Task 1: Classification and Clustering

Section 1: General

1.1 Problem Statement

The dataset "Blood Transfusion Service Centre Dataset", which was retrieved from UCI Machine Leaning Repository, includes the target variable of whether a donor donated blood in 2017. This is the same dataset we used for plotting in assignment 1. All the data description and summary is the same as before. In this assignment, we are tasked to perform supervised and unsupervised machine learning on said dataset. The "Blood Transfusion Service Centre Dataset" (BTSCD) has 4 features and a target attribute on whether that individual donated blood or not. The target attribute is a binary data with 0 as not donated and 1 as donated.

```
def catogories_label():

    row, col = df_cat.shape

    #create columns
    df_cat['last_donate_group']=''
    df_cat['frequency_lvl']=''
    df_cat['blood_volumn']=''
    df_cat['state_of_first_donate']=''

    # Define Categories
    for i in range(row):
```

Figure 1.1

To solve the problem of not having multiple classes for classification, we create a function categories_label() to generate four new classes according to the four features as shown in figure 1.1. The data are as the table below. After the function is called, a Pandas dataframe

last_donate	0-14: group 1, 15-29: group 2, 30-44: group 3, 45-59: group 4, 60-74: group 5.
Frequency	1-10: level 1, 11-20: level 2, 21-30: level 3, 31-40: level 4, 41-50: level 5
Blood_cc	1-10: level 1, 11-20: level 2, 21-30: level 3, 31-40: level 4, 41-50: level 5, 7501-10000: volumn 4, 10001-12500: volumn 5
First_donate	1-25: normal, 26-50: long, 51-75: rare, 76-100: inactive

is generated and is assigned to df_cat to differentiate the original dataframe. Before performing classifying algorithm or clustering algorithm, we apply scaling to the datasets. There are two types of scaler used throughout this assignment 2, the MinMaxScaler and the StandardScaler. The MinMaxScaler is used to fit all the data including outliers into a range between 0 and 1. StandardScaler scales the dataset into a normal distribution and make the mean as 0 and the

standard deviation of the data as 1 throughout the whole data. Due to needs of each different model, scaling of data is done separately in each model.

Furthermore, the dataset will be splitted using train_test_split() function before training each model because each model have different ratio of train test spliting. The normalisation of data is done after the train split using the training data to fit and transform both the train and test data to prevent contamination on test data on split data.

1. 2 Software Libraries and Tools

```
# Basic Imports
import numpy as np
import pandas as pd
import math

# Plotting
import seaborn as sns
import matplotlib.pyplot as plt
```

Numerical Python, Pandas DataFrame and Math were imported as basic libraries to be used. For plotting libraries, we've used Seaborn and Matplotlib.

```
#sklearn
import itertools

from sklearn import preprocessing, svm, datasets
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import LabelEncoder, StandardScaler, normalize, MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import precision_score, recall_score, accuracy_score
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.cluster import AgglomerativeClustering, KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
import scipy.cluster.hierarchy as shc
```

The above picture shows the tools imported from library Sci-kit Learn.

- Data preprocessing: Label Encoder, Standard Scaler, Normalize and Min Max Scaler.
- Metrics (Model Evaluation): Classification Report, Confusion Matrix, Precision Score, Recall Score, Accuracy Score, silhouette_score.
- For supervised learning: SVC, Linear SVC, KNeighborsClassifier.
- For unsupervised learning: AgglomerativeClustering, KMenas, shc.
- For generating training and testing sets: train_test_split.
- For cross validation: cross_val_score.

Section 2: Classification and Clustering Data Mining Models

2.1 KNN Model

K-nearest-neighbors (hereinafter KNN) is one of the hot-picked supervised learning methods due to the simplicity and easiness in interpreting its result. In this part of the report, we will discuss about the flow, algorithm, and evaluate the validity of the model based on classification result and confusion matrix.

```
X = new_df.iloc[:, :-1].values
y = new_df.iloc[:, 4].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
```

First, columns of all predictor variables are assigned to X and target variable to y before the train test split is performed. The testing set size for this model is set to 30% .

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Standard scaler is used to scale the dataset.

```
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train, y_train)
```

```
y_pred = classifier.predict(X_test)
```

And then, the model is built with function KNeighborsClassifier() with the 5 neighbors is set. Fitting the model to the classifier with the training sets.

```
print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))
```

Lastly, we used classification report and confusion matrix to evaluate the validity of the model. The details of the report are stated below:

```
                   precision    recall  f1-score   support

                0       0.79      0.93      0.85       160
                1       0.61      0.32      0.42        59

         accuracy                           0.76       219
        macro avg       0.70      0.62      0.64       219
     weighted avg       0.74      0.76      0.74       219

[[148  12]
 [ 40  19]]
```

Precision for value 1 indicates number of correct predicted blood donors, in this case, we have 0.61 indicates this model gives moderate recall value, it means that out of 100 cases predicted

as donors, only 61 would donate. Precision for value 0 means that number of correct predicted non-donors, and we have got 0.79. That tells us that out of 100 predicted non-donors, 79 was correctly predicted, hence this negative precision is considered a moderate high value. Regarding the recall, we've got relatively high value for negative cases, 0.93. Which means that out of 100 donors, 93 was correctly predicted, but 7 was actually predicted as non-donors. But the recall for positive cases was low, it's only 0.32. So, based on the confusion matrix, the precision and recall values, we concluded that KNN model is a good model to use to predict any new donors with given data would donate blood in March 2017.

K-Mean Clustering

K-Mean Clustering is one of the famous unsupervised learning methods. It assigns each data point to one of K numbers of groups based on the features data given. It computes the centroids of each cluster and iterates until the optimal centroids of all clusters are found. The purpose of using K-Means Clustering is to find out the specific pattern of the blood donors and non-donors. The flow, algorithm and the result evaluation will be discussed in this part of the report.

```
minmax_scaler = MinMaxScaler()
print(minmax_scaler.fit(new_df))
scaled_df = minmax_scaler.transform(new_df)
```

First, we start off scaling the dataset with Min Max Scaler into scaled_df.

```
inertia_list = []
for k in np.arange(1, 20):
    kmeans = KMeans(n_clusters=k)
    kmeans = kmeans.fit(scaled_df)
    inertia_list.append(kmeans.inertia_)

plt.plot(np.arange(1,20), inertia_list, 'ro-')
plt.xlabel('number of clusters')
plt.ylabel('inertia')
plt.show()

kmeans = KMeans(n_clusters=5)
kmeans.fit(scaled_df)
```

Later on, we wanted to find the optimal number of K with the famous Elbow Method. This part also plotted the graph for us to see which number of clusters is minimum and also gives us the smallest variance.

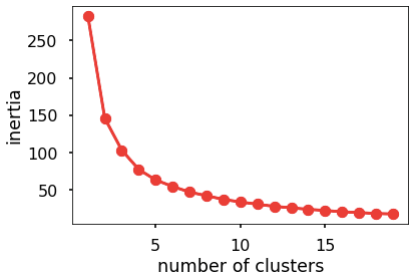


Fig 2.1.0 Inertia vs. Number of Clusters

Based *figure 2.1.0*, we can see that when the number of clusters reached 5, the difference has started becoming smaller as number of K increases. Hence, we choose K = 5 to build the clusters.

```
labels = kmeans.predict(scaled_df)
labels
```

This part of codes assigned the predicted value into a variable named labels and is printed out.

```
Out[15]: array([3, 3, 3, 4, 0, 2, 3, 2, 4, 3, 2, 0, 0, 3, 3, 0, 0, 0, 2, 0, 0, 4,
                2, 4, 3, 0, 4, 2, 3, 3, 3, 2, 0, 3, 0, 3, 0, 3, 4, 0, 4, 3, 3, 2,
                4, 4, 4, 2, 3, 4, 2, 0, 3, 0, 0, 2, 4, 4, 3, 2, 0, 0, 2, 0, 4, 4,
                2, 4, 4, 3, 4, 0, 0, 0, 4, 4, 4, 0, 4, 4, 4, 0, 4, 4, 4, 4, 0, 0,
                4, 4, 4, 2, 4, 2, 3, 0, 0, 3, 3, 2, 4, 4, 2, 2, 4, 4, 2, 0, 2, 4,
                3, 2, 3, 2, 2, 3, 0, 0, 0, 0, 4, 4, 3, 4, 3, 0, 4, 2, 4, 4, 2, 4,
                4, 2, 4, 4, 2, 2, 4, 4, 0, 0, 4, 4, 0, 4, 3, 4, 2, 0, 4, 4, 4, 4,
                4, 4, 4, 4, 4, 0, 0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 2, 0, 0, 0, 4,
                0, 4, 4, 2, 1, 2, 4, 4, 2, 0, 2, 2, 4, 2, 4, 0, 3, 2, 1, 2, 2, 2,
                0, 2, 2, 1, 4, 2, 1, 4, 4, 4, 4, 4, 4, 4, 0, 4, 4, 4, 4, 4, 0, 0, 4,
                0, 0, 4, 4, 4, 4, 4, 0, 4, 2, 1, 2, 4, 4, 4, 2, 3, 2, 0, 3, 4, 4,
                2, 2, 4, 4, 4, 2, 3, 0, 2, 4, 1, 1, 4, 2, 4, 3, 4, 0, 2, 0, 4, 2,
                2, 1, 2, 1, 4, 2, 2, 4, 2, 4, 2, 2, 2, 0, 1, 4, 1, 4, 4, 0, 0, 0,
                0, 2, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 4, 1, 0, 2, 1, 1, 1, 1, 4,
                2, 4, 4, 4, 4, 1, 3, 1, 2, 0, 1, 3, 1, 1, 4, 2, 0, 1, 1, 1, 0, 3,
                1, 0, 1, 2, 0, 4, 1, 2, 4, 4, 4, 4, 4, 4, 4, 4, 1, 1, 1, 1, 4, 1,
                1, 4, 2, 1, 1, 2, 2, 2, 0, 2, 1, 1, 0, 0, 1, 4, 2, 1, 0, 1, 1, 1,
                1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 2, 1, 1, 4, 1, 1, 2, 0, 1, 2, 0, 1,
                1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                1, 1, 1, 1, 2, 1, 1, 1, 1, 3, 1, 1, 2, 1, 1, 2, 2, 1, 2, 1, 2, 2,
```

Fig 2.1.1 Labels

Figure 2.1.1 shows a part of the result. We can see that each data is assigned to cluster named 0, 1, 2, 3, 4, a total of 5 clusters.

Among 4 predictor variables, we have chosen the total volume of blood donated in C.C. as input variables and time since last and first donation and frequency of donated will be the output variable.

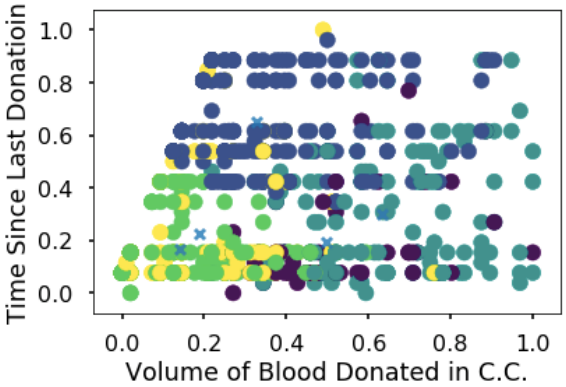


Fig 2.1.2 Time since Last Donation vs. Volume of Blood in C.C.

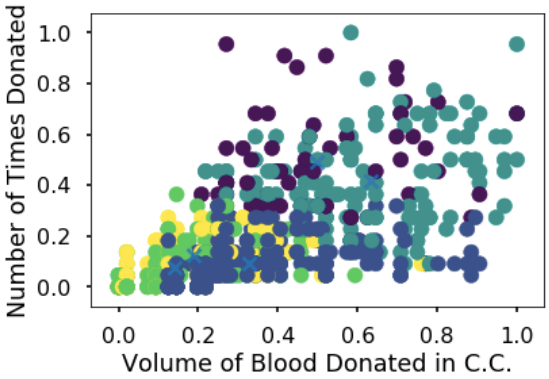


Fig 2.1.3 Number of Times Donated vs. Volume of Blood in C.C.

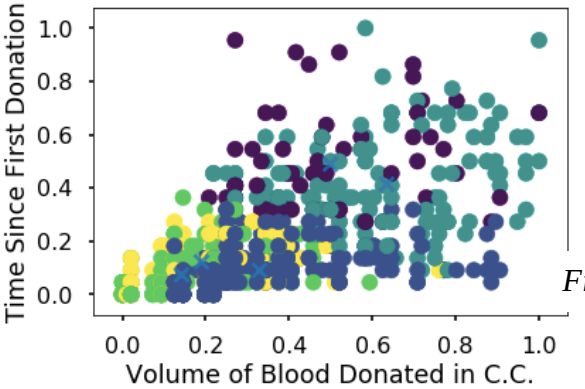


Fig 2.1.4 Time since First Donation vs. Volume of Blood in

Figure 2.1.2 shows us the time since last donation vs. the total volume of blood donated. Based on the cluster, we observed that the purple dark green clusters donors are those who had donated larger volume of blood were donors who recently just did blood donation; whereas yellow and light green clusters donors who donated lesser C.C., seems to donated quite recently so we could infer that they might be new donors. Dark blue cluster were donors who donated lesser volume are donors who has not been donating blood in recent.

Fig 2.1.3 and *fig 2.1.4* has similar pattern of clusters. Based on the purple and dark green clusters, we can conclude that donors who had donated more blood usually had longer history of blood donation (larger value of time since last donation) and had more times in donating blood. The yellow, blue and light green clusters are mostly donors who have shorter donation history and donated lesser times.

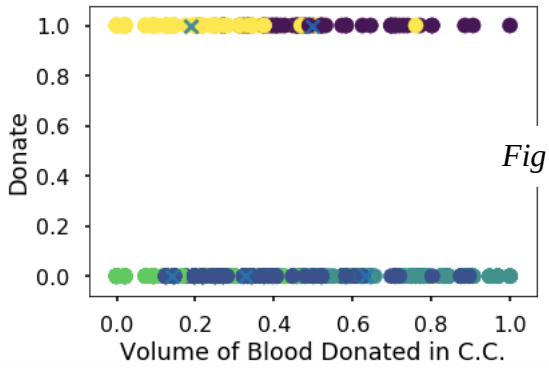


Fig 2.1.5 Donate vs. Volume of Blood in C.C.

Hence, based on the result shown on *fig 2.1.5*, we see that donors who donated blood in March 2017 are donors who are from yellow and purple clusters. Hence, no specific pattern of the donors could be used as standard to predict if a person with certain given data will donate blood or not.

2.2 Logistic Regression

Logistic regression is a classification model despite having regression in its name. The data we used to train the model is a dataset from a hospital on whether an individual donated blood on that particular time based on a few features (refer problem statement). The target data is a binary data with only two possible outputs, donated blood or not donated blood, so logistic regression is suitable for classifying said dataset on paper. This is because logistic regression utilizes sigmoid function which ultimately separates the data into zeros and ones. A self-defined function generate_scores() is used to generate the accuracy, precision, recall, f1, and confusion matrix.

```
def generate_scores(model, x_test, y_test):
    y_pred = model.predict(x_test)
    score_list = {}
    score_list['Accuracy'] = accuracy_score(y_test, y_pred)
    score_list['Precision'] = precision_score(y_test, y_pred, average='macro')
    score_list['Recall'] = recall_score(y_test, y_pred, average='macro')
    score_list['f1'] = f1_score(y_test, y_pred, average='macro')
    score_list['Confusion Matrix'] = confusion_matrix(y_test, y_pred)
    return score_list
```

Figure 2.2.0 Self-defined Function generate_scores()

First and foremost, we have identified a few hyper-parameters to be fine-tuned, namely penalty, C, solver, and max_iter. There are four types of penalty used in smoothing the data, l1 is lasso regression which adds penalty term to the cost function, l2 is ridge regression which adds a penalty term equals to the square of the coefficient, elasticnet is a combination of both lasso and ridge regression, and none is proceed without smoothing. C is the inverse of regularization strength, we set 30 numbers in the range of 10^-4 to 10^4 for the model to run through. There are 5 solvers

in logistic regression used to solve the sigmoid function, lbfgs, newton-cg, liblinear, sag, and saga. Lbfgs stands for Limited-memory Broyden–Fletcher–Goldfarb–Shanno, it approximates the second derivatives matrix updates using gradient evaluations. Newton-cg uses an exact hessian matrix. Liblinear stands for Library for Large Linear Classification which utilizes a coordinate descent algorithm. Sag stands for Stochastic Average Gradient descent which is a variation of gradient descent. Saga is an extension of the sag solver allowing the use of l1 regularization. Max_iter means the max iteration for the model to converge, to find the optimum point where the model does not overfit or underfit.

GridSearchCV is used to perform a search on which combination of the four parameters gives the best yield of the model. we set the parameter cv to 5 to perform a fivefold cross validation on each of the combination. The total number of combination of logistic regression hyper-parameters are 4 * 30 * 5 * 4 = 2400, hence the GridSearchCV is totaling 12000 fits to find the best model. For processing the model quicker, the parameter n_job is set to -1 to use all available processor to process through the the 12000 fits. The GridSearchCV returns the best combination of hyper-parameters as l1 for the penalty, 0.0003562247890262444 for C, max_iter of 5000, and use sag to solve the function.

Default Model

	Type	Score
0	Accuracy	0.78
1	Precision	0.816683
2	Recall	0.574483
3	f1	0.568627
4	Confusion Matrix	[[111, 1], [32, 6]]

Best Model on train data

	Type	Score
0	Accuracy	0.77592
1	Precision	0.695677
2	Recall	0.548706
3	f1	0.536015
4	Confusion Matrix	[[447, 11], [123, 17]]

Best Model on test data

	Type	Score
0	Accuracy	0.773333
1	Precision	0.739286
2	Recall	0.578712
3	f1	0.578373
4	Confusion Matrix	[[109, 3], [31, 7]]

Figure 2.2.1 Scores of Logistic Regression on the Dataset

Figure 2.2.1 shows that logistic regression by default (penalty = l1, C = 1.0, solver = lbfgs, max_iter = 100) gives the accuracy of 0.78 which is higher than the supposing best model with accuracy of 0.773333. However, the best model has a higher f1 score than the default model indicating the model is a more balanced model on with a more balanced precision and recall scores. Hence, the best model will have a more even guess on whether a person is donating blood or not than the default model which leans relatively more towards a person donates blood. Furthermore,

the scoring of the best model on both training and testing datasets are not very desirable suggesting the model is underfitting. This may be caused by a few issues; the most prominent issue is the data having many overlapping patterns than a clear cut division of features between those who donated or not donated blood making the logistic regression have a hard time to separate out the datasets into the target based on the features, hence the model is underfitting.

Ensemble learning

Ensemble learning is a method to utilize the "wisdom of the crowd" to improve the performance of the model, in this case, logistic model. There are 2 types of ensemble learning algorithm in the code file, which are the bagging (BaggingClassifier) and boosting (AdaBoostClassifier). Bagging is an algorithm where multiple model is run in parallel and get the average if using regression or a voting is taken. Then the resulting model with the tuned weight will be the final model. On the other hand, boosting algorithm is a sequential algorithm to train the same model using random data-point from the dataset including those misclassified data to furthermore tune the weights to better corrects the wrongs in the misclassification until eventually getting a model with good performance. Although boosting algorithm tend to give higher accuracy, it is also at a higher risk of overfitting to a model.

```
param_grid = [
    {
    'n_estimators' : (np.arange(15)+1)*2,
    'max_samples' : (np.arange(10)+1)*0.1,
    'max_features' : (np.arange(10)+1)*0.1,
    }
]
```

Figure 2.2.2 Possible Hyper-parameters of BaggingClassifier

Figure 2.2.2 shows the possible hyper-parameters to be tuned for a better model to be trained. I set it to have 2 to 30 number of bags with increment of 2 at a time and the ratio of the max samples and max features ranging from 0.1 to 1.0 with increment of 0.1 each model. The possible combinations of the hyper-parameters are 15 * 10 *10 = 1500 combinations. By using GridSeacrhCV to run a five-folds on to the combinations, totaling to 7500 fittings of BaggingClassifier. As a result, the best model of Bagging Classifier is mas_sample of 0.1,

```
Default Model
                Type                Score
0           Accuracy                 0.78
1          Precision             0.816683
2             Recall             0.574483
3                 f1             0.568627
4  Confusion Matrix  [[111, 1], [32, 6]]
Best Model on train data
                Type                  Score
0           Accuracy               0.772575
1          Precision               0.676942
2             Recall               0.544042
3                 f1                0.52909
4  Confusion Matrix  [[446, 12], [124, 16]]
Best Model on test data
                Type                Score
0           Accuracy             0.786667
1          Precision             0.792857
2             Recall             0.596335
3                 f1             0.603175
4  Confusion Matrix  [[110, 2], [30, 8]]
```

Figure 2.2.3 Scores of Bagging Classifier on the Dataset

max_features of 1.0 and 18 number of estimators (bags). *Figure 2.2.3* shows that the Bagging Classifier have trained the model to be more accurate in the testing data, around 0.67%. Besides, we also see a significant increase in the recall of Bagging Classifier from the default logistic regression suggesting it being more balanced in guessing the positives and negatives.

```
param_grid = [
    {
    'n_estimators' : (np.arange(15)+1)*2,
    'learning_rate' : (np.arange(10)+1)*0.1,
    }
]
```

Figure 2.2.4 Possible Hyper-parameters of AdaBoostClassifier

Figure 2.2.4 shows that there are 30 * 10 = 300 model to be process through by GridSearchCV for five-folds resulting in 1500 total fittings. After processing through the 1500 fittings, the best model is learning rate of 0.1 and 8 number of estimators. *Figure 2.2.5* shows that AdaBoostClassifier increased the accuracy by 0.67% and the precision by around 7% from the default logistic regression. Since AdaBoostClassifier trains the model using random sample and the misclassified samples, it makes the model to be more precise than before hence better fitted the dataset than previously. From the comparison of the model performance on both training and testing datasets, the model is not overfitting the dataset.

2.3 Decision Tree Classification Model

Decision Tree is an algorithm that can be use in both classification and regression problem. It is simple representation of tree structure where internal node denoted as attributes variable and the branch represented as decision rule. The base root produces each of the leaf node which

presents the output value. Apply decision tree classifier to split the node into smallest subset in order to obtain purity node. There are two component of criterion which are Gini index and Entropy information gain. We will compare both accuracy by using Randomized Search CV for hyperparameters tuning in the following section.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

def decision_tree_nc(xtrain1,xtest1,ytrain1,ytest1):

    #decision tree model with 1 iteration
    clf1 = DecisionTreeClassifier(criterion="gini", max_depth=4,
                                  splitter='best',min_samples_split=2,
                                  min_samples_leaf=2,max_features=4,
                                  random_state=70,
                                  min_impurity_decrease=0.0)

    # Train Decision Tree Classifer
    clf1 = clf1.fit(xtrain1,ytrain1)

    #Predict the response for test dataset
    ypred1 = clf1.predict(xtest1)

    # Model Accuracy
    score = accuracy_score(ytest1, ypred1)
    print("Accuracy for Decision Tree Model: %f" %score,'\n')
```

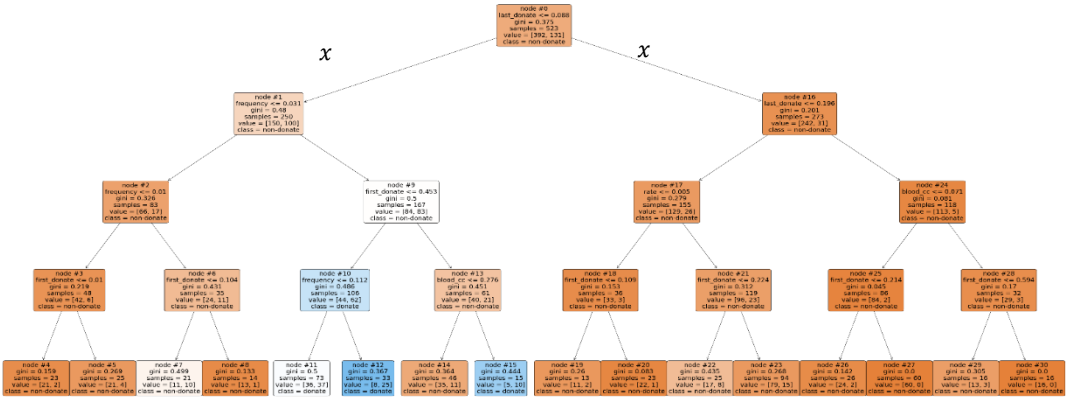


Figure 2.3.0 Decision Tree of Gini Index

Figure 2.3.0 shown the *best split, level 4 maximum depth* to avoid over fitting, *minimum 2 samples leaf, and 70 random state* by using *Gini index* rule for decision tree model. First, we import Decision Tree classifier model from scikit learn library and create a function to return decision tree model and accuracy value. Fitting the model by using the proportional training sample and predicting target variable output to obtain accuracy score. The base root is "last_donate" which has strong correlated relationship with the target variable "donate". The accuracy score was obtained 0.733 from Gini index decision tree.

Cross Validation for Decision Tree

Decision tree is sensitive model to a small change of the data which can cause unstable changes in the model structure. We apply cross validation to avoid over-fitting problem or random split the data structure to improve accuracy of Decision Tree Classifier model. Hence, the concept of cross validation to split the data set into training and validation set corresponding to x and y

with the threshold K-fold parameter. Let *k = 4 folds*, and estimate the accuracy of decision model from cross validation. Example below *Table 2.3.1* shown the process of cross validation.

Table 2.3.1 Cross Validation set.

Training set	Training set	Training set	Validation set
Training set	Training set	Validation set	Training set
Training set	Validation set	Training set	Training set
Validation set	Training set	Training set	Training set

Figure 2.3.2 Decision Tree Accuracy & Cross Validation

```
#cross validation with 4 batches
cross_val = cross_val_score (model,x_dat,y_dat,cv=c_v)
```

```
Accuracy for Decision Tree Model: 0.733333
Cross Validation for Decision Tree:  [0.5989304812834224, 0.7647058823529411, 0.7540106951871658, 0.8288770053475936]
```

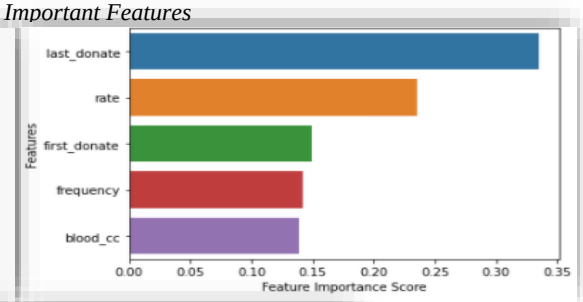
From *Figure 2.3.2* shown that there was 73.33% accuracy score where considered the model able to train and predict the data set correctly. We can observe the accuracy of Decision Tree model are under the range of (60% to 82.89%) of 4-fold cross validation. Therefore, the model is just right for training and fitting by the training data sample size.

Random Forest Classifier

Random Forest is a flexible classifier that easy to use for both classification and regression task on machine learning algorithm. Random forest is also an ensemble learning algorithm. It is usually trained by bagging method to combine multiple the decision trees and take the average score or pruning the child node into a final tree model. It randomly takes replacement on the training data set and splits the node iteratively. Hence, the Random Forest will grow largest. Below is the example of importance feature for Random Forest with 1st iteration of hyperparameter tuning.

Figure 2.3.3 Random Forest & Important Features

```
from sklearn.ensemble import RandomForestClassifier

def random_forest(xtrain1,xtest1,ytrain1,ytest1):

    #Create a random forest Classifier {'gini' or 'entropy'}
    clf2 = RandomForestClassifier(n_estimators=300,criterion='gini',
                                  max_depth=4,
                                  min_samples_split=2,
                                  min_samples_leaf=2,
                                  max_features='auto',
                                  random_state=200,
                                  min_impurity_decrease=0.0)

    #Train the model using the training sets y_pred=clf.predict(X_test)
    clf2 = clf2.fit(xtrain1,ytrain1)

    y_pred1 = clf2.predict(xtest1)

    # Model Accuracy
    score = accuracy_score(ytest1, y_pred1)
    #print("Accuracy for Decision Tree Model: %f" %score)
```




```
    #important feature by calculating percentage
    #and sort values largest to smallest
    feature_imp = pd.Series(clf2.feature_importances_
                            ,index=xx1.columns).sort_values(ascending=False)

    plt.xlabel('Feature Importance Score')
    plt.ylabel('Features')
    plt.title("Important Features of Blood donation Data Sample")

    # Show in bar chart
    sns.barplot(feature_imp, y=feature_imp.index)
    print('\n',feature_imp)
```

```
 last_donate     0.334611
rate             0.235087
first_donate     0.149535
frequency        0.141967
blood_cc         0.138801
dtype: float64
```

Figure 2.3.3 shown the important feature of impurity Random forest with *300 (n estimators)*, *Gini index criterion*, *4 max depth* of splitting child nodes to avoid over fitting, minimum *2 sample leaf*, and *200 random generator number* in the 1st parameter tuning. We fit and train the model with training data set and obtain the predictive result. Apply feature importance to observe the most important attributes that impact the target variable in percentage format. "Last_donate" was the highest impaction to decreasing the weighted of impurity. The accuracy score of Random Forest model was 0.7422 which highest than the Decision Tree 0.7333. This indicates that the Random Forest has slightly improved the accuracy score compare to a single Decision Tree model.

Cross Validation for Random Forest

Figure 2.3.4 Random Forest Accuracy & Cross Validation

```
Accuracy for Decision Tree Model: 0.742222
Cross Validation for Random Forest:  [0.6737967914438503, 0.7593582887700535, 0.7647058823529411, 0.786096256684492]
```

From *Figure 2.3.4* shown the accuracy of Random Forest from 67.61% to 78.69 % with the 4-fold cross validation test. The accuracy of the Random Forest was within the 4-fold cross validation test. Hence, we can say that the model is fitted by the current training and testing data sample.

F1 Score & Confusion Matrix for Decision Tree and Random Forest

Figure 2.3.5 F1 Score for Decision Tree

	precision	recall	f1-score	support
non-donate	0.85	0.80	0.83	178
donate	0.39	0.47	0.42	47
accuracy			0.73	225
macro avg	0.62	0.64	0.62	225
weighted avg	0.75	0.73	0.74	225

Figure 2.3.6 F1 Score for Random Forest

	precision	recall	f1-score	support
non-donate	0.84	0.83	0.84	178
donate	0.39	0.40	0.40	47
accuracy			0.74	225
macro avg	0.61	0.62	0.62	225
weighted avg	0.75	0.74	0.74	225

```
#print classification report (recall, precision, accuracy, f1 score)
print(metrics.classification_report(ytest1,ypred1))
```

From *Figure 2.3.5* and *Figure 2.3.6* shown that the F1 scores of "donate" were 0.42 and 0.40 which were lower than the non-donate classes, respectively. *Figure 2.3.5* shown the positive precision was 39% of the donor would donate their blood in 2017 which considered low precision

with 22 of True positive class. 85% of the donor would donate their blood in 2017 which captured 143 of True negative class. The 47% of positive recall had slightly higher than the positive precision. *Figure 2.3.5*. We wish to achieve low positive recall and high positive precision in the model. Hence, the results shown both of models had low positive precision and high positive recall which were not the desired outcome of the models. Both models captured a lot of the "True negative" class which were 143 and 148 of the donors would not donate in 2017. Overall, the models of accuracy were 50% above which mean both of the models work well in prediction of True Negative class where those donors would not donate in 2017.

However, there is not always true and accurate to obtain best fit of the models with only 1st iteration of hyperparameter tuning. Hence, we apply Randomized Search CV for obtaining best score of the model for decision tree and random forest. Randomized Search CV is useful method when we have many iterations of parameters need to be tuned and it will cause time consuming. We can tune the parameter without exhaustive step to specified all the parameter in each iteration. Now, we execute *10 iterations* with *random or best split*, *entropy or Gini*, *4-fold cross validation* and *different maximum depth* for the decision tree and random forest to obtain difference mean test scores. We *import RandomizedSearchCV* from *scikit-learn* library and apply the training data set to fit the RandomizedSearchCV model on both Decision Tree and Random Forest and return the value in DataFrame format.

Randomized Search CV for Decision Tree

Figure 2.3.7 RandomizedSearchCV for

```
The best parameter score:  {'splitter': 'random', 'max_depth': 6, 'criterion': 'gini'}
The best mean score:  0.745989304812834
```

	param_criterion	param_splitter	param_max_depth	mean_test_score	rank_test_score
0	gini	random	6	0.745989	1
1	entropy	random	4	0.744652	2
2	entropy	random	10	0.671123	9
3	entropy	best	2	0.681818	7
4	gini	random	4	0.721925	3
5	gini	best	6	0.692513	6
6	gini	best	2	0.681818	7
7	gini	random	8	0.720588	4
8	entropy	best	10	0.664439	10
9	entropy	random	8	0.713904	5

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RandomizedSearchCV

#Randomized seacrch cv in the decision tree model
#either gini or entropy, best split or random split with diference depth length
dtc = RandomizedSearchCV(DecisionTreeClassifier(),
                         {
                             'criterion' : ['gini','entropy'],
                             'splitter':['best','random'],
                             'max_depth':[2,4,6,8,10]
                         },

                         cv=4,
                         n_iter =10)
#fit model
rdm_dtc = dtc.fit(xx1,yy1)

#craete dataframe with view column
decision_rdm_cv = pd.DataFrame(rdm_dtc.cv_results_)[[
    'param_criterion','param_splitter','param_max_depth','mean_test_score','rank_test_score'
]]
```

From *Figure 2.3.7*, we can observe that the highest mean score for the Decision Tree was 74.60% or 0.7460 in the second iteration with Gini criterion, best split and maximum depth 6. In comparing, it was improved 1.3 % accuracy of decision tree model in *Figure 2.3.2* with 1st hyperparameter tuning.

Randomized Search CV for Random Forest

Figure 2.3.8 RandomizedSearchCV for Random Forest

```
The best parameter score:  {'n_estimators': 250, 'max_depth': 2}
The best mean score:  0.7633689839572192
```

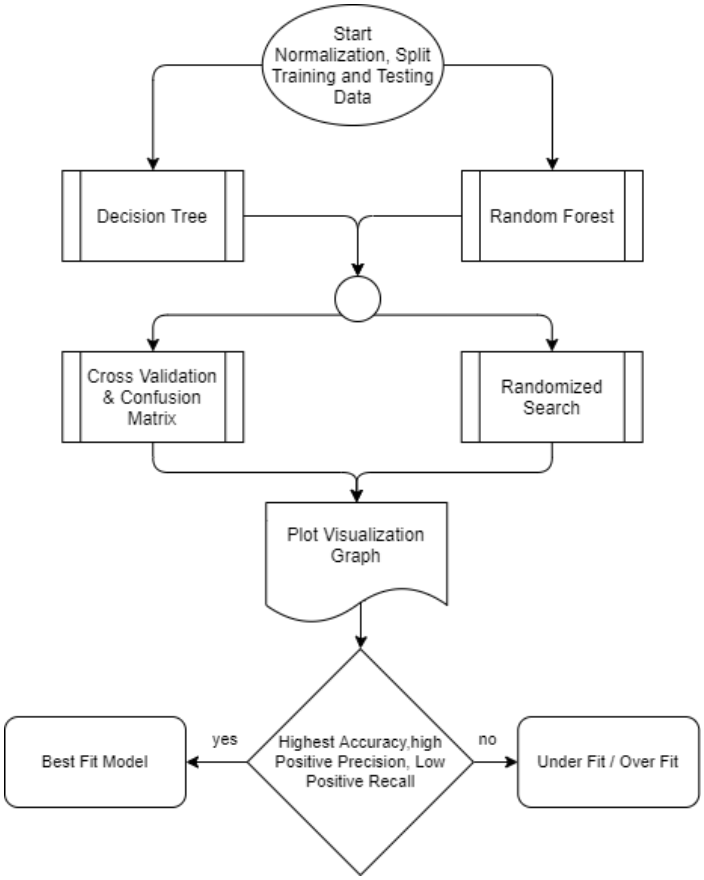
	param_n_estimators	param_max_depth	mean_test_score	rank_test_score
0	250	14	0.696524	10
1	100	10	0.700535	8

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV

#randomized search cv for random forest
# with different number of sub-tree and depth length in the forest
dtc2 = RandomizedSearchCV(RandomForestClassifier(),
                          {
                              'n_estimators':[100,150,200,250,300],
                              'max_depth':[2,4,6,8,10,12,14]
```

From *Figure 2.3.8*, the highest mean score was 76.34% or 0.7634 in the sixth iteration and the parameter score was 250 or 300 of sub-trees and maximum depth 2 in the Random Forest. It was slightly improved by 2.1% of the accuracy in the RandomizedSearchCV model. The largest improvement was Random Forest. Hence, we plot a line graph to observe the mean score for both decision tree and random forest models.

The Line Chart above shown the comparison between Decision Tree and Random Forest with RandomSearchCV. We can observe that the Random Forest had highest mean test score at iteration 6 and 7 with 0.7634. While, Decision tree were 0.6925 and 0.6818 of mean test score at iteration 6 and 7 respectively. Both of the models shown highly fluctuated at each iteration. The Decision Tree and Random Forest Classifier is not suitable to perform prediction on this particular data set. In conclusion, both of the models were not stable for training the donor whether they donate or not donate in 2017.

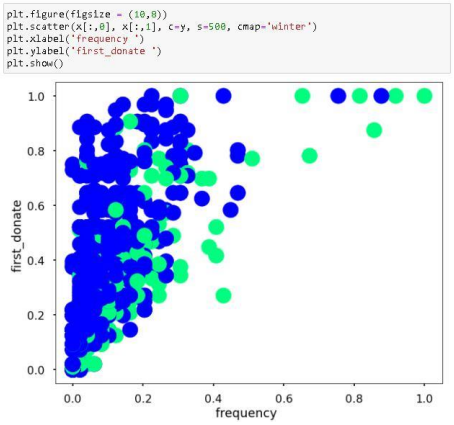
Below is the progress of obtaining the best fit model for Decision Tree, Random Forest with Randomized search method.



2.4.1 Support-vector Machine Classification Model

Support-vector machine(*SVM*) is a supervised learning model with associated learning algorithm for analyzing the data for classification or regression analysis. Since SVM is suitable for two-class visualization, we wanted to deal with features, “first_donate” and “frequency” as it can separate both features either linearly or non-linearly based on our target, “donate”. We look at original scatterplot that shows us the differences between features on their respective target.

figure 2.4.1 Scatterplot



Based on *figure 2.4.1*, the donors with higher frequency in blood donation which is more likely to be donate in 2017, where we have majority of donors who are not donating blood in 2017 falls to low frequency in blood donation.

By building the model of support-vector machine with kernel '*rbf* ', and having the 5-fold cross validation of the model. Also, we want to get the accuracy, precision and recall of the model as shown below.

```
# Building model with default rbf Kernel
clf = SVC(kernel='rbf', C = 100, gamma = .5).fit(x_train, y_train)
y_pred = clf.predict(x_test)
```

```
# Cross validation 5-fold
cross_val_score(clf, x, y, cv=5)
```

```
array([0.77333333, 0.76      , 0.75333333, 0.77181208, 0.76510067])
```

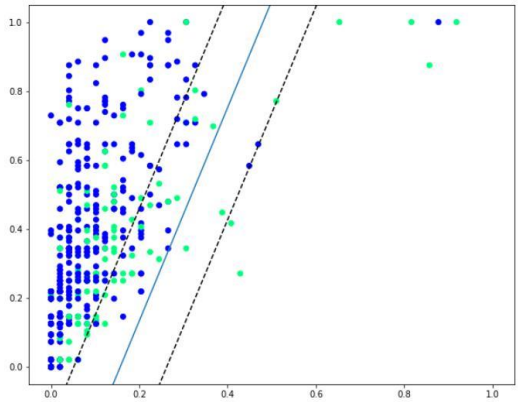
```
Rbf Kernel SVM:
              precision    recall  f1-score   support

           0       0.80      0.99      0.89       178
           1       0.67      0.04      0.08        47

    accuracy                           0.80       225
   macro avg       0.73      0.52      0.48       225
weighted avg       0.77      0.80      0.72       225
```

We are having around 74% of accuracy on this model, where the range falls in between 74% and 76.5% with 5-fold cross validation.

2.4.2 Support-vector Machine Linear Kernel

Before that, we do not use linear kernel for our support-vector machine model, as the illustration is diffcult to separate them into 2 classification. For example, we use *LinearSVC()* to get the possible separate line to split them linearly in *figure2.4.2* below.

figure 2.4.2 Linear Kernel SVM



The overfitting of the model is definitely not suitable for further prediction, as many green dots falls on the left hand side. We can say that it is impossible to classify the data with the use of linear kernel, so we should consider about the higher dimensions for classification instead. For instance, '*rbf* ' model for this situation could be more suitable.

2.4.3 Support-vector Machine RBF Kernel

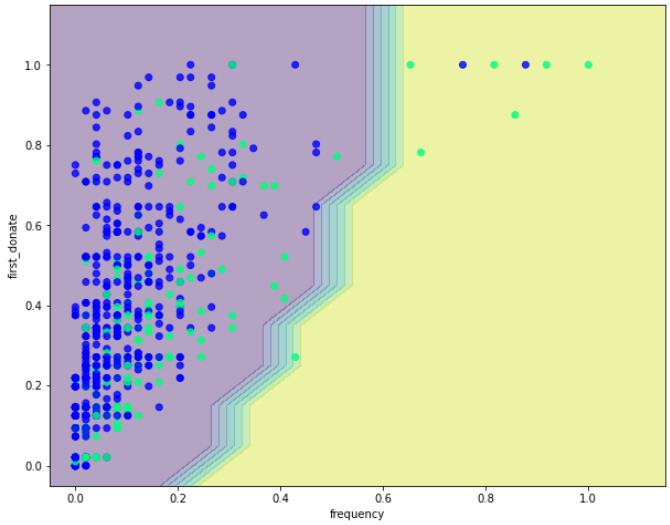
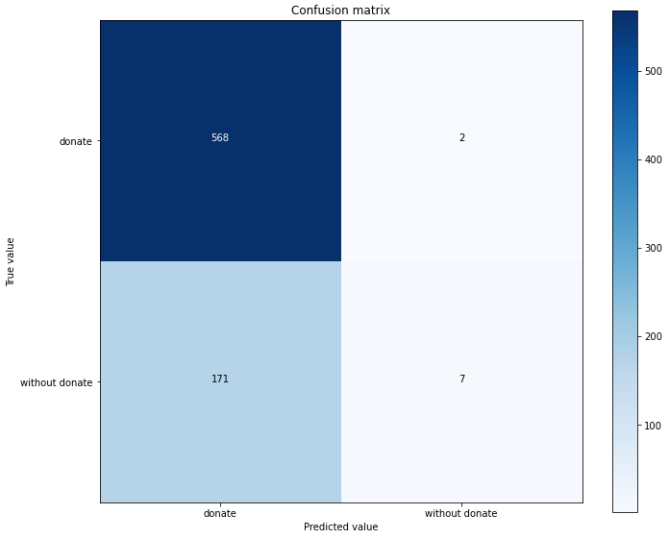


figure 2.4.3 RBF Kernel SVM

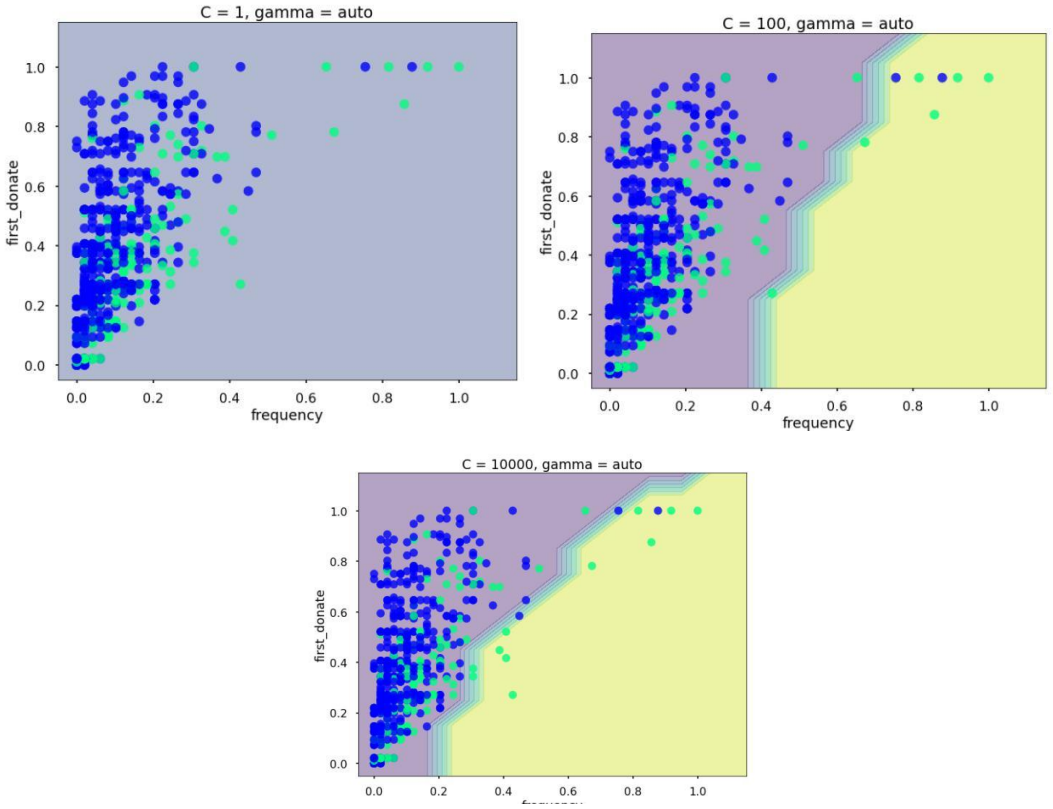
Based on the *figure 2.4.3*, we are using rafial basis function kernal for classification. The classification on the model now is better than previous, as '*rbf* 'kernel is more flexible, but it's still overfitting the data. We can look into the confusion matrix as shown.



As expected, there consists of 168 false negative data in our classification model. It would be not useful for determine for prediction. By far, we are using the '*rbf* ' model with *C* = 100 and

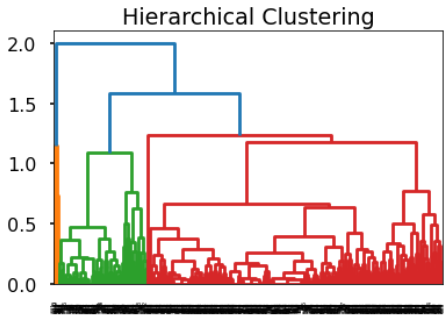
gamma = 0.5. The following illustration will show the '*rbf* ' model with *C* = 1, *C* = 100 and *C* = 10000 and *gamma* set into '*auto*'.

2.4.4 Hyperparameters on C in Support-vector Machine

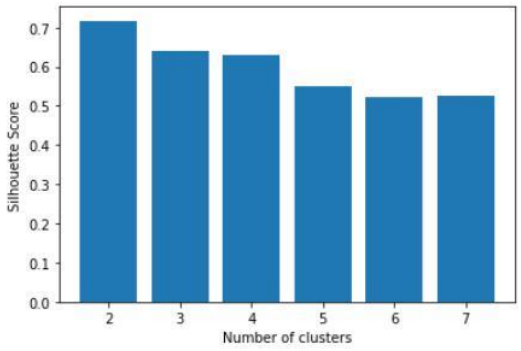


2.4.5 Hierarchical Clustering

Hierarchical clustering is a unsupervised learning model where it is an algorithm to group the similar objects into clusters. Basically, it can be divided into two main types which is agglomerative and divisve hirarchical clustering. One of the best illustration is by using the dendrogram to show hierarchical relationship between our features based on our target here.



We will be finding the most suitable k clusters with 'silhouette_scores' from 'sklearn'.



In the illustration above, we have number of clusters 2 to be the highest score in silhoulette score. Coincidently, our data set has only 2 set of target, thus it might be suitable to clustering our target with this clustering model.

When our target has been allocated into two clusters. For a better visualization, we create the table with 'Cluster_Labels' to demostrate which clusters are the features belong in.

	last_donate	frequency	blood_cc	first_donate	donate	Cluster_Labels
0	0.027027	1.000000	1.000000	1.000000	1	0
1	0.000000	0.244898	0.244898	0.270833	1	1
2	0.013514	0.306122	0.306122	0.343750	1	1
3	0.027027	0.387755	0.387755	0.447917	1	1
4	0.013514	0.469388	0.469388	0.781250	0	1

Once they have been grouped into two different clusters, we can visualize with boxplot for the features against the clustering group. Here we consider about 'Cluster_Labels' against 'frequency' and 'first_donate' feature.

figure 2.4.8 Cluster_Labels against frequency

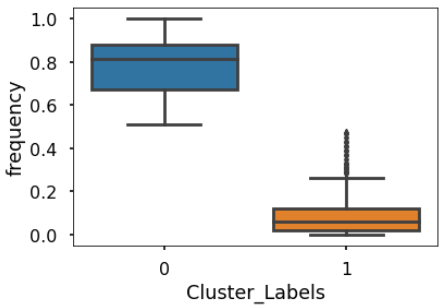
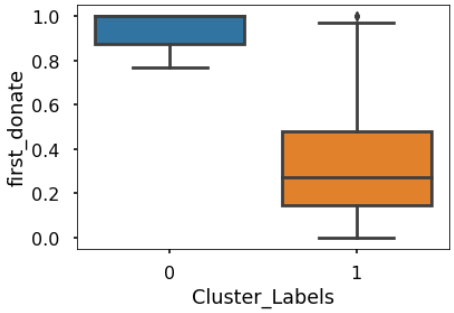


figure 2.4.9 Cluster_Labels against first_donate



In general, we have clustered them with our hierarchical clustering model. From *figure 2.4.8*, 'Cluster_Labels' 0 has the donors with high frequency in donation, while 'Cluster_Labels' 1 has low frequency in donation, as a result we can assume the 'Cluster_Labels' 0 is the cluster of donors donate blood in 2017 whereas 'Cluster_Labels' 1 represents cluster for donor without donate blood in 2017. In the fact that, higher frequency of donating blood tends to have high chance

of next donation. From *figure 2.4.9*, ‘Cluster_Labels’ 0 which is the cluster that will donate blood in 2017 seems to have their first donation far from ‘Cluster_Labels’ 1 which the cluster without donate blood in 2017.