

Blood Donation Analysis



Kübra Nur Tiryaki

Software Engineering student, İstanbul, Türkiye

Courriel: kubratiryaki@stu.aydin.edu.tr

Abstract My aim is building a model which can identify who is likely to donate blood again. There are lots of methods we can use to do this. You will see all the models used in the project in the rest of my report. In our project called Blood Donation Analysis, I decided on Support Vector Machine as the method that gives the most appropriate results. The accuracy score is 0.78, and the ROC AUC score is 0.79. AUC ROC stands for “Area Under the Curve” of the “Receiver Operating Characteristic” curve.

Keywords blood, donation, Support Vector Machine, machine learning, bioinformatics

INTRODUCTION

Donating blood is one of the most useful discoveries in human history. In this project, we calculated how inclined blood donors are to donate blood again.

I implemented 5 different models:

- Logistic Regression
- Support Vector Machine
- Random Forest
- Decision Tree
- Multi-Layer Classifier

Then I decided which of them was the most suitable for my project. As a result of the calculations I made, I saw that the most accurate result was achieved by using the Support Vector Machine model.

METHODOLOGY

Libraries

- Pandas: It is a software library written in the Python programming language for data manipulation and analysis.
- Matplotlib: It is a drawing library for the Python programming language and numerical math extension NumPy. Used for visualization.
- Seaborn: It is a Python visualization library based on the MatPlotLib library. It can also be easily integrated into the data structure of the Pandas library. It provides a very useful code structure, especially for statistical graphs. Used for visualization.
- `train_test_split` from `sklearn.model_selection` : Library for splitting model into train and test and for data transformation.
- `confusion_matrix`, `accuracy_score`, `roc_auc_score` from `sklearn.metrics` : Metrics used to examine and visualize the accuracy of the model.
- `RandomForestClassifier` from `sklearn.ensemble`: imports Random Forest Classifier model to compare with other models.
- `LogisticRegression` from `sklearn.linear_model`: imports Logistic Regression model to compare with other models.
- `DecisionTreeClassifier` from `sklearn.tree`: imports Decision Tree Classifier model to compare with other models.
- `MLPClassifier` from `sklearn.neural_network`: imports Multi-Layer Perceptron Classifier model to compare with other models.
- `SVC` from `sklearn.svm`: imports Support Vector Classifier model to compare with other models.

Datasets

Train dataset: `blood-train.csv`, its shape is (576,5). The 5 columns are: Months since Last Donation, Number of Donations, Total Volume Donated (c.c.), Months since First Donation and Made Donation in March 2007.

Made Donation in March 2007 is the dependent variable. I store it in Y.

Test dataset: `blood-test.csv`, its shape is (200,4). The 4 columns are: Months since Last Donation, Number of Donations, Total Volume Donated (c.c.), Months since First Donation

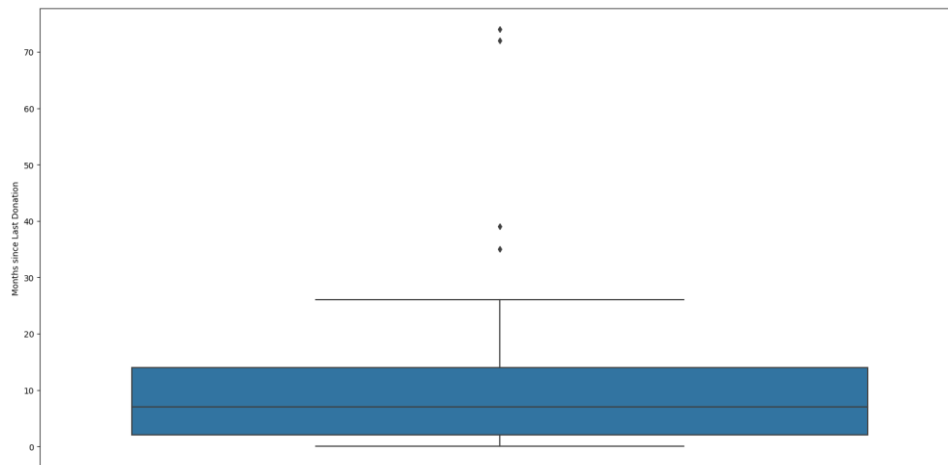


Fig. 1 Months since Last Donation

We see from Fig 1 that the max people have donated blood in nearby 10 months.

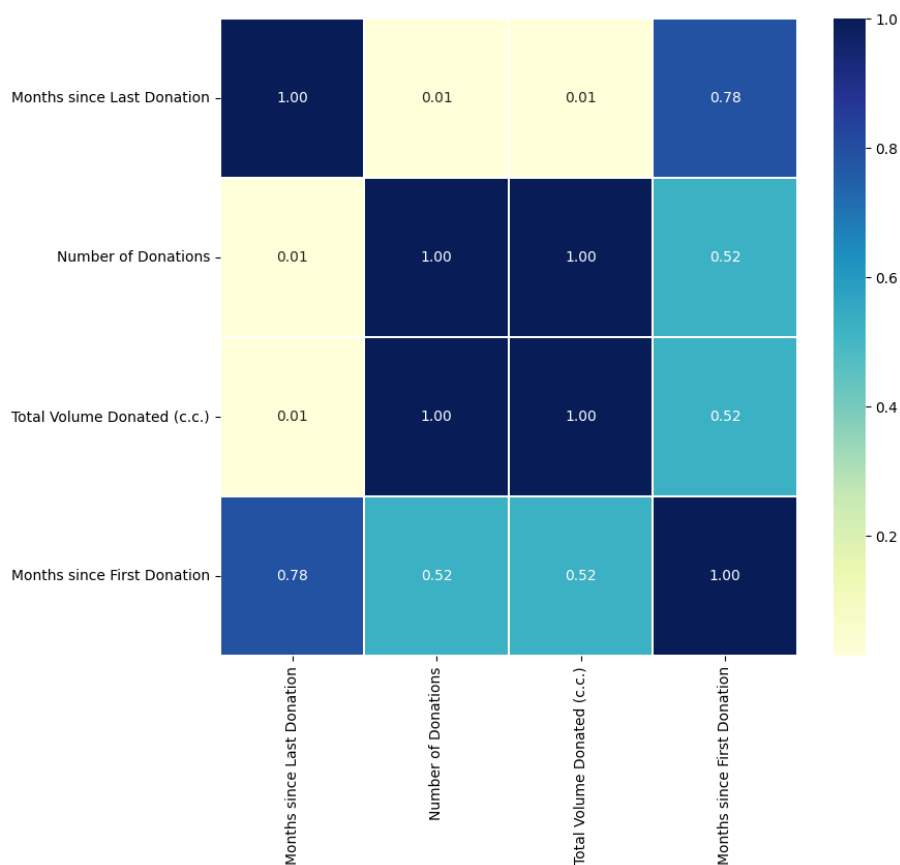


Fig. 2 Correlation between all variables

In Fig 2 we can check how different variable are related.

Feature Engineering

Volume donated is also a good feature to know whether the donor will donate or not.

```
[23] #Creating new variable for calculating how many times a person have donated
      X["Donating for"] = X["Months since First Donation"] - X["Months since Last Donation"]
```

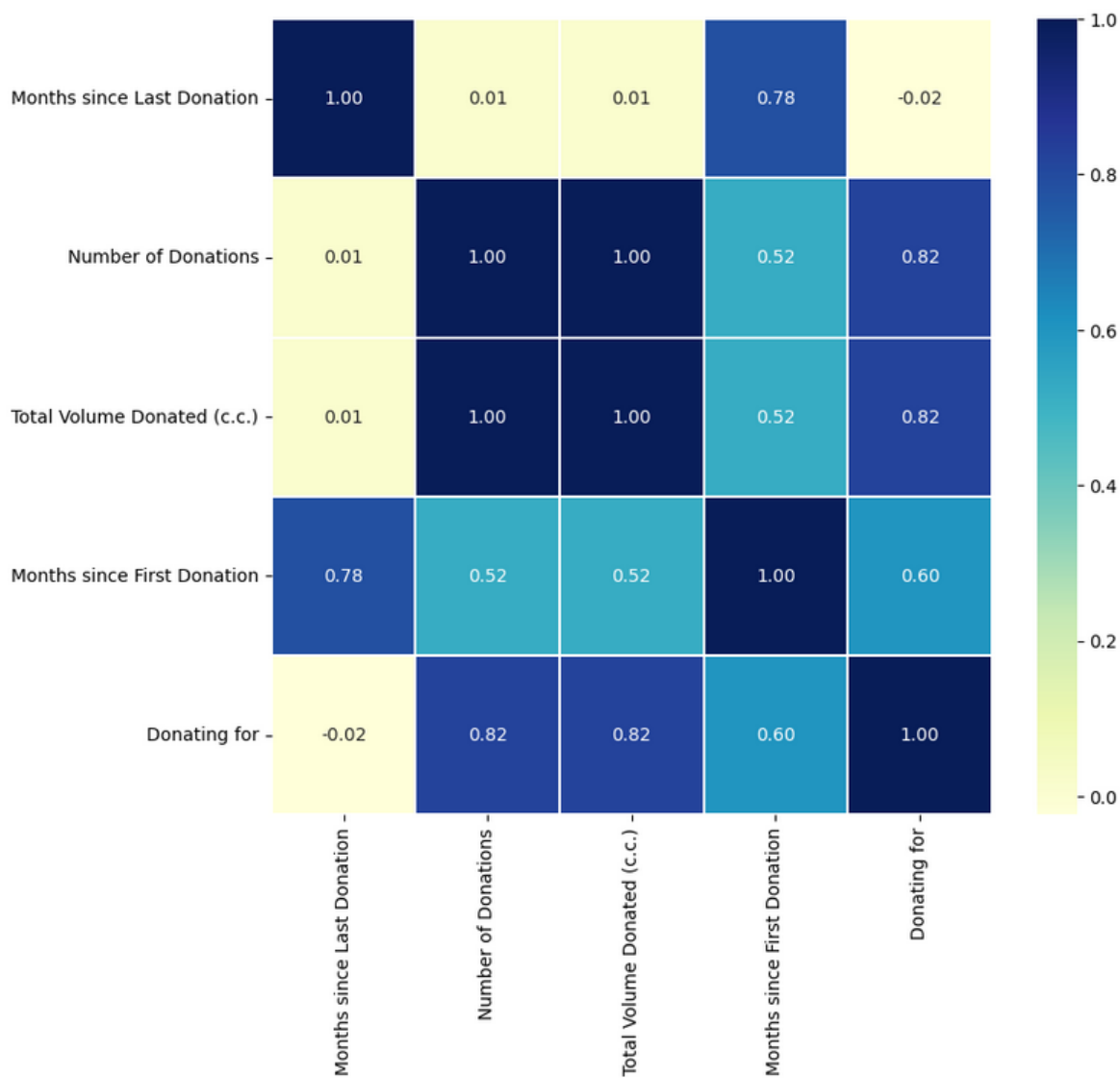


Fig. 3 Correlation between all variables after adding “Donating for”

Feature Transformation

StandardScaler from sklearn.preprocessing: Scaled the independent variables to be accurate range to create model.

Model Building

By using train_test_split() method, we splitted the train and test sets.

Steps to Follow

- Create the object
- Do the necessary hyperparameter tuning
- Fit the model
- Predict test set
- Compute roc_auc_score
- Repeat above step for all model
- Compare roc_auc_score of all models and choose the best model

Logistic Regression

By using LogisticRegression(), I build Logistic Regression model and fitted our train datasets.

```
[35] accuracy_score(pred, ytest)
0.75

[36] #printing the roc_auc_score
roc_auc_score(pred, ytest)
0.8078703703703703
```

Fig. 4 Accuracy score and AUC ROC score of our Logistic Regression model

Support Vector Machine

By using SVC(), I build a Support Vector Classifier model and fitted our train datasets.

```
[38] #predicting on the test data
      pred = SVC.predict(xtest)
      accuracy_score(pred, ytest)

      0.7931034482758621

[39] #printing the confusion matrix
      confusion_matrix(pred, ytest)

      array([[76, 19],
             [ 5, 16]])

      roc_auc_score(pred, ytest)

      0.7809523809523811
```

Fig. 5 Accuracy score and AUC ROC score of our SVM model

Random Forest

By using RandomForestClassifier(), I build a Random Forest model and fitted our train datasets.

```
[43] # printing confusion matrix
      confusion_matrix(pred,ytest)

      array([[77, 26],
             [ 4,  9]])

[44] accuracy_score(pred, ytest)

      0.7413793103448276

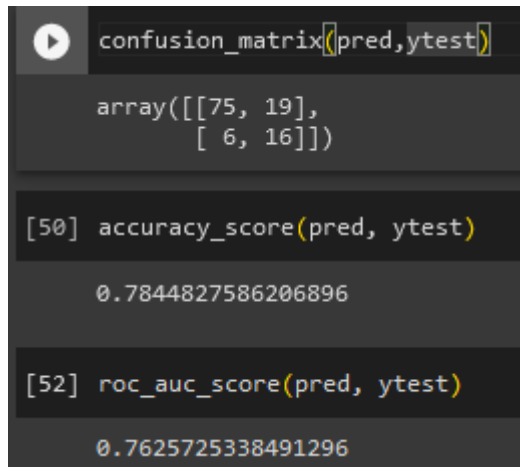
[45] roc_auc_score(pred, ytest)

      0.7199402539208364
```

Fig. 6 Accuracy score and AUC ROC score of our Random Forest model

Decision Tree

By using `DecisionTreeClassifier()`, I build a Decision Tree model and fitted our train datasets.



A screenshot of a Jupyter Notebook cell showing the output of three Python functions. The first line is `confusion_matrix(pred, ytest)`, which returns a 2x2 array: `array([[75, 19], [6, 16]])`. The second line is `accuracy_score(pred, ytest)`, which returns the value `0.7844827586206896`. The third line is `roc_auc_score(pred, ytest)`, which returns the value `0.7625725338491296`.

```
confusion_matrix(pred, ytest)
array([[75, 19],
       [ 6, 16]])

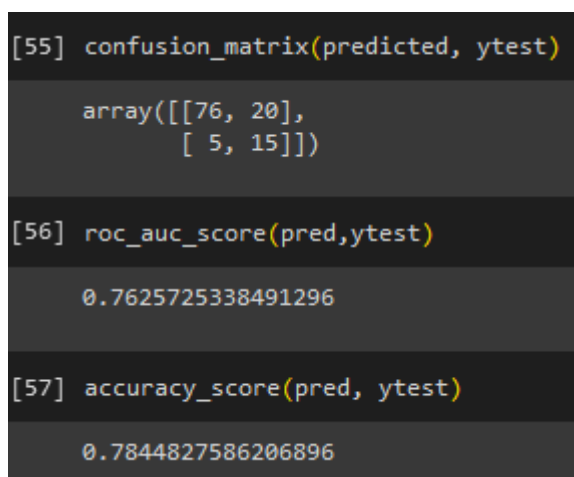
[50] accuracy_score(pred, ytest)
0.7844827586206896

[52] roc_auc_score(pred, ytest)
0.7625725338491296
```

Fig. 7 Accuracy score and AUC ROC score of our Decision Tree model

MLP Classifier (multilayer perceptron classifier)

By using `MLPClassifier()`, I build a Multilayer Perceptron Classifier model and fitted our train datasets. Hidden layer size is 25.



A screenshot of a Jupyter Notebook cell showing the output of three Python functions. The first line is `confusion_matrix(predicted, ytest)`, which returns a 2x2 array: `array([[76, 20], [5, 15]])`. The second line is `roc_auc_score(pred, ytest)`, which returns the value `0.7625725338491296`. The third line is `accuracy_score(pred, ytest)`, which returns the value `0.7844827586206896`.

```
[55] confusion_matrix(predicted, ytest)
array([[76, 20],
       [ 5, 15]])

[56] roc_auc_score(pred, ytest)
0.7625725338491296

[57] accuracy_score(pred, ytest)
0.7844827586206896
```

Fig. 8 Accuracy score and AUC ROC score of our MLP Classifier model

RESULTS AND DISCUSSION

Results: As seen in Figure 5, the Accuracy score of our SVM model is 0.78 and the AUC ROC score is 0.79. When I compared it with other models, this model gave the most accurate result.

CONCLUSION

People who donate blood at short intervals are more likely to donate blood again. After 10-odd months, people's tendency to donate blood again decreases.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my professors Prof Dr Ali Okatan and Assoc Prof Dr Atınc Yılmaz, who contributed to my development in the fields of Artificial Intelligence, Machine Learning, Data Analysis and Bioinformatics that I used in my project.

REFERENCES

<https://www.kaggle.com/datasets/bonastreyair/predicting-blood-analysis?resource=download&select=blood-test.csv>
<https://www.kaggle.com/datasets/bonastreyair/predicting-blood-analysis?resource=download&select=blood-train.csv>