INTRODUCTION

**About the data set**:

Creating a predictive model is a goal of a national veterans' organization looking to increase the efficiency of their direct marketing campaign. The company is one of the biggest direct mail fundraisers in the US, with a donor database of over 13 million people.

Their most recent mailing records show a response rate of 5.1% overall. The average donation from those who responded (and gave) is $13.00. Each mailing costs $0.68 to produce and send and includes a gift of personalized address labels along with a variety of cards and envelopes.

We use a sample of this dataset and the aforementioned information to create a classification model that can accurately identify donors in order to maximize expected net profit. To ensure that the sample contains an equal number of donors and non-donors, weighted sampling is used, under-representing the non-responders.

For the first part of the analysis, we worked on the variable TARGET\_B which is the binary indicator for response, where 1 = donor, 0 = non-donor.  
First we used the pd.read\_csv () function from pandas to read the csv file and store it into a dataframe. Next, we used pandas "dtypes" to get the different data types and pandas "value counts()" to get the count of each data type to see how many columns are categorical and numerical. Value counts aggregates all unique instances and returns the count for each of them.

To get how many missing values are in each column we use **sum()**along with **isnull().**

We sorted the columns in descending order to see which columns had the most missing values. We used the sort values() function to accomplish this.

**Data analysis**:

We built a decision tree model to predict TARGET\_B.

An effective and well-known algorithm is the decision tree. An example of a supervised learning algorithm is the decision-tree algorithm. It works with output variables that are both continuous and categorical.

Used Python Packages:

1. **sklearn:**

Here, we are using some of its modules like train\_test\_split, DecisionTreeClassifier and accuracy\_score.

1. **NumPy :**

It is used to read data in NumPy arrays and for manipulation purpose.

1. **Pandas :**

Used to read and write different files, Data manipulation can be done easily with dataframes.

Steps involved in building a tree model:

* Prepare the dataset.
* Using the Python sklearn package, divide the dataset into train and test parts.
* train the classifier.

Phase of Operations:

* Make your predictions.
* Determine the precision.

Data Slicing:

We can set aside some data to validate the efficacy of our model by dividing our dataset into training and testing data. To test the efficacy against data that our model hasn't yet seen, we perform this split before we build our model.

In order to avoid overfitting our analysis and to assess the precision of our model, we can also separate our data into training and testing data. The sklearn function train\_test\_split() can be used to accomplish this.

Before training the model, we divided the dataset into two parts: training and testing.

First, we must separate the target variable from the dataset's attributes.

For x and y variables, we selected all the values except for Target\_B and Target D for x, and for y which is the target field, we put Target\_B. and spliced the data into training and validation datasets in 70:30 ratio.

DecisionTreeClassifier(), is the tree model we used in our analysis.

Data is divided into a series of binary decisions by decision trees to function. Your ability to move down the tree is dependent on these choices. You proceed through the choices until you reach a leaf node, where the predicted classification will be returned.

Our Decision Tree Classifier model was developed, and it was then assigned to the variable Tree.

The model was then trained using the.fit() method. We entered our training data in order to accomplish this.

For better or worse, Scikit-Learn takes care of making all the decisions for us. Let's now examine how we can use this recently developed model to make predictions:

We created a new variable called predictions, which receives values from our model Tree's application of the.predict() method.

Based on our X test data, we make predictions.

Using Python's Sklearn to Validate a Decision Tree Classifier Algorithm:

Machine learning models of various kinds rely on various accuracy metrics. Sklearn returned an array of predictions when we used the X test array to make predictions. The true values for these are already known because they are kept in y test.

The accuracy score() sklearn function can be used to return a percentage out of 1 that represents how effective the algorithm is. The accuracy score examines the percentage of correct predictions among all predictions.

By adjusting a few of our model's hyper-parameters, we increased its accuracy. The variables you specify when creating a machine learning model are called hyper-parameters.

Scikit-learn offers a way to automate these tests so that you don't have to try a ton of different combinations. This technique, called GridSearchCV, greatly accelerates the process. Scikit-Learn will take care of the process; all we must do is give it a dictionary of various values to test.

Further we performed cross-validation, the cross-validation procedure is finished by the class. In order to avoid overfitting, the class will cycle through various combinations of training and testing data.

For the second part, we are working on the variable, TARGET\_D which is the donation amount (in dollars).

For this we split the variables into predictors and outcomes just like for the first case, here the target variable is TARGET\_D.

After Splicing the data into test and validation datasets in 70:30 ratio, we performed linear regression().

After performing the linear regression, we used AIC\_score function to find the Accuracy\_score of the regression model.

Challenges faced when executing : ﻿

* plotDecisionTree(Tree, feature\_names = train\_x.columns) - got error, ﻿InvocationException: GraphViz's executables not found.

Installed pydotplus, and imported the function in python code, which solved the issue.

* Did not get expected results for matplotlib plots in Spyder, so while working on Juptyer we used : %matplotlib inline after the import function, and it worked fine.

**Findings and Conclusions:**

1. For the DecisionTreeClassifier() for the Target\_B,

1 = donor, 0 = non-donor

The accuracy of the model is around 50% (49%-52%),

Accuracy on the training data:

Confusion Matrix (Accuracy 1.0000)

Prediction

Actual 0 1

0 1099 0

1. 0 1085

Which means, the accuracy of predictions on the training data is perfect, the prediction of target\_B being a donor and non-donor is perfect.

Accuracy on the validation data:

Confusion Matrix (Accuracy 0.5470)

Prediction

Actual 0 1

0 248 213

1 211 264

Which means, the accuracy of predictions on the Validation data is 54% right, the prediction of target\_B being a donor and non-donor is half and half.

Correct prediction of TARGET\_B being a donor is 264 , and non-donor is 248.

1. Performed gridsearch for more accuracy:

Confusion Matrix (Accuracy 0.5032)

Prediction

Actual 0 1

0 1099 0

1 1085 0

Confusion Matrix (Accuracy 0.4925)

Prediction

Actual 0 1

0 461 0

1 475 0

Initial score: 0.5329708395440136

Initial parameters: {'max\_depth': 10, 'min\_impurity\_decrease': 0.005, 'min\_samples\_split': 20}

Improved score: 0.5361713517939244

Improved parameters: {'max\_depth': 3, 'min\_impurity\_decrease': 0.0009, 'min\_samples\_split': 11}

check model's accuracy:

Confusion Matrix (Accuracy 0.5783)

Prediction

Actual 0 1

0 948 151

1 770 315

Confusion Matrix (Accuracy 0.5395)

Prediction

Actual 0 1

0 389 72

1 359 116

According to the prediction results, the model leans more towards TARGET\_B non-donors,

And the models accuracy is around 54%-57% accurate.

Creating a confusion matrix to measure the model’s performance on the training and validation data sets:

From the results, we can see the best accuracy score from running the DecisionTreeClassifier() model number of times, and it is: Best Accuracy Score: 0.744505

Best Max Leaf Nodes: 98.000000

| 0 | 727 | 372 |

| 1 | 186 | 899 |

Where the correct prediction for no-donor was 727 and the wrong prediction for non-donor was only 186, like wise the correct prediction for donor was 899 and wrong prediction was 372.

Chart, line chart

Description automatically generated

1. **Performing LinearRegrresion() for TARGET\_D :**

**The results are:**

**Text

Description automatically generated with low confidence**

1. Assessing the models performance and finding the AIC\_Score:

Graphical user interface, text, application, email

Description automatically generated

The code block above includes the entirety of the code, where our model returned an accuracy of over 70%!

So the prediction of the amount of donation based on all the other factors is 70% accurate.s

So I think using a linearregression() model on datasets to do predictions is more simple and accurate.