**TECHNOCOLABS MACHINE LEARNING INTERNSHIP**

PROJECT REPORT ON

# Predict Blood Donation for Future Expectancy

## INTRODUCTION

Forecasting blood supply is a serious and recurrent problem for blood collection managers: in January 2019, "Nationwide, the Red Cross saw 27,000 fewer blood donations over the holidays than they see at other times of the year." Machine learning can be used to learn the patterns in the data to help to predict future blood donations and therefore save more lives.

## PURPOSE

The center passes its blood transfusion service bus to one university in Hsin-Chu City to gather blood donated about every three months. The dataset, obtained from the UCI Machine Learning Repository, consists of a random sample of 748 donors.

## Loading the blood donations data

```
import pandas as pd
import numpy as np

df = pd.read_csv("C:/Users/Dell/Desktop/transfusion.data")
df
```

## Inspecting transfusion DataFrame a binary variable representing whether he/she donated blood in March 2007 (1 stands for donating blood; 0 stands for not donating blood)

```
df.shape
```

```
(748, 5)
```

```
df.info()
```

## Creating target column

We are aiming to predict the value in `whether he/she donated blood in March 2007` column. Let's rename this it to `target` so that it's more convenient to work with.

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

df.head()
```

## Checking target incidence

We want to predict whether or not the same donor will give blood the next time the vehicle comes to campus. The model for this is a binary classifier, meaning that there are only 2 possible outcomes:

- `0` - the donor will not give blood
- `1` - the donor will give blood

```
df.target.value_counts(normalize=True)
```

## Splitting transfusion into train and test datasets

Target incidence informed us that in our dataset `0`s appear 76% of the time. We want to keep the same structure in train and test datasets, i.e., both datasets must have 0 target incidence of 76%. This is very

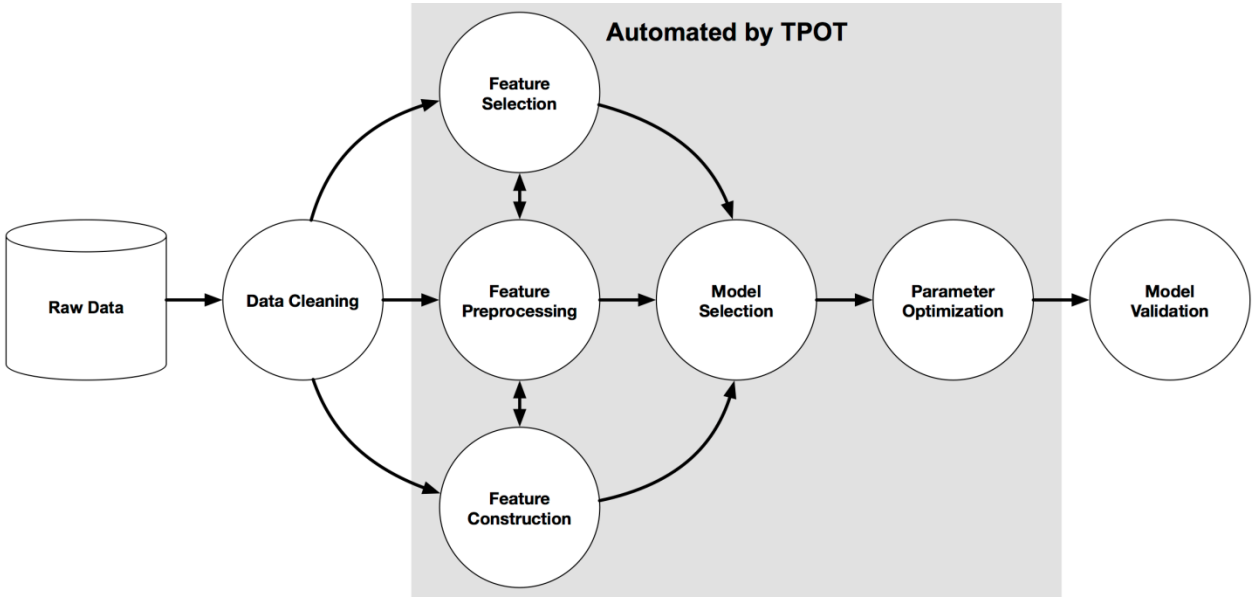
easy to do using the `train_test_split()` method from the `scikit learn` library - all we need to do is specify the `stratify` parameter. In our case, we'll stratify on the `target` column.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test =
train_test_split(df.drop(columns='target'),df.target,test_size=0.25,random
_state=42,stratify=df.target)
X_train.head()
```

## Selecting model using TPOT

TPOT is a Python Automated Machine Learning tool that optimizes machine learning pipelines using genetic programming




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

tpot =
TPOTClassifier(generations=5,population_size=20,verbosity=2,scoring='roc_a
uc',random_state=42,disable_update_check=True,config_dict='TPOT light')
tpot.fit(X_train, y_train)

tpot_auc_score = roc_auc_score(y_test, tpot.predict_proba(X_test)[:, 1])
print(f'\nAUC score: {tpot_auc_score:.4f}')

print('\nBest pipeline steps:', end='\n')
for idx, (name, transform) in enumerate(tpot.fitted_pipeline_.steps,
start=1):

    print(f'{idx}.{transform}')
```

## Checking the variance

Correcting for high variance is called normalization. It is one of the possible transformations you do before training a model. Let's check the variance to see if such transformation is needed.

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

## Log normalization

`Monetary (c.c. blood)`'s variance is very high in comparison to any other column in the dataset. This means that, unless accounted for, this feature may get more weight by the model (i.e., be seen as more important) than any other feature.

**One way to correct for high variance is to use log normalization**

```
X_train_norm, X_test_norm = X_train.copy(), X_test.copy()

col_norm = 'Monetary (c.c. blood)'      ## Specify which column to
normalize

# Log normalization
for df_ in [X_train_norm, X_test_norm]:
    df_['monetary_log'] = np.log(df_[col_norm])
    df_.drop(columns=col_norm, inplace=True)

X_train_norm.var()
```

## Training the linear regression model

```
from sklearn import linear_model

logreg = linear_model.LogisticRegression(
    solver='liblinear',
    random_state=69
)

# Train the model
logreg.fit(X_train_norm, y_train)

# AUC score for tpot model
logreg_auc_score = roc_auc_score(y_test,
logreg.predict_proba(X_test_norm)[:, 1])
print(f'\nAUC score: {logreg_auc_score:.4f}')

print("Tpot AUC Score: ",tpot_auc_score.round(2))
print("Logistic Regression AUC Score: ",logreg_auc_score.round(2))
```

## Conclusion

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.