Final Report

Team Name - Data Miners 1

No	Team Members
1.	Shahriar Rahman Dipon(Leader)
2.	Chidambaram Crushev
3.	Erick Oritz

Introduction:

Our dataset contains laboratory values of blood donors and Hepatitis C patients and demographic values like age. Our aim was to classify a patient based on the laboratory results. The target labels contained blood donors vs. Hepatitis C (including its progress “just” Hepatitis C, Fibrosis, Cirrhosis).

We have decided to perform binary classification to distinguish between blood donor and Hepatitis C patients. Also, we have performed multiclass classification to distinguish different stages of Hepatitis.

Link to the google colab notebook for the project -

https://colab.research.google.com/drive/1UY6rKQUxZ-YVP_x8QpmtJ3t53Kxj-o7B?usp=sharing

Data Description:

- The data set is a multivariate data set containing records of laboratory values of blood donors and Hepatitis C patients. The target attribute of the data set has records of Hepatitis C patients with different stages of Hepatitis C.

Statistics about data:

Number of attributes	**14**
Number of instances	**615**

Data columns:

```
df.head()
```

	Unnamed: 0	Category	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CREA	GGT	PROT
0	1	0=Blood Donor	32	m	38.5	52.5	7.7	22.1	7.5	6.93	3.23	106.0	12.1	69.0
1	2	0=Blood Donor	32	m	38.5	70.3	18.0	24.7	3.9	11.17	4.80	74.0	15.6	76.5
2	3	0=Blood Donor	32	m	46.9	74.7	36.2	52.6	6.1	8.84	5.20	86.0	33.2	79.3
3	4	0=Blood Donor	32	m	43.2	52.0	30.6	22.6	18.9	7.33	4.74	80.0	33.8	75.7
4	5	0=Blood Donor	32	m	39.2	74.1	32.6	24.8	9.6	9.15	4.32	76.0	29.9	68.7

X(Patient Id/No)	Type - int64
Category(Target label)	Values - '0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis', '2=Fibrosis', '3=Cirrhosis' Type - Object
Age	Type-int64
Sex(f,m)	Type - string
ALB, ALP,ALT,AST, BIL, CHE, CHOL, CREA, CGT, PROT(*Columns 5-14*)	Type - float64

Data Pre-Processing for Binary Classification:

Step 1(Addressing null values):

```
df.isnull().sum()
```

```
Unnamed: 0      0
Category        0
Age             0
Sex             0
ALB             1
ALP            18
ALT             1
AST             0
BIL             0
CHE             0
CHOL           10
CREA            0
GGT             0
PROT            1
dtype: int64
```

Replacing null values:

- Find the mean in the attribute column for a given target label.
- Replace the mean in the columns of that attribute column corresponding to a given target value.

Step 2(Label encoding for target label):

```
[62] df.loc[(df.Category == '3=Cirrhosis'), 'Category'] = 'Hepatitis'
     df.groupby('Category').nunique()
```

Category	Unnamed: 0	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CREA	GGT	PROT
Hepatitis	75	39	2	27	57	70	71	35	72	66	63	70	62
No-Hepatitis	540	42	2	180	375	293	236	163	364	284	73	302	177

(Before label encoding)

- The target label has multi-labels values - '0=Blood Donor', '0s=suspect Blood Donor', '1=Hepatitis', '2=Fibrosis', '3=Cirrhosis'
- First, the values '0=Blood Donor', '0s=suspect Blood Donor' are replaced with 'No-Hepatitis'. The values '1=Hepatitis', '2=Fibrosis', '3=Cirrhosis' are replaced with 'Hepatitis'.
- Then, label encoding is done to produce the target labels to produce 1(No-Hepatitis) and 0(Hepatitis).

```
df.groupby('Category').nunique()
```

Category	Unnamed: 0	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CREA	GGT	PROT
0	75	39	2	27	57	70	71	35	72	66	63	70	62
1	540	42	2	180	375	293	236	163	364	284	73	302	177

(After Label encoding)

Step 3(Converting non-numerical values for Column Sex(f,m)):

- In column 'Sex', f is assigned as value 0 and m is assigned as value 1. This is done in order to make all the values as numerical and consistent across the dataset.

Step 4(Addressing class imbalance problem):

- The target column ‘Category’ had 540 instances of blood donors and suspected blood donors. There were 75 instances of Hepatitis C patients recorded. The dataset was imbalanced.

- So, we need to address the class imbalance problem and this is done by the Synthetic Minority Oversampling Technique (SMOTE) algorithm. It balances the class distribution by generating minority samples within a distribution.

Before applying SMOTE:

```
[76] unique, frequency = np.unique(Y,
                                   return_counts = True)
     # print unique values array
     print("Unique Values:",
           unique)

     # print frequency array
     print("Frequency Values:",
           frequency)
```

```
Unique Values: [0 1]
Frequency Values: [ 75 540]
```

After applying SMOTE:

```
unique, frequency = np.unique(Y_res,
                              return_counts = True)
# print unique values array
print("Unique Values:",
      unique)

# print frequency array
print("Frequency Values:",
      frequency)
```

```
Unique Values: [0 1]
Frequency Values: [540 540]
```

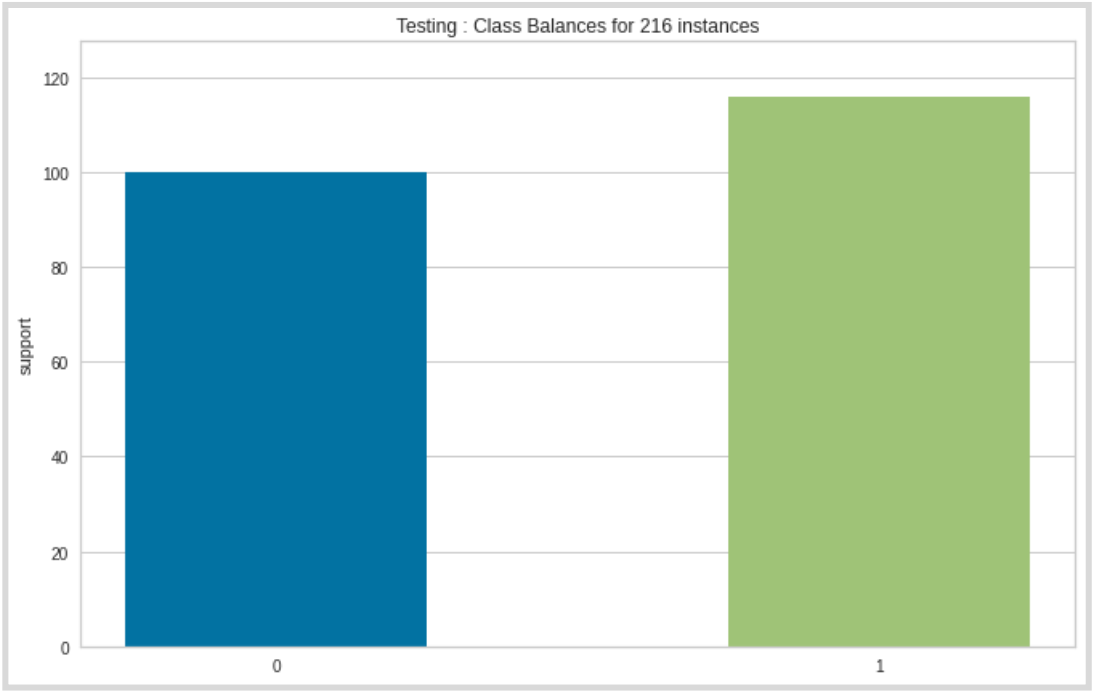
- 

Step 5(Standardize the data):

- The penultimate step is to standardize the data using the StandardScaler() function. This will give us better accuracy when we input the dataset into classifier models.

Step 6(Training and testing data split):

- Now, split the data into training and testing set in the ratio of 4:1 (Training set-80% and testing set-20%)



Reason why we didn't remove any column:

- There are 30 null instances in the whole dataset which had 615 instances which is equal to 5 percent approximately. To replace the null values, we calculated the mean in the attribute column corresponding to the target values. However, for many columns the data distribution was not uniform. For some columns, the mean and the median were similar and for other columns they varied significantly. We have replaced all the missing values with the mean value obtained from that column. We have not dropped any row containing null value because our dataset was imbalanced and there were not many instances for a given target label.

- Also, we converted the non-numerical values in 'Sex' column to numerical values. Then we oversampled the data in order to address the class imbalance problem by using the SMOTE algorithm. Thus, the target labels(1 or 0) have equal instances (540 each).

- Then, we standardized the data. We tried to save our data as much as we can, rather than opting to drop any column. Before inputting it to the classifier, we converted all the datatypes of columns to datatype '**float-64**' to make it consistent and the same.

Data Pre-Processing for Multi-class Classification:

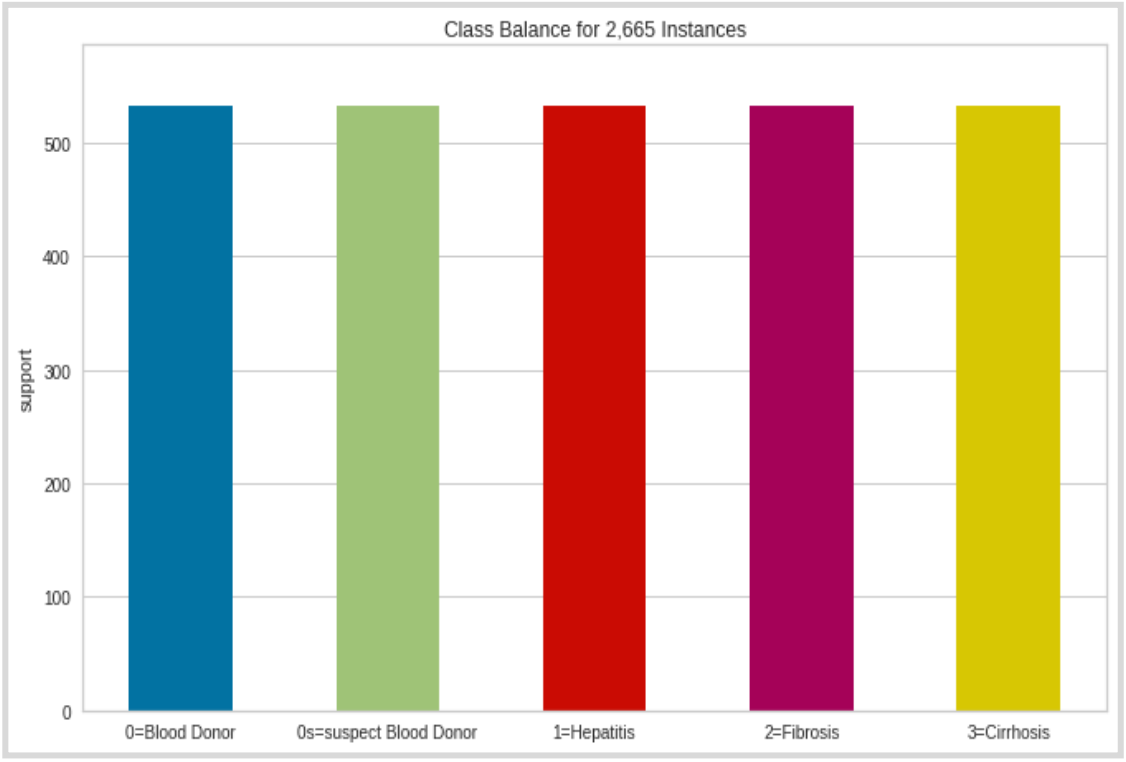
- Step 1 is the same as the binary classification problem by replacing the null values with the mean of the column.
- Since this is a "**multi-class classification**" problem, the labels are not encoded. Category column is the "**target-class**".

```
[376] df_multiclass.groupby('Category').nunique()
```

Category	Unnamed: 0	Age	Sex	ALB	ALP	ALT	AST	BIL	CHE	CHOL	CREA	GGT	PROT
0=Blood Donor	533	41	2	174	370	288	231	162	358	280	69	296	172
0s=suspect Blood Donor	7	7	2	7	7	7	7	7	7	7	7	7	6
1=Hepatitis	24	20	2	12	21	24	23	16	24	23	24	24	22
2=Fibrosis	21	18	2	13	12	20	20	13	21	19	18	20	18
3=Cirrhosis	30	18	2	16	25	29	30	24	29	29	30	28	28

(Target Label - Category)

- The next step would be converting the non-numerical values of Sex column(M, F) to (1,0).
- The next step would be addressing the class imbalance problem using the SMOTE algorithm.



(After applying SMOTE algorihtm)

- Standardize the data and split the data into training and testing data in the ratio 4:1.

Binary Classification

Part of our preprocessing, we have converted our target labels to Hepatitis (=1) and non-Hepatitis (=0). As it is mentioned before that our dataset was unbalanced given the

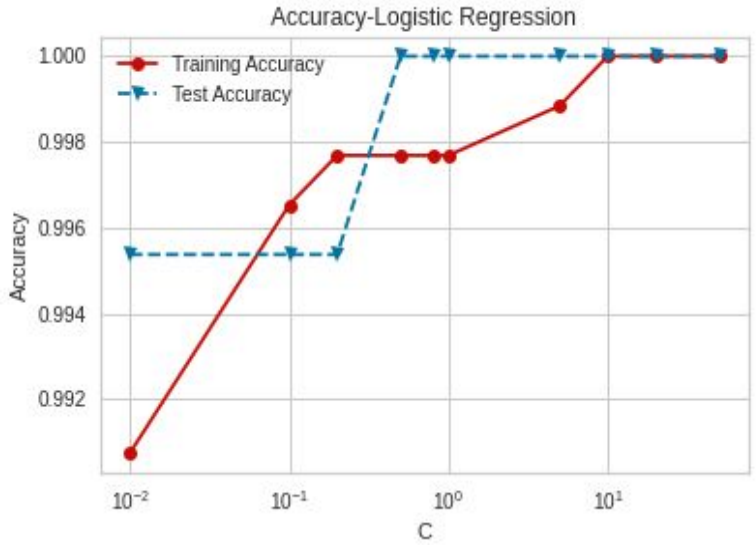
distribution of data frame along with unequal instances of target labels, we have generated target labels for the label that had smaller instances using Synthetic Minority Oversampling Technique (SMOTE). After data generation, we have performed Machine Learning algorithms. The algorithms we have used for Binary Classification are as follows:

i) Logistic Regression
ii) Support Vector Machine
ii) k Nearest Neighbor
iv) Decision Tree
v) Random Forest

In addition, we have used Bagging and Ensemble Learning techniques to improve our model.

Overall, our accuracy score for our models for binary classification were high. Below are the accuracy scores for the aforementioned algorithms after oversampling the dataset :

Logistic Regression : Binary Classification

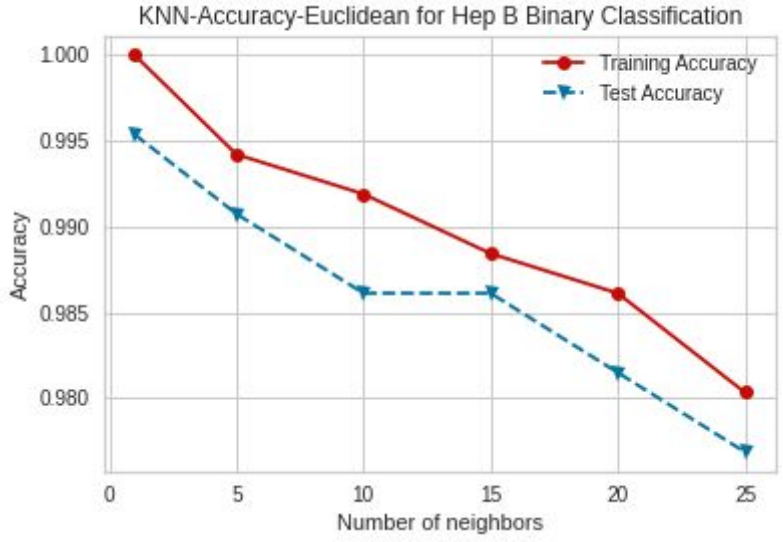


For binary classification using logistic regression, we see in the accuracy/regularization strength diagram for logistic regression that our best value for regularization strength, C, was at 0.2. After 0.2, the model starts to overfit. At our preferred regularization strength, the accuracy value is quite high- above 99%.

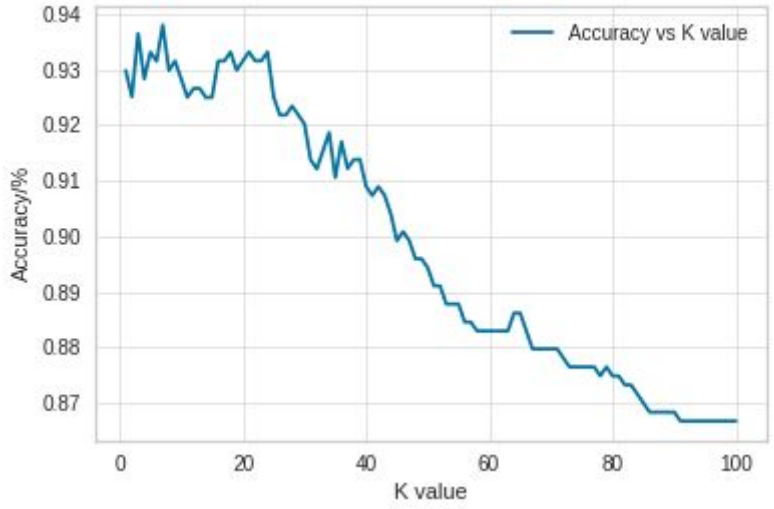
Support Vector Machine: Binary Classification

For Support Vector Machine (SVM), the accuracy of our model was high. However, as the regularization parameter went over 1, the test accuracy starts to decrease. For the test dataset, the model starts fluctuating instead of converging to a minima.

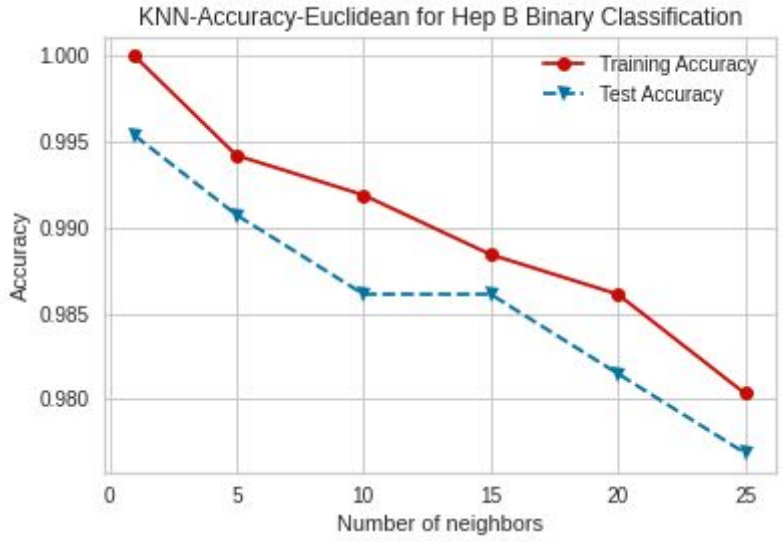
K Nearest Neighbor: Binary Classification



For our oversampled datasets, k Nearest Neighbor (knn) performed as well as other algorithms. However, for our synthetic dataset, the optimum k value stayed low at 2. However, for the original dataset, the best value at 7. The accuracy was high. Below we show the best k value for the original dataset:

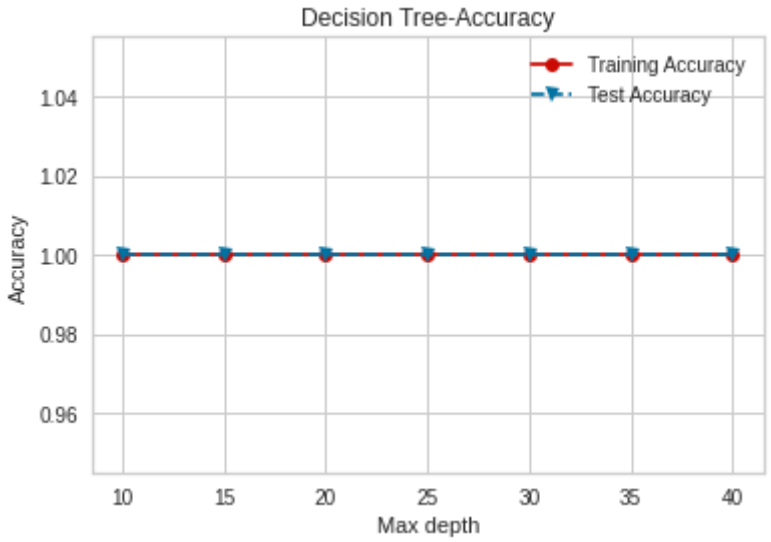


For our oversampled dataset, we have the following outcome:

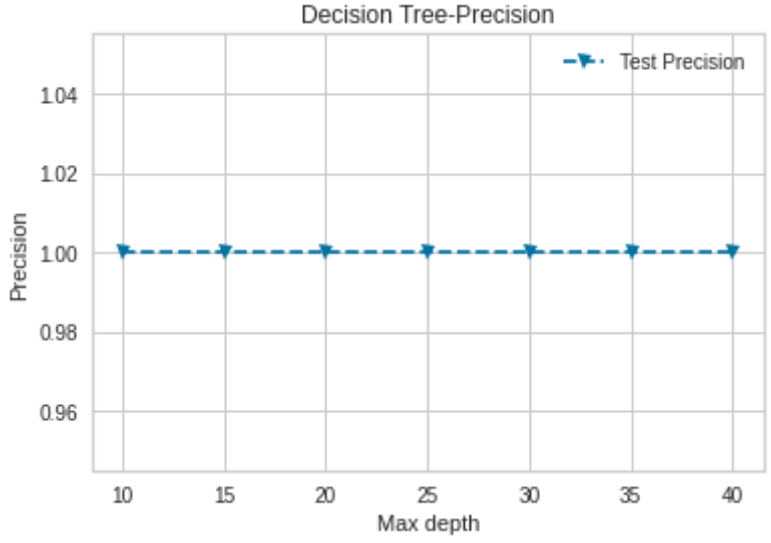


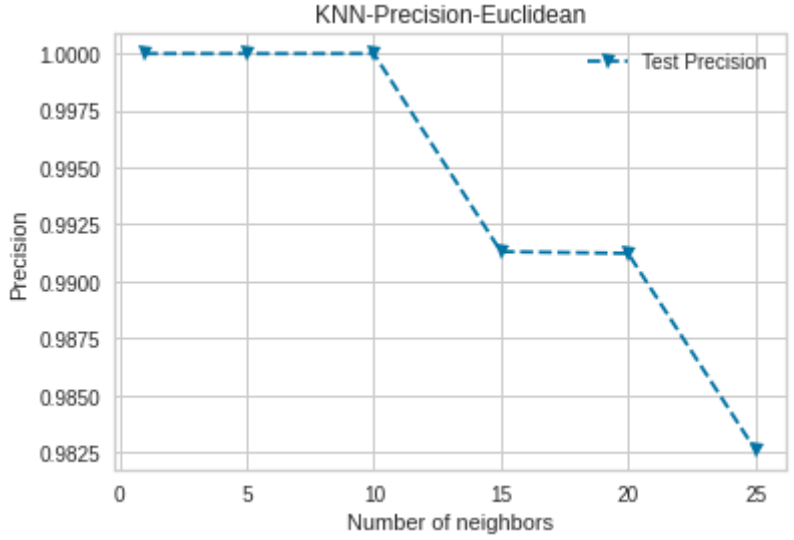
Decision Tree: Binary Classification

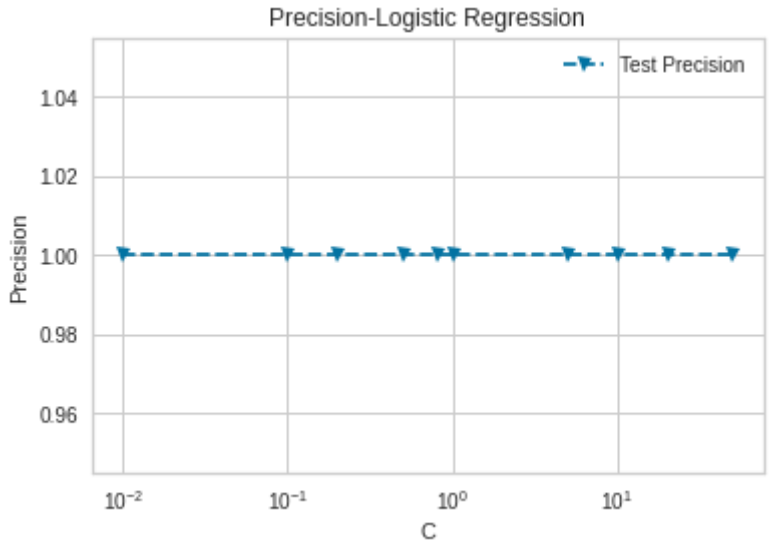
We have used both ‘gini’ and ‘entropy’ for our criterion in using Decision Tree. We have seen that either criteria returns good results for binary classification.

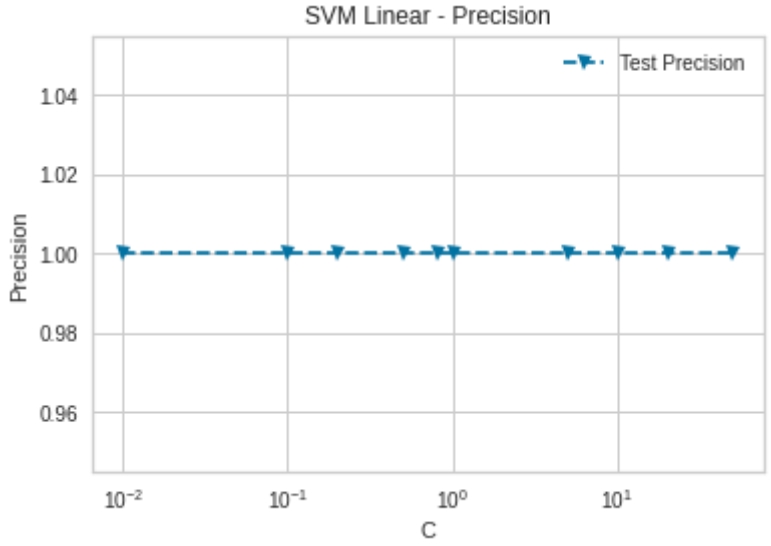


Precision(Binary Classification algorithms):

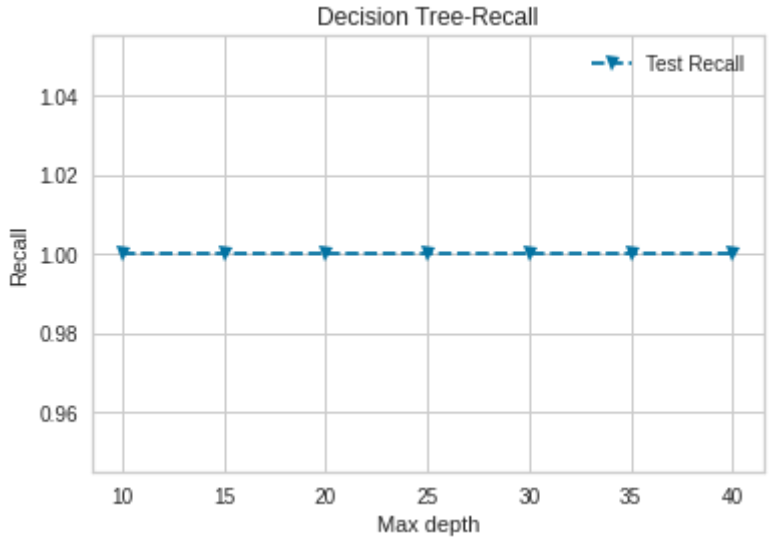


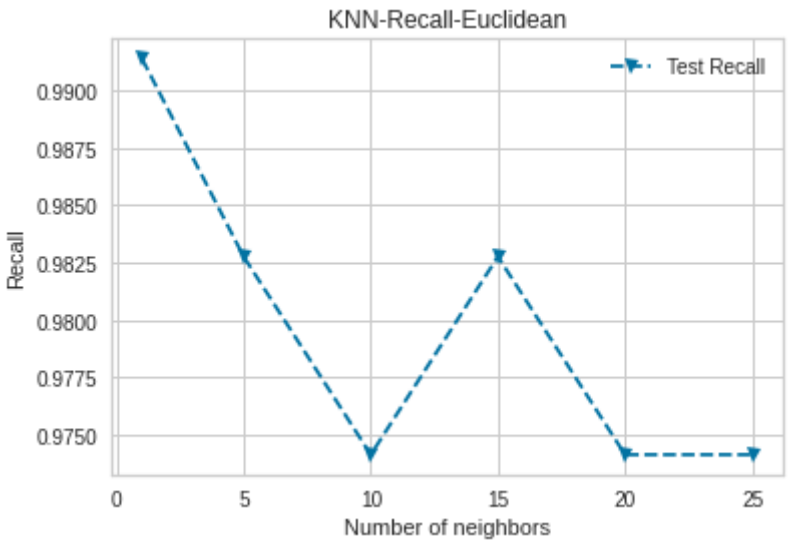


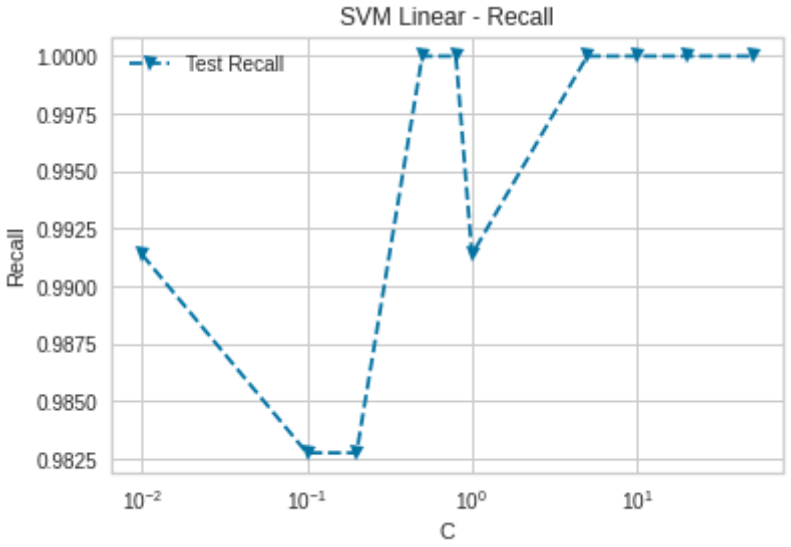


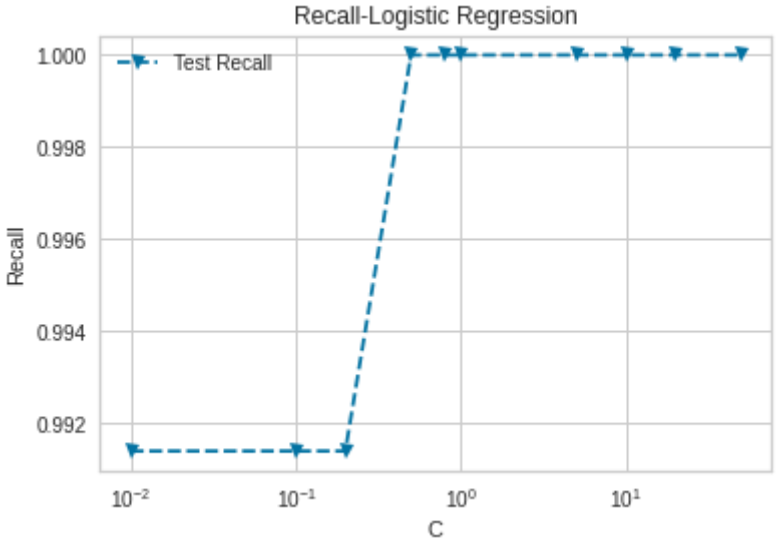


Recall(Binary Classification algorithms):

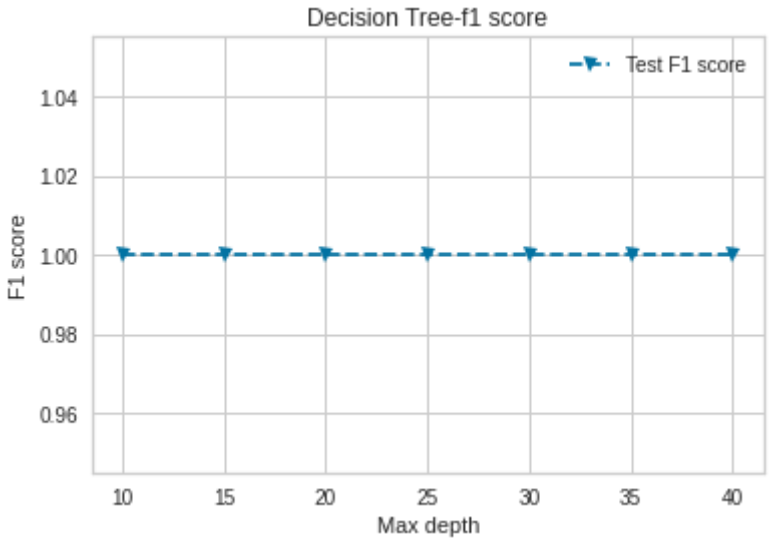


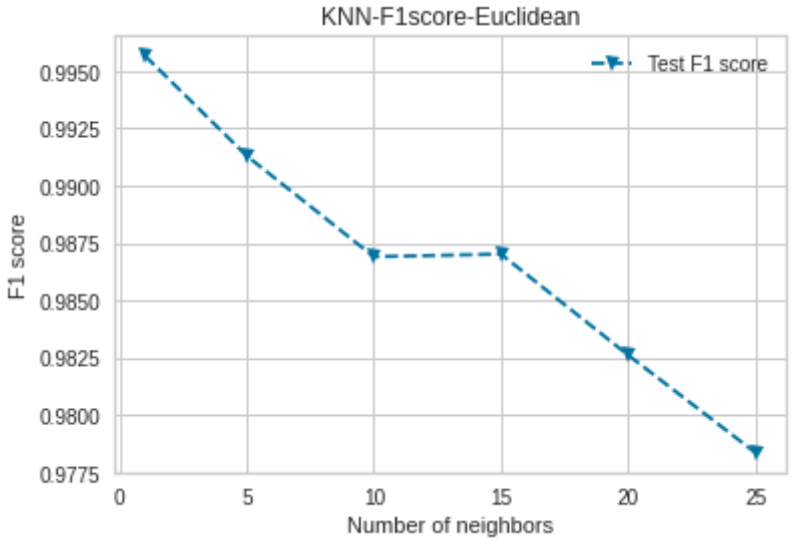

KNN-Recall-Euclidean
Test Recall
Recall
0.9900
0.9875
0.9850
0.9825
0.9800
0.9775
0.9750
0
5
10
15
20
25
Number of neighbors

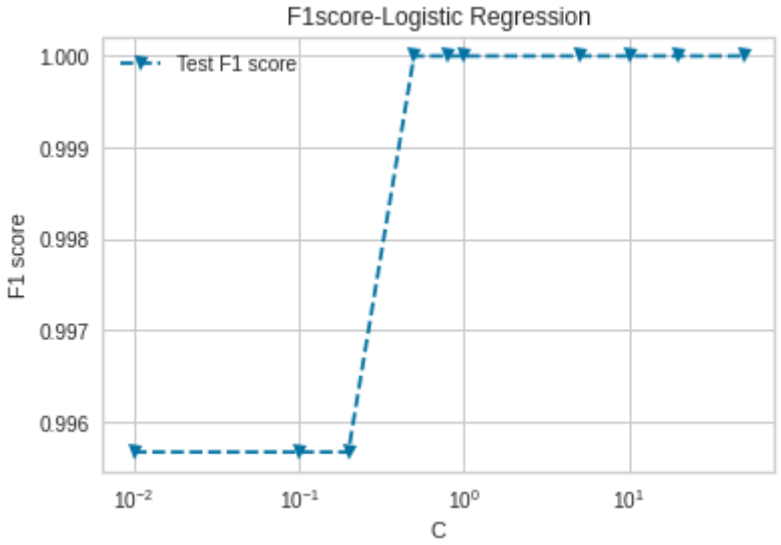

SVM Linear - Recall
Test Recall
Recall
1.0000
0.9975
0.9950
0.9925
0.9900
0.9875
0.9850
0.9825
10^{-2}
10^{-1}
10^{0}
10^{1}
C

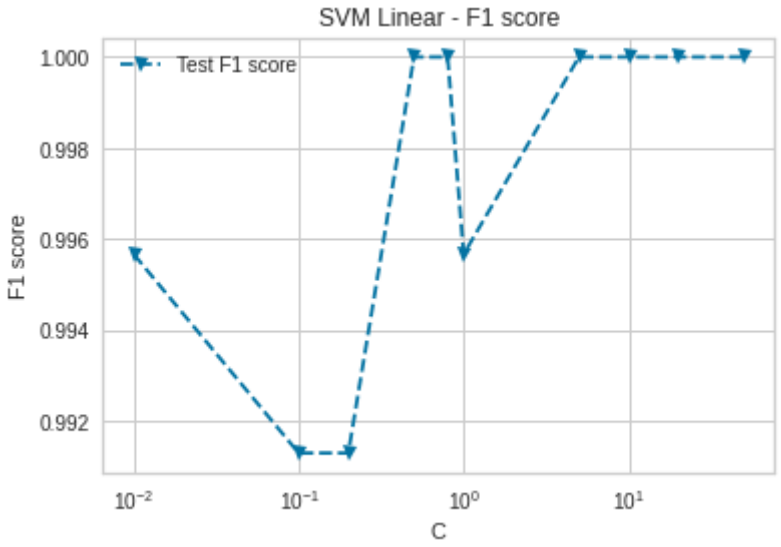

Recall-Logistic Regression
Test Recall
Recall
1.000
0.998
0.996
0.994
0.992
10^{-2}
10^{-1}
10^{0}
10^{1}
C

F1 score(Binary classification algorithms):









From the diagrams above, we see that our models perform well for different algorithms. The F1 score is as high as 1 in most cases.

Bagging:

Bagging, or bootstrap aggregating is a technique that repeatedly samples from a data set. The dataset is of uniform distribution. In our oversampled dataset for binary classification, we have performed bagging with replacement. We have used the Decision Tree algorithm as our only estimator for bagging. Our returned result after running the bagging on test data is 100% and the Out of Bag Score is 99.7 %

Ensemble Method:

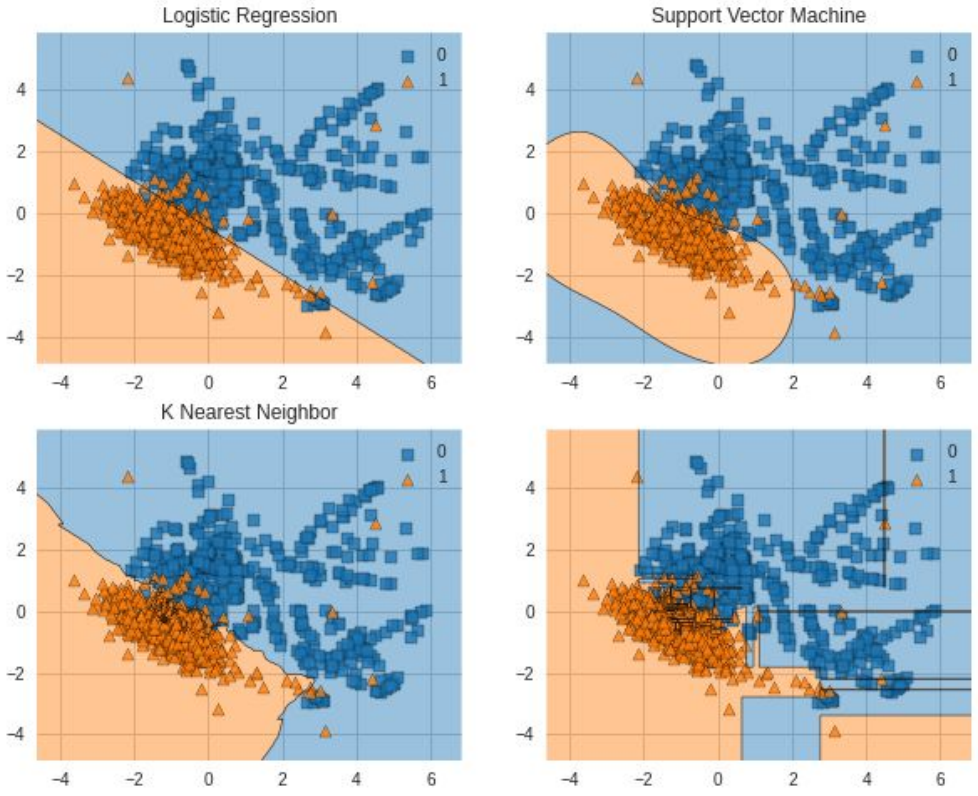
Although the algorithms chosen for our dataset showed high accuracy, we opted to implement the Ensemble Method with four algorithms. Below are the scores:

```
Accuracy: 0.99 (+/- 0.02) [Logistic Regression]
Accuracy: 0.99 (+/- 0.02) [Support Vector Machine]
Accuracy: 0.99 (+/- 0.02) [K Nearest Neighbor]
Accuracy: 0.97 (+/- 0.07) [Decision Tree]
Accuracy: 0.99 (+/- 0.03) [Ensemble]
```

For the original dataset before oversampling, we had the following accuracies using Ensemble Method:

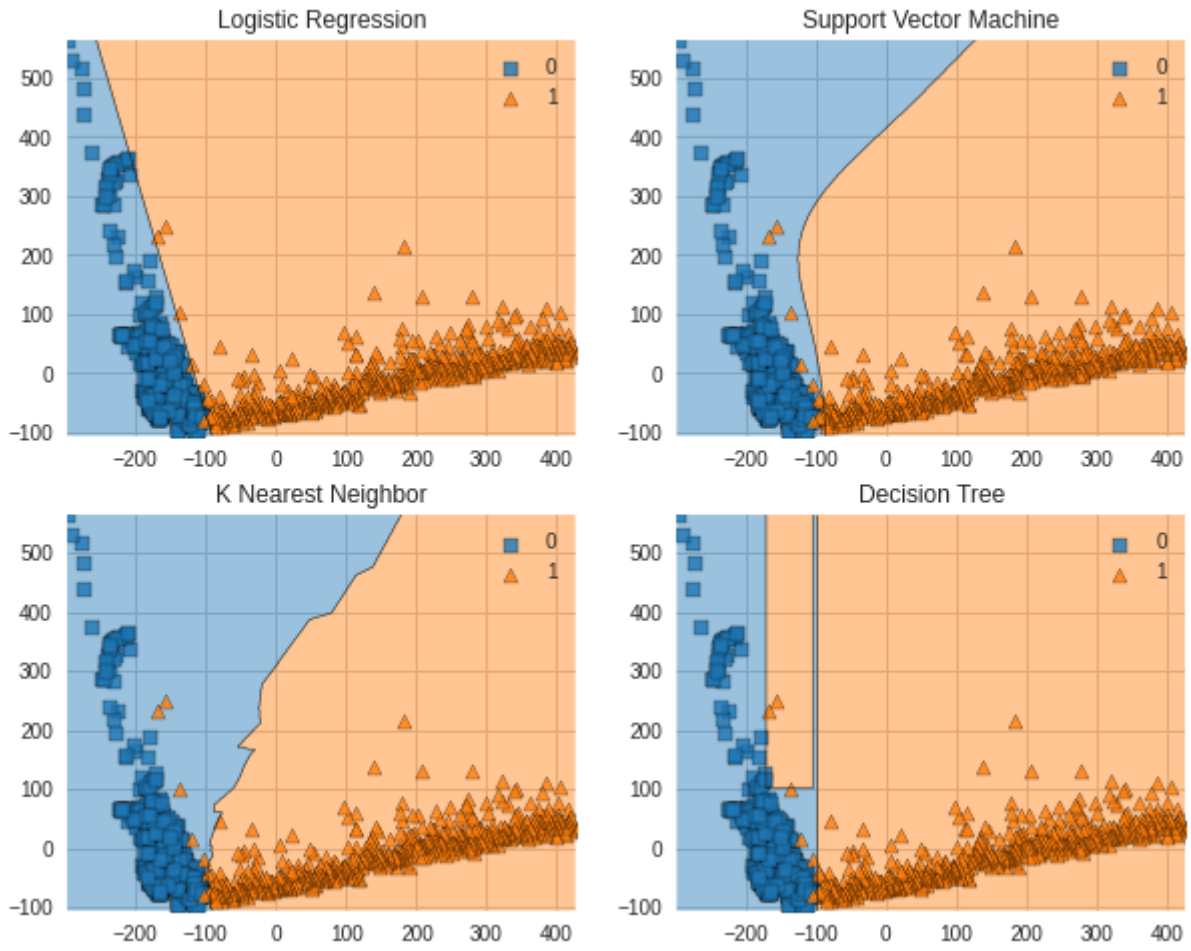
```
Accuracy for original dataset: 0.95 (+/- 0.04) [Logistic Regression]
Accuracy for original dataset: 0.93 (+/- 0.03) [Support Vector Machine]
Accuracy for original dataset: 0.94 (+/- 0.04) [K Nearest Neighbor]
Accuracy for original dataset: 0.95 (+/- 0.03) [Ensemble]
```

The decision boundaries for binary classification with Ensemble Method is shown as follows:



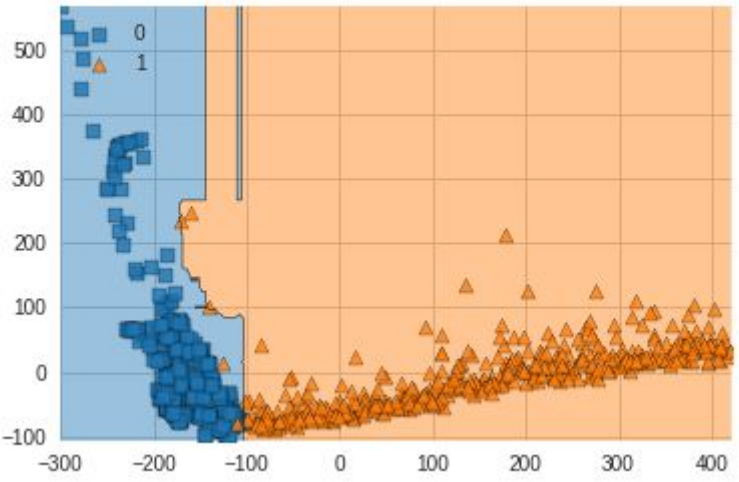
To generate the decision boundaries, we have reduced the attribute dataframe to two-label dataframe using Principal Component Analysis. It should be noted that the decision boundary diagram on bottom right is that for Decision Tree. Please notice datasets were standardized before they were plotted.

Without standardizing the dataset, we get the following decision boundaries:

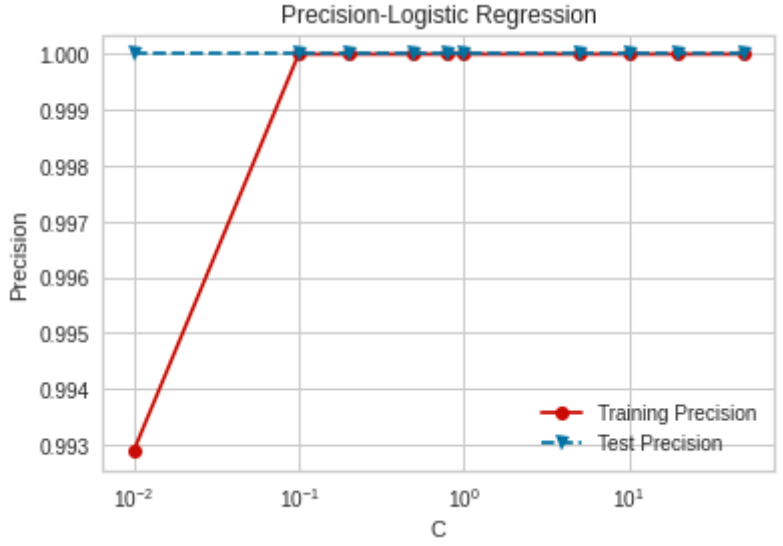


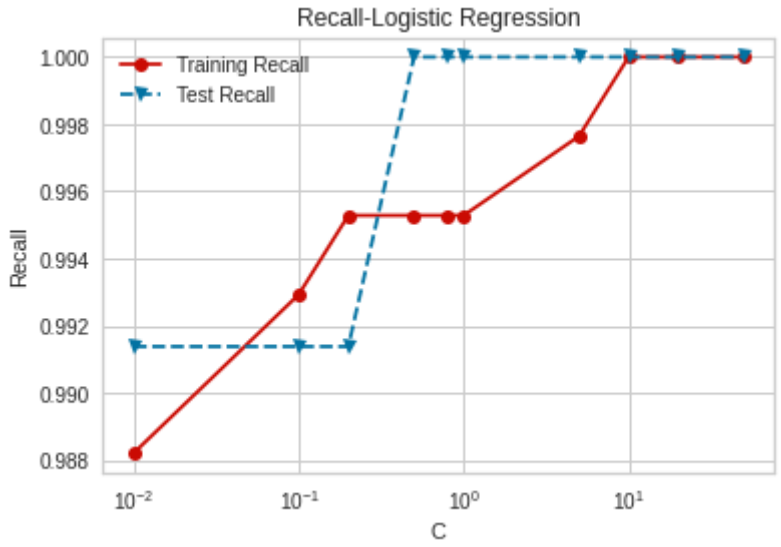
Random Forest:

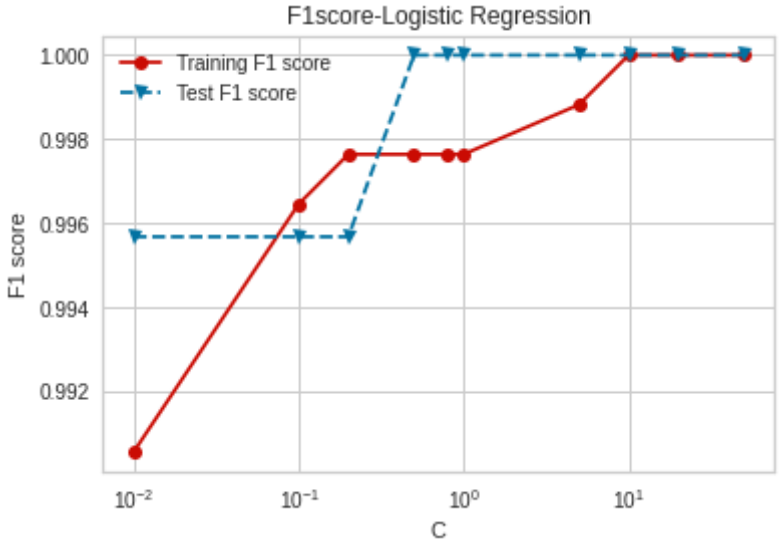
We are aware that decision trees tend to overfit. For binary classification, we sought to find the best length of the decision tree which was at 10-12 . To reduce the trade-off of overfitting in decision trees, we have implemented the Random Forest Classifier. For the Random Forest Classifier, our accuracy was 99.5%. To generate the decision boundaries, we have reduced the attribute dataframe to two-label dataframe using Principal Component Analysis. Below is a diagram for the decision boundary for Random Forest. It should be noted that the values on either axis are standardized values of the attribute:

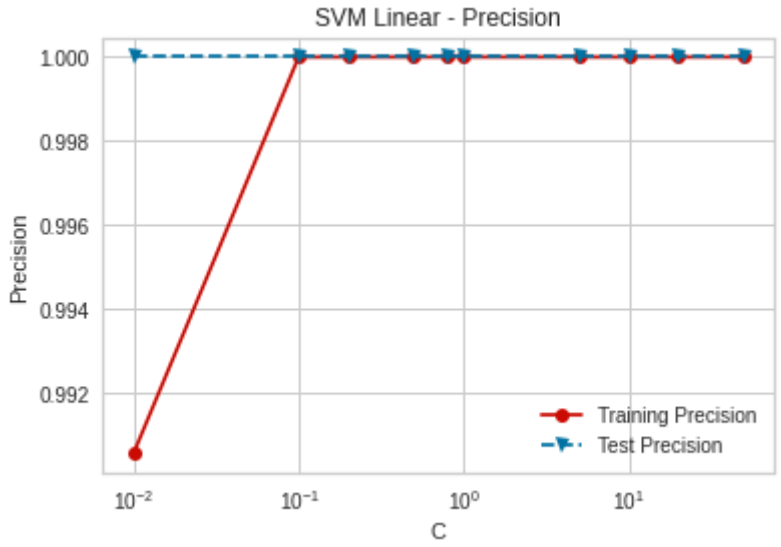


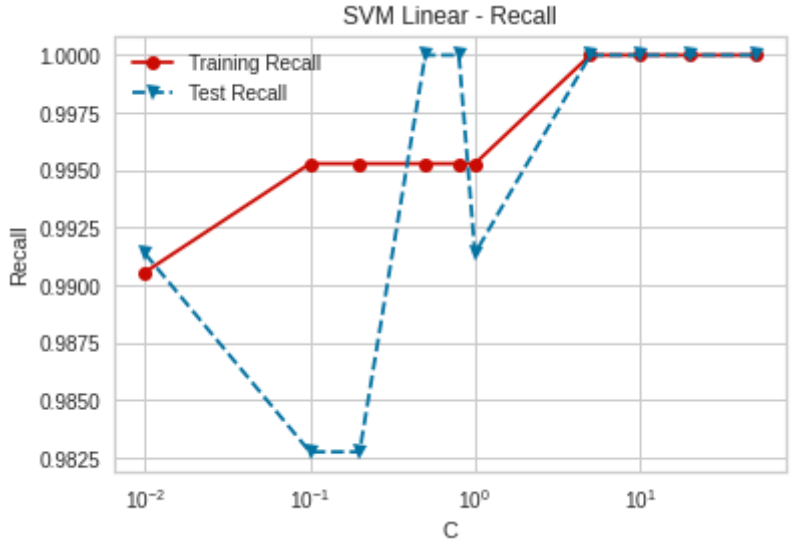
Performance Metric for Binary Classification:

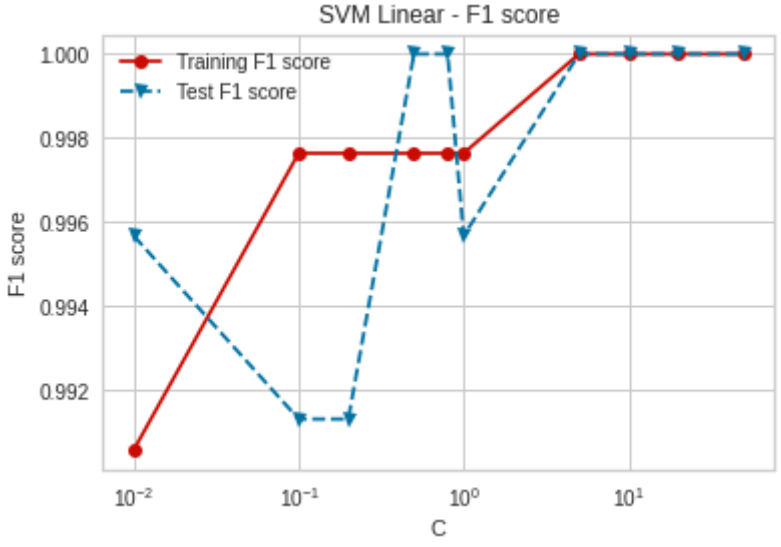


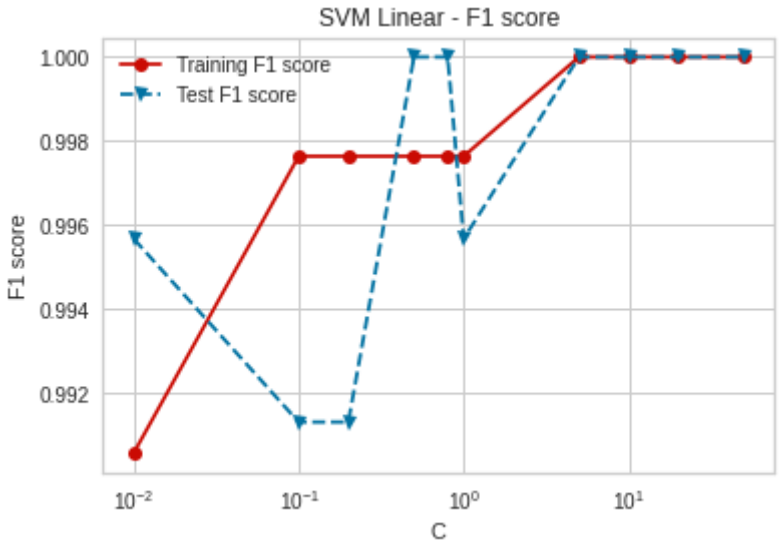

Recall-Logistic Regression
Training Recall
Test Recall
Recall
1.000
0.998
0.996
0.994
0.992
0.990
0.988
10^{-2}
10^{-1}
10^{0}
10^{1}
C

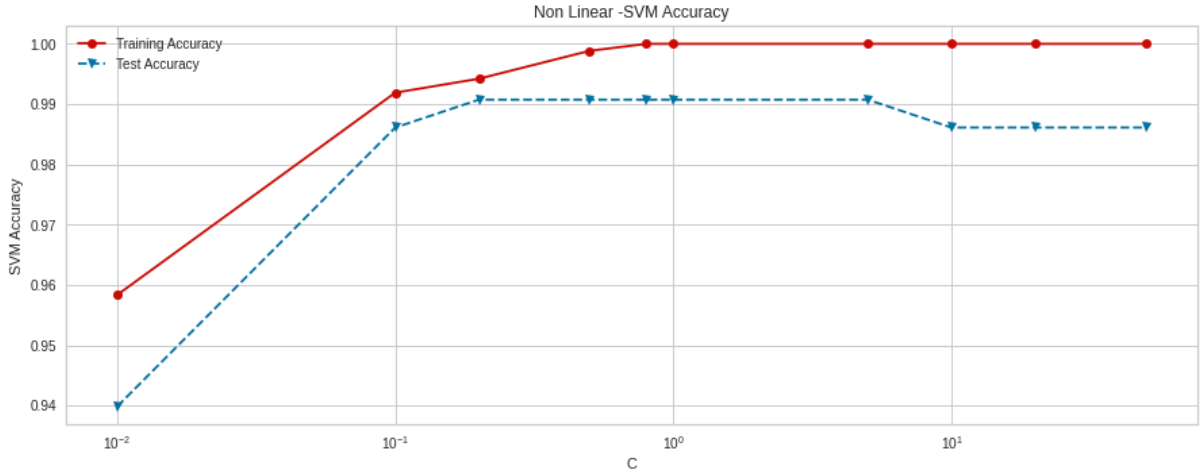

F1score-Logistic Regression
Training F1 score
Test F1 score
F1 score
1.000
0.998
0.996
0.994
0.992
10^{-2}
10^{-1}
10^{0}
10^{1}
C

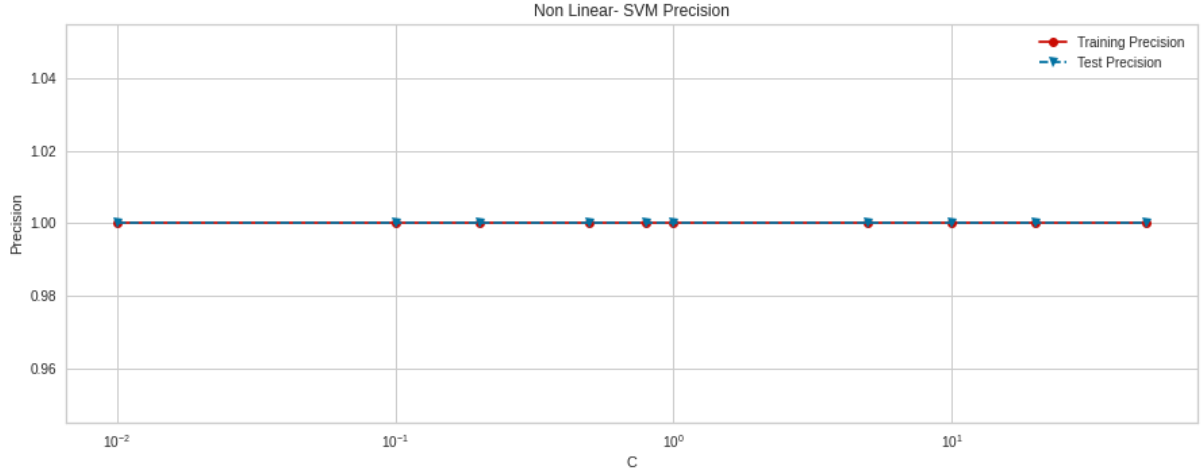

SVM Linear - Precision
Precision
1.000
0.998
0.996
0.994
0.992
Training Precision
Test Precision
10^{-2}
10^{-1}
10^{0}
10^{1}
C

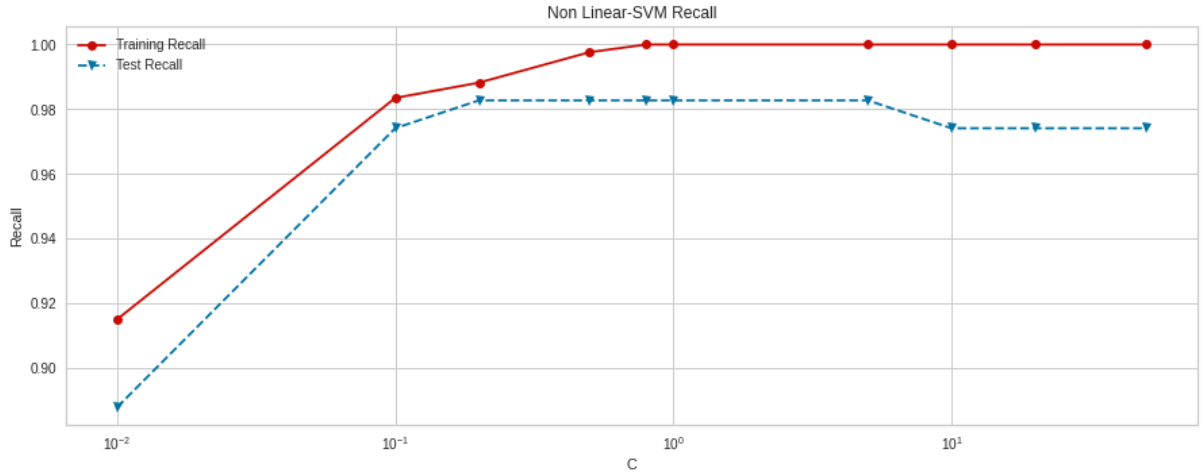

SVM Linear - Recall
Recall
1.0000
0.9975
0.9950
0.9925
0.9900
0.9875
0.9850
0.9825
Training Recall
Test Recall
10^{-2}
10^{-1}
10^{0}
10^{1}
C

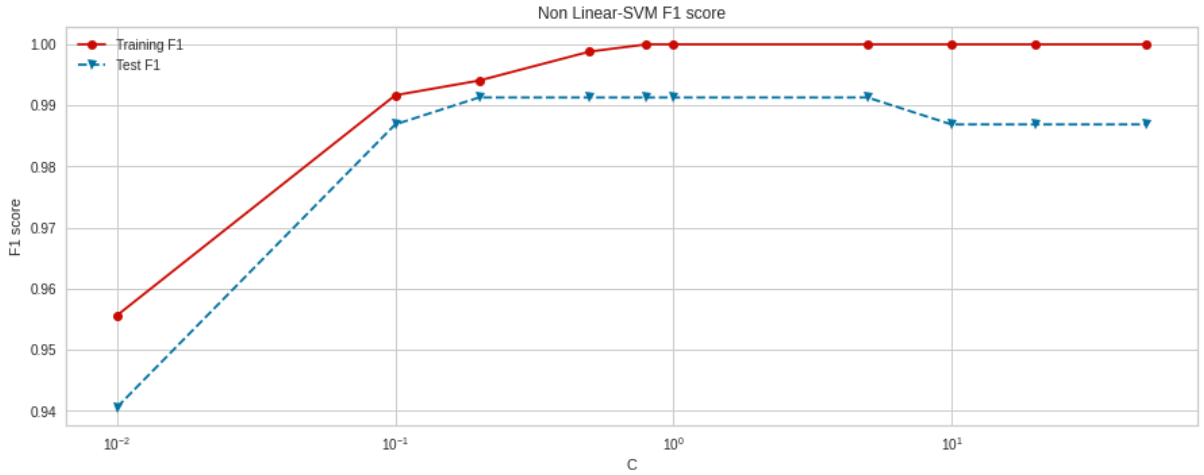

SVM Linear - F1 score
F1 score
1.000
0.998
0.996
0.994
0.992
Training F1 score
Test F1 score
10^{-2}
10^{-1}
10^{0}
10^{1}
C


SVM Linear - F1 score
Training F1 score
Test F1 score
F1 score
1.000
0.998
0.996
0.994
0.992
10^{-2}
10^{-1}
10^{0}
10^{1}
C

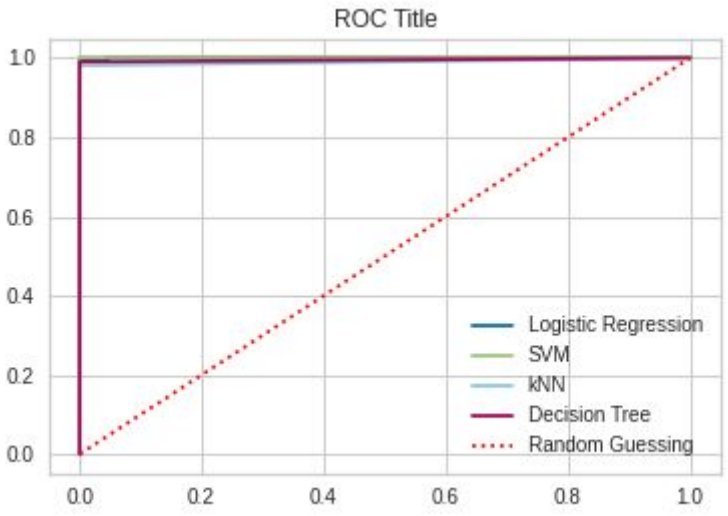

Non Linear -SVM Accuracy
Training Accuracy
Test Accuracy
SVM Accuracy
1.00
0.99
0.98
0.97
0.96
0.95
0.94
10^{-2}
10^{-1}
10^{0}
10^{1}
C


Non Linear- SVM Precision
Training Precision
Test Precision
Precision
1.04
1.02
1.00
0.98
0.96
10^{-2}
10^{-1}
10^{0}
10^{1}
C


Non Linear-SVM Recall
Training Recall
Test Recall
Recall
1.00
0.98
0.96
0.94
0.92
0.90
10^{-2}
10^{-1}
10^{0}
10^{1}
C


Non Linear-SVM F1 score
Training F1
Test F1
F1 score
1.00
0.99
0.98
0.97
0.96
0.95
0.94
10^{-2}
10^{-1}
10^{0}
10^{1}
C

ROC Curve for Binary Classification:



We see from the ROC Curve that our models perform better than random sampling.

Findings for Binary Classification:

The dataset has an uneven number of target labels. The generated data instances we all within the dimension decided by few cases of target instances for some class labels. Therefore, even after generating data synthetically, we see that our target labels are well separated.

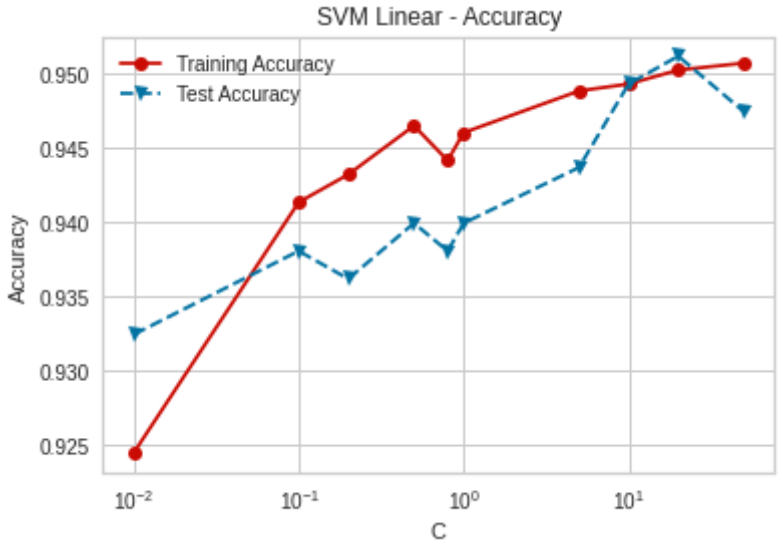
Multi-class classification:

The accuracy scores of the algorithms are:

Logistic regression- Multi-class classification:

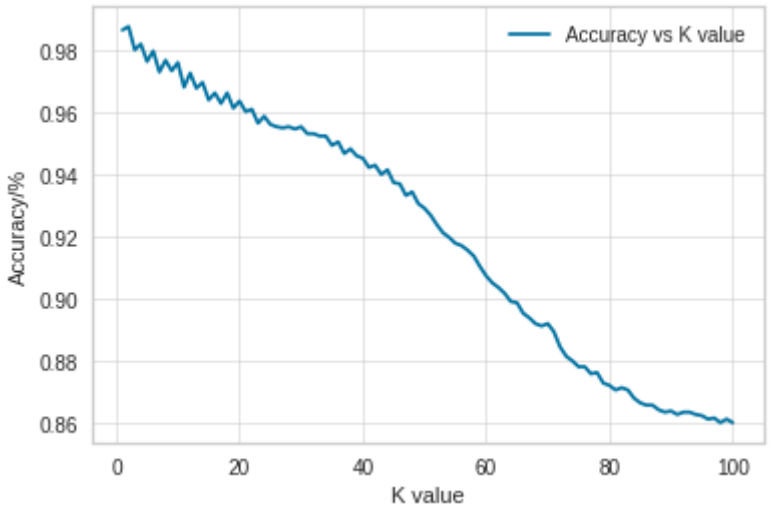
For multi-class classification using logistic regression, the best value for regularization strength, C was 50. At C = 50, the accuracy value is higher - around 92%.

Support Vector Machine - Multi-Class classification:



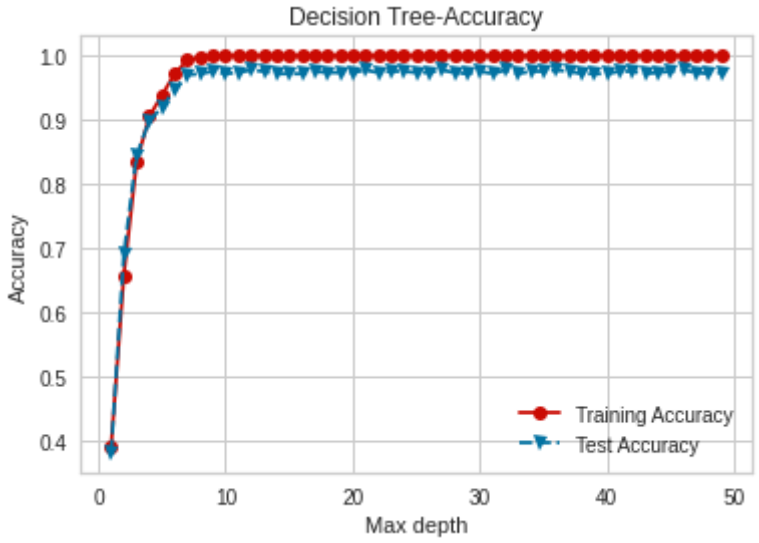
For SVM models, the accuracy is high. However, as the regularization parameter went over 10, the model started to overfit. So, the optimum regularization strength value is 10 because after C=10 the model starts to overfit.

KNN - Multi-Class classification



The diagram shows the best K value versus test accuracy for our oversampled dataset.The optimum k value stayed low at 2. The accuracy was high and found to be 0.987609(K=2) for the oversampled dataset .

Decision tree - Multi-class classification:



The above diagram shows the training and testing accuracy for the decision tree model. From the diagram, we could see that the testing accuracy(0.97749) is high for maxDepth-value 13 and then, it remained consistent for maxDepth values >13. So, the optimum maxDepth value is 13.

Bagging:

In our oversampled dataset for multi-class classification, we have performed bagging with replacement. We have used the Decision Tree algorithm as our only estimator for bagging.

Our returned result after running the bagging on test data is 98.3% and the Out of Bag Score is 96.7 %. This is a visible drop from the results we had for binary classification.

Ensemble Method:

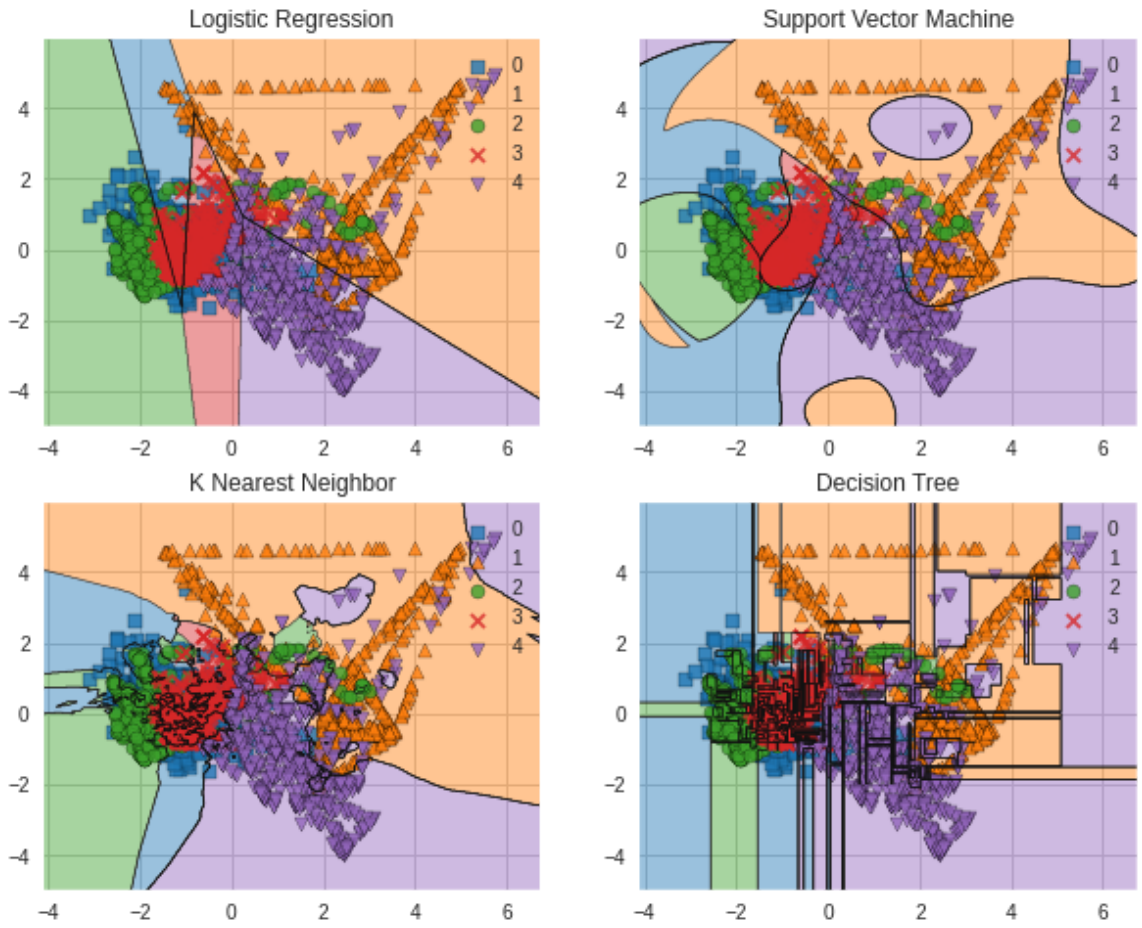
Although the algorithms chosen for our dataset showed high accuracy, we opted to implement the Ensemble Method with four algorithms. Below are the scores:

```
Accuracy: 0.92 (+/- 0.03) [Logistic Regression]
Accuracy: 0.99 (+/- 0.01) [Support Vector Machine]
Accuracy: 0.98 (+/- 0.01) [K Nearest Neighbor]
Accuracy: 0.98 (+/- 0.02) [Decision Tree]
Accuracy: 0.99 (+/- 0.01) [Ensemble]
```

Decision Boundary:

The table below shows the target classes along with the label encoded for them in multi class classification.

Label	Target - class
0	0= Blood Donor
1	0s = Suspect Blood Donor
2	1= Hepatitis
3	2= Fibrosis
4	3= Cirrhosis

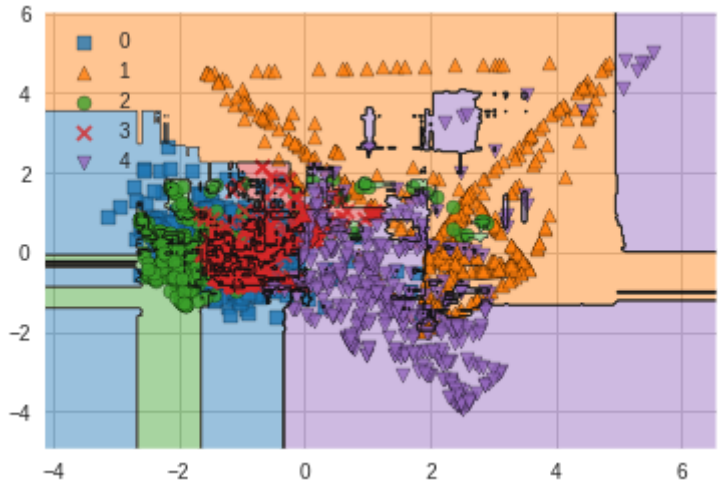


To plot decision boundaries, we have reduced our attributes set to 2 two dimensional data sets using Principal Component Analysis (PCA). Logistic Regression tries to separate the class labels linearly and we see that the data labels are spread well across respective class boundaries.

For Support Vector Machine and k Nearest Neighbor, we see separation with non-linear decision boundaries. For the Decision Tree, multiple decision boundaries are set. The decision boundaries are linear which explains why it performs slightly less accurately than Support Vector Machine.

Random Forest - Multi-class classification:

We have tried Random Forest on our multi-label dataset. Our accuracy percentage was 72.6%. For our synthetic dataset, this is a serious setback. However, given that the generated dataset were within a small margin within the multidimension, this low accuracy is rational.



Conclusion:

We have implemented traditional machine learning algorithms on a classification dataset and tried to elaborate the behaviour of the algorithms. Our studies have its shortcomings. We could not show performance metrics for multiclass classification due to time constraint. Also, our accuracy for some algorithms in multiclass algorithms were low. Feedback from domain experts could have made the results better. In addition, small instances of target labels in the original data set remained a major constraint. Collecting more data for the original data set can improve our results.

Thank you!!!