Fig 4.4: Initializing TPOT Classifier

Next we fit our train data into the model by **tpot.fit(X_train, y_train)**, after the fitting of the data onto the TPOT model, we now have to generate the AUC score by

comparing the results of **tpot.predict_proba**, with **y_test**. We also print out the Best Pipeline Steps. The output is shown below,

```
Generation 1 - Current best internal CV score: 0.7422459184429089

Generation 2 - Current best internal CV score: 0.7422459184429089

Generation 3 - Current best internal CV score: 0.7422459184429089

Generation 4 - Current best internal CV score: 0.7422459184429089

Generation 5 - Current best internal CV score: 0.7456308339276876

Best pipeline: MultinomialNB(Normalizer(input_matrix, norm=l2), alpha=0.001, fit_prior=True)

AUC score: 0.7637

Best pipeline steps:
1. Normalizer(copy=True, norm='l2')
2. MultinomialNB(alpha=0.001, class_prior=None, fit_prior=True)
```

Fig 4.5: Fitting data on TPOT Classifier

The Generations are the result of the fitting of the data onto the tpot model, we can see different accuracy, the tpot model selected the best model and our final accuracy was 0.7637.

In the nest step we need to check the Variance, the tpot library has picked up Logistic Regression as the best model for our data.

Logistic Regression works on the assumption that the features are in a linear fashion.

```
3   X_train.var().round(3)
```

```
Recency (months)              66.929
Frequency (times)             33.830
Monetary (c.c. blood)    2114363.700
Time (months)                611.147
dtype: float64
```

Fig 4.6: Variance before Normalization

We can see that the variance of the column ‘Monetary (c.c. blood)’ is not in the range as compared to other columns. So we need to normalize the column, for that we are using the Log Normalization method from the numpy library.

We can see the new variance is in the range after the normalization.

```
Recency (months)      66.929
Frequency (times)     33.830
Time (months)        611.147
monetary_log           0.837
dtype: float64
```

Fig 4.7: Variance after Normalization

We now need to Fit our dataset into the Logistic Regression Model and train the model, and then we find the AUC score again to compare the output prediction of both the models.

```
14  logreg_auc_score = roc_auc_score(y_test, logreg.predict_proba(X_test_normed)[:, 1])
15  print(f'\nAUC score: {logreg_auc_score:.4f}')
```

```
AUC score: 0.7890
```

Fig 4.8: Logreg AUC score

As we can see the AUC score has increased and we are getting better results.

CHAPTER – 5

RESULT & CONCLUSION

In this notebook, we explored automatic model selection using TPOT and AUC score we got was 0.7850. This is better than simply choosing 0 all the time (the target incidence suggests that such a model would have 76% success rate). We then log normalized our training data and improved the AUC score by 0.5%. In the field of machine learning, even small improvements in accuracy can be important, depending on the purpose. Benefit of using logistic regression model is that it is interpretable. We can analyze how much of the variance in the response variable (target) can be explained by other variables in our dataset.

We can check the output of the prediction of the TPOT library and the Logistic Regression model after the normalization below,

```
tpot.predict(X_test)
```

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

Fig 5.1: TPOT predictions

```
logreg.predict(X_test_normed)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Fig 5.2: Logreg predictions

As we can see the accuracy has been increased after the normalization,

```
# Importing itemgetter
from operator import itemgetter

# Sort models based on their AUC score from highest to lowest
sorted(
    [('tpot', tpot_auc_score), ('logreg', logreg_auc_score)],
    key=itemgetter(1),
    reverse=True)

```

```
[('logreg', 0.7890178003814368), ('tpot', 0.7637476160203432)]
```

Fig 5.3: AUC score comparison

We can conclude that the demand for blood fluctuates throughout the year. As one prominent example, blood donations slow down during busy holiday seasons. An accurate forecast for the future supply of blood allows for an appropriate action to be taken ahead of time and therefore saving more lives.