Assignment 2

CS422 - Data Mining
Amit Nikam (A20470263)

Recitation Exercises

Exercise 2:

1. Gini index = 1 - sum of (squared probabilities of a classes from total)

   #C0 = #C1 = 10

   Gini = $1 - (10/20)^2 - (10/20)^2 = 0.5$

2. Gini for individual Customer ID will be 0 as customer ids are unique and will only appear once in each class.

3.

Male		Female	
C0	6	C0	4
C1	4	C1	6

   Gini for male = $1 - (6/10)^2 - (4/10)^2 = 0.48$
   Gini for female = $1 - (4/10)^2 - (6/10)^2 = 0.48$
   Total Gini = $(10/20)(0.48) + (10/20)(0.48) = 0.48$

4.

Family		Sports		Luxury	
C0	1	C0	8	C0	7
C1	3	C1	0	C1	1

   Gini for car type family = $1 - (¼)^2 - (¾)^2 = 0.375$
   Gini for car type sports = $1 - (8/8)^2 - (0)^2 = 0$
   Gini for car type luxury = $1 - (¼)^2 - (¾)^2 = 0.2188$
   Total gini = $(4/20)(0.375) + 0 + (8/20)(0.2188) = 0.1625$

5. Gini for small shirt size：$1 - (\frac{3}{5})^2 - (\frac{2}{5})^2 = 0.48$
   Gini for medium shirt size：$1 - (3/7)^2 - (3/7)^2 = 0.489$
   Gini for large shirt size：$1 - (2/4)^2 - (2/4)^2 = 0.5$
   Gini for extra large shirt size：$1 - (2/4)^2 - (2/4)^2 = 0.5$
   Total Gini = (5/20) (0.48) + (7/20) (0.489) + (4/20) (0.5) + (4/20) (0.5) = 0.491

6. As gini is a measure of impurity we desire to get a minimal value. This means car type is better.

7. Customer ID happens to be a sort of primary key, i.e. it is unique for each customer and will only appear once in each class. This cannot help us to predict the class. Therefore it should not be used as the attribute test condition.

Exercise 3:

1. Entropy = $-\sum^{class} P(\text{class i / t})\log_2 p(\text{class i / t})$

   Entropy = -(4/9)ln(4/9) -(5/9)ln(5/9) = 0.9911

2. For a1:

Class a1	+	-
T	3	1
F	1	4

   Information gain = 0.9911 - [(4/9)*(−(¾)ln(¾)−(¼)ln(¼)) + (5/9)*(−(⅕)ln(⅕)−(⅘)ln(⅘))]
   = 0.229

   For a2:

Class a2	+	-
T	2	3
F	2	2

   Information gain = 0.9911 - [(5/9)*(−(⅖)ln(⅖)−(⅗)ln(⅗)) + (4/9)*(−(2/4)ln(2/4)−(2/4)ln(2/4))]
   = 0.0072

3. Split point is the center point between values, so we arrange the values in ascending order and find their split points:

a3	Class	Split Point
1.0	+	2.0
3.0	-	3.5
4.0	+	4.5
5.0	-	5.5
5.0	-	5.5
6.0	+	6.5
7.0	+	7.5
7.0	-	7.5
8.0	-	8.5

For split point 2.0 :
Entropy for values less than SP = -(1/1)ln(1/1) - (0/1)ln(0/1) = 0
Entropy for values greater than SP = -(⅜)ln(⅜) - (⅝)ln(⅝) = 0.951
Entropy = (1/1)(0) + (8/9)(0.951) = 0.846

Information gain for split point 2.0 = 0.991 − 0.846 = 0.415

In the same manner we get,
Information Gain for split point 3.5 = 0.0026
Information Gain for split point 4.5 = 0.0128
Information Gain for split point 5.5 = 0.0072
Information Gain for split point 6.5 = 0.0183
Information Gain for split point 7.5 = 0.1022

4. A1 shows the highest information gain among all the other features therefore a1 is the best split.

5. In a1:
   Error Count for True = 1 - max(¾,¼) = ¼
   Error Count for False = 1 - max(⅕,⅘) = ⅕
   Total Error Rate = 2/9
   In a2:
   Error Count for True = 1 - max(⅖ , ⅗) = ⅖
   Error Count for False = 1 - max(2/4 ,2/4) = 2/4
   Total Error Rate = 4/9

Thus error rate at a1 is lesser, hence the best split will be a1.

6. Gini for a1 = (4/9)(1−(¾)^2 − (¼)^2) + (5/9)(1 − (⅕)^2 − (⅘)^2) = 0.344
   Gini for a2 = (4/9)(1 − (⅖)^2 − (⅗)^2) + (5/9)(1 − (2/4)^2 − (2/4)^2) = 0.489

   As the gini index at a1 is smaller, it will be used as the best split.

Exercise 5:

1. Total Entropy: − (4/10)ln(4/10) −(6/10)ln(6/10) = 0.97

   For A split:
   Entropy for T = -(4/7)ln(4/7) - (3/7)ln(3/7) = 0.985
   Entropy for F = -(0/3)ln(0/3) - (3/3)ln(3/3) = 0
   Information gain at A = 0.97 - (7/10)(0.985) - 0 = 0.281

   For B split:
   Entropy for T = -(¾)ln(¾) - (¼)ln(¼) = 0.811
   Entropy for F = -(⅛)ln(⅛) - (⅞)ln(⅞) = 0.65
   Information gain at B = 0.97 - (4/10)(0.811) - (6/10)(0.65) = 0.256

   As A has more information gain than B, we select A for the split.

2. Total Gini: 1 − (4/10)^2 − (6/10)^2 = 0.48

   For A Split:
   Gini for T = 1 − (4/10)^2 − (6/10)^2 = 0.48
   Gini for F = 1 − 0^2 − 1^2 = 0
   Information Gain at A = 0.48 − (7/10) (0.48) = 0.14

   For B Split:
   Gini for T = 1 − (¾)^2 − (¼)^2 = 0.375
   Gini for F = 1 − (⅛)^2 − (⅞)^2 = 0.2778
   Information Gain at B = 0.48 − (4/10) (0.375) - (6/10)(0.2778) = 0.163

   Since information gain at B is more than A, attribute B is selected as best split.

3. As we saw in the above example, the entropy based splitting selects A for best split but for the same records Gini selects B as the best split. Therefore it is possible that the information gain and the gain in the Gini index favour different attributes.

Exercise 6:

1. Gini index = $1 - (7/10)^2 - (3/10)^2 = 0.42$

   Misclassification error rate: 1- Max(7/10,3/10) = 3/10

2. Gini index for C1 = $1 - (3/3)^2 - (0/3)^2 = 0$
   Gini index for C2 = $1 - (4/7)^2 - (3/7)^2 = 0.4879$

   Weighted Gini index = (3/10)(0)+(7/10)(0.48) = 0.3428

   We can consider this attribute test condition if Gini is used as the impurity measure as it is lower gini index value than parent index.

3. Misclassification error rate for C1 : 1 − max (3/3 ,0/3) = 0
   Misclassification error rate for C2 : 1 − max (4/7,3/7) = 0.4285

   Weighted Misclassification error rate :(3/10) (0) + (7/10)(0.4285) = 0.3

Exercise 7:

1. Split at Level 1:

Error for (X=0) = 1 − max (60/120 , 60/120) = 0.5
Error for (X=1) = 1 − max (40/80 , 40/80) = 0.5
Error for X = (120/200)(0.5) + (80/200)(0.5) = 0.5

Similarly,
Error for Y = 0.4
Error for Z = 0.3

Since Entropy of Z is least, we choose Z as a splitting attribute.

Split at Level 2:
When Z = 0:

Error for (X=0) = 1 − max (15/60 , 45/60) = 0.25
Error for (X=1) = 1 − max (15/40 , 25/40) = 0.375
Error for X = (60/100)(0.25) + (40/100)(0.375) = 0.3

Similarly,
Error for Y = 0.3

When Z = 1:

Error for X = 0.3
Error for Y = 0.3

We get error rate for the attribute x and y same at level 2.
Total error rate = 60/200 = 0.3

2. When X = 0:

   Error when (y=0) = 1 − max (5/60, 55/60) = 0.0834
   Error when (y=1) = 1 - max(5/60, 55/60) = 0.0834
   Error for Y = (60/120)(0.0834) + (60/120)(0.0834) = 0.0834

   Similarly,
   Error for Z = 0.25

   As error of Y is less than error of Z, Y is selected as the splitting attribute.

   In the same way, when X=1:

   Error for Y = 0.125
   Error for Z = 0.375

   Therefore as error for Y is less than error for Z, Y is selected as the splitting attribute.

   Overall Error Rate = 20/200

3. Looking at the previous two splits and their overall error rate, it could be said that the error rate in tree (b) is lower than (a). This is because greedy algorithms choose the best possible solution at the current stage and move on, it is not a globally optimum solution. So we can say that greedy algorithms do not necessarily provide the best solution.

Exercise 8:

1. Error rates:

   Total Error = 1-max(50/100,50/100) = 0.5
   At A:

	T	F
+	25	25

-	0	50

Error at (A=T) = 1-Max(25/25,0/25) = 0
Error at (A=F) = 1-Max(25/75,50/75) = 0.33333
Gain A = 0.5 - (25/100)0 + (75/100)(0.33333) = 0.25

At B:

	T	F
+	20	30
-	30	20

Error at (B=T) = 1-Max(30/50,20/50) = ⅗
Error at (B=F) = 1-Max(20/50,30/50) = ⅗
Gain at B = 0.5 - (50/100)(⅗) + (50/100)(⅗) = 0.1

At C:

	T	F
+	25	25
-	25	25

Error at (C=T) = 1-Max(25/50,25/50) = 0.5
Error at (C=F) = 1-Max(25/50,25/50) = 0.5
Gain at C = 0.5 - (50/100)(0.5) + (50/100)(0.5) = 0

Since we get the highest gain at A, A is selected as the best split.

2. A is selected as the root node. As A = T is a pure node no further splitting is needed.

   When A = F:
   Error = 1-max(25/75,50/75) = 0.33333

   At B:

	T	F
+	25	0

-	20	30

Error for (B=T) = 1 - max(25/45,20/45) = 20/45
Error for (B=F) = 1 - max(0/30,30/30) = 0
Gain at B = 0.33333 - (45/75)(20/45) - 0 = 0.0666

At C:

	T	F
+	0	25
-	25	25

Error for (C=T) = 1 - max(0/25,25/25) = 0
Error for (C=F) = 1 - max(25/50,25/50) = 0.5
Gain at C = 0.33333 - (25/75)(0) - (50/75)(0.5) = 0

As Gain at B is more than gain at C, we will split at attribute B.

3. We got an error rate of 20/100, thus 20% i.e. 20 instances are misclassified.

4. For C=T:
   Error = 25/50 = 0.5

   At A:

	T	F
+	25	0
-	0	25

Error at (A=T) = 1-Max(25/25,0/25) = 0
Error at (A=F) = 0
Gain at A = 0.5 - 0 = 0.5

At B:

	T	F

+	5	20
-	20	5

Error for (B=T) = 1 - max(5/25,20/25) = ⅕
Error for (B=F) = 1 - max(20/25,5/25) = ⅕
Gain at B = 0.3

Since gain at A is more than B, we split at attribute A.

Now for C = F:
E = 25/50 = 0.5

At A:

	T	F
+	0	25
-	0	25

E at (A=T) = 1-Max(25/25,0/25) = 0
E at (A=F) = 1-max(25/25,25/25) = 0.5
Gain at A = 0.5 - 0.5 = 0

At B:

	T	F
+	25	0
-	25	0

Error for (B=T) = 1 - max(25/25,0/25) = 0
Error for (B=F) = 1 - max(1,0) = 0
Gain at B = 0.5 - 0 = 0.5

Since gain at B is more than A, we split on attribute B.

Overall error rate of trees is 0 as both gains were the same values.

5. Is it clearly observed that we got the best results in the decision tree we got in (d). Although by greedy method we could not get this tree. This tells us that greedy nature does not always result in the best decision tree.

Exercise 12:

1. We know that error rate = 1 - accuracy

   Therefore we find error rates for both the trees w.r.t both the data sets.

   T10:
   Ea = 1-0.86=0.14
   Eb = 1-0.84=0.16

   T100:
   Ea = 1-0.97=0.03
   Eb = 1-0.77=0.23

   Observing the error rates of both these trees we can see that the T10 decision tree is more stable in the sense that it will give constant output. Since it has small output variance, the classification model with 10 nodes is better.

2. T10: Ea+b = 1 - 0.85 = 0.15
   T100: Ea+b = 1 - 0.87 = 0.13

   After observing the values of error rates for a+b dataset, we can say that the variance in output still remains small for the classifier with 10 nodes. Therefore we will still selected T10 as a better classification model.

Practicum Problems

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
from sklearn import tree
from sklearn import model_selection
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.decomposition import PCA
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt

%matplotlib inline
```

Problem 1

In [2]:
```
# Prepare Data
iris = load_iris()
X_train, X_test, y_train, y_test = model_selection.train_test_split(iris.data, iris.target, test_size=0.2)
```

In [3]:
```
#
### Decision Tree For Depth 1
#
clf = DecisionTreeClassifier(min_samples_split=5, min_samples_leaf=2, max_depth=1)
clf = clf.fit(X_train, y_train)
y_hat = clf.predict(X_test)
```

In [4]:
```
# print metrics
print(f'Precision: {metrics.precision_score(y_test,y_hat,average=None)}')
print(f'Recall: {metrics.recall_score(y_test,y_hat,average=None)}')
print(f'F1 score: {metrics.f1_score(y_test,y_hat,average=None)}')
print(f'\n\n {"":<9}| Precision{"":<9}| Recall{"":<9}| Weighted')
print(f'Micro:    {metrics.precision_recall_fscore_support(y_test,y_hat,average="micro")}')
print(f'Macro:    {metrics.precision_recall_fscore_support(y_test,y_hat,average="macro")}')
print(f'Weighted: {metrics.precision_recall_fscore_support(y_test,y_hat,average="weighted")}')
```

```
Precision: [1.         0.         0.41176471]
Recall: [1. 0. 1.]
F1 score: [1.         0.         0.58333333]


          | Precision         | Recall         | Weighted
Micro:    (0.6666666666666666, 0.6666666666666666, 0.6666666666666666, None)
Macro:    (0.4705882352941176, 0.6666666666666666, 0.5277777777777778, None)
Weighted: (0.5294117647058824, 0.6666666666666666, 0.5694444444444445, None)
```

In [5]:
```
#
### Decision Tree For Depth 2
#
clf = DecisionTreeClassifier(min_samples_split=5, min_samples_leaf=2, max_depth=2)
clf = clf.fit(X_train, y_train)
y_hat = clf.predict(X_test)
```

In [6]:
```
# print metrics
print(f'Precision: {metrics.precision_score(y_test,y_hat,average=None)}')
print(f'Recall: {metrics.recall_score(y_test,y_hat,average=None)}')
print(f'F1 score: {metrics.f1_score(y_test,y_hat,average=None)}')
print(f'\n\n {"":<9}| Precision{"":<9}| Recall{"":<9}| Weighted')
print(f'Micro:    {metrics.precision_recall_fscore_support(y_test,y_hat,average="micro")}')
print(f'Macro:    {metrics.precision_recall_fscore_support(y_test,y_hat,average="macro")}')
print(f'Weighted: {metrics.precision_recall_fscore_support(y_test,y_hat,average="weighted")}')
```

```
Precision: [1.         0.90909091 1.        ]
Recall: [1.         1.         0.85714286]
F1 score: [1.         0.95238095 0.92307692]


          | Precision         | Recall         | Weighted
Micro:    (0.9666666666666667, 0.9666666666666667, 0.9666666666666667, None)
Macro:    (0.9696969696969697, 0.9523809523809524, 0.9584859584859585, None)
Weighted: (0.9696969696969696, 0.9666666666666667, 0.9661782661782662, None)
```

In [7]:
```
#
### Decision Tree For Depth 3
#
clf = DecisionTreeClassifier(min_samples_split=5, min_samples_leaf=2, max_depth=3)
clf = clf.fit(X_train, y_train)
y_hat = clf.predict(X_test)
```

In [8]:
```
# print metrics
print(f'Precision: {metrics.precision_score(y_test,y_hat,average=None)}')
print(f'Recall: {metrics.recall_score(y_test,y_hat,average=None)}')
print(f'F1 score: {metrics.f1_score(y_test,y_hat,average=None)}')
print(f'\n\n {"":<9}| Precision{"":<9}| Recall{"":<9}| Weighted')
print(f'Micro:    {metrics.precision_recall_fscore_support(y_test,y_hat,average="micro")}')
print(f'Macro:    {metrics.precision_recall_fscore_support(y_test,y_hat,average="macro")}')
print(f'Weighted: {metrics.precision_recall_fscore_support(y_test,y_hat,average="weighted")}')
```

```
Precision: [1. 1. 1.]
Recall: [1. 1. 1.]
F1 score: [1. 1. 1.]


          | Precision         | Recall         | Weighted
Micro:    (1.0, 1.0, 1.0, None)
Macro:    (1.0, 1.0, 1.0, None)
Weighted: (1.0, 1.0, 1.0, None)
```

In [9]:
```
#
### Decision Tree For Depth 4
#
clf = DecisionTreeClassifier(min_samples_split=5, min_samples_leaf=2, max_depth=4)
clf = clf.fit(X_train, y_train)
y_hat = clf.predict(X_test)
```

In [10]:
```
# print metrics
print(f'Precision: {metrics.precision_score(y_test,y_hat,average=None)}')
print(f'Recall: {metrics.recall_score(y_test,y_hat,average=None)}')
print(f'F1 score: {metrics.f1_score(y_test,y_hat,average=None)}')
print(f'\n\n {"":<9}| Precision{"":<9}| Recall{"":<9}| Weighted')
print(f'Micro:    {metrics.precision_recall_fscore_support(y_test,y_hat,average="micro")}')
print(f'Macro:    {metrics.precision_recall_fscore_support(y_test,y_hat,average="macro")}')
print(f'Weighted: {metrics.precision_recall_fscore_support(y_test,y_hat,average="weighted")}')
```

```
Precision: [1. 1. 1.]
Recall: [1. 1. 1.]
F1 score: [1. 1. 1.]


          | Precision         | Recall         | Weighted
Micro:    (1.0, 1.0, 1.0, None)
Macro:    (1.0, 1.0, 1.0, None)
Weighted: (1.0, 1.0, 1.0, None)
```

In [11]:
```
#
### Decision Tree For Depth 5
#
clf = DecisionTreeClassifier(min_samples_split=5, min_samples_leaf=2, max_depth=5)
clf = clf.fit(X_train, y_train)
y_hat = clf.predict(X_test)
```

In [12]:
```
# print metrics
print(f'Precision: {metrics.precision_score(y_test,y_hat,average=None)}')
print(f'Recall: {metrics.recall_score(y_test,y_hat,average=None)}')
print(f'F1 score: {metrics.f1_score(y_test,y_hat,average=None)}')
print(f'\n\n {"":<9}| Precision{"":<9}| Recall{"":<9}| Weighted')
print(f'Micro:    {metrics.precision_recall_fscore_support(y_test,y_hat,average="micro")}')
print(f'Macro:    {metrics.precision_recall_fscore_support(y_test,y_hat,average="macro")}')
print(f'Weighted: {metrics.precision_recall_fscore_support(y_test,y_hat,average="weighted")}')
```

```
Precision: [1. 1. 1.]
Recall: [1. 1. 1.]
F1 score: [1. 1. 1.]


          | Precision         | Recall         | Weighted
Micro:    (1.0, 1.0, 1.0, None)
Macro:    (1.0, 1.0, 1.0, None)
Weighted: (1.0, 1.0, 1.0, None)
```

Conclusion

Lowest Precision is 0.39 at max_depth = 1

- Precision is the ratio between True Positive and sum of True Positive and False Position
- High precision relates to a low false positive rate.
- We are getting a low precision because of our low value of True Positive rate which is not good.

Highest Recall is achieved at max_depth = 2 and is constant for depths until 5

- A High Recall consider as Low False Negative rate. It provides us information regarding classifier's performance about false negatives.
- High recall relates to a low false negative rate.
- Thus at depth 1 we have low number of False Negative rate, which is desired.

F1 score:

- F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.
- We get better F1 score for max-depths 2 to 5.

Weighted-average Method:

- calculates the F1 score for each class independently but when it adds them together uses a weight that depends on the number of true labels of each class.

Micro-average Method:

- uses the global number of TP, FN, FP and calculates the F1 directly.

Macro-average Method:

- calculates the F1 separated by class but not using weights for the aggregation

---

Problem 2

In [13]:
```
# import data
bc_df = pd.read_csv('./data/breast-cancer-wisconsin.data', header=None, na_values='?')
bc_df.columns = ['Sample Code Number','Clump Thickness', 'Uniformity of Cell Size', 'Uniformity of Cell Shape','N
bc_df = bc_df.dropna(how='any', axis=0)
```

In [14]:
```
bc_df
```

Out[14]:

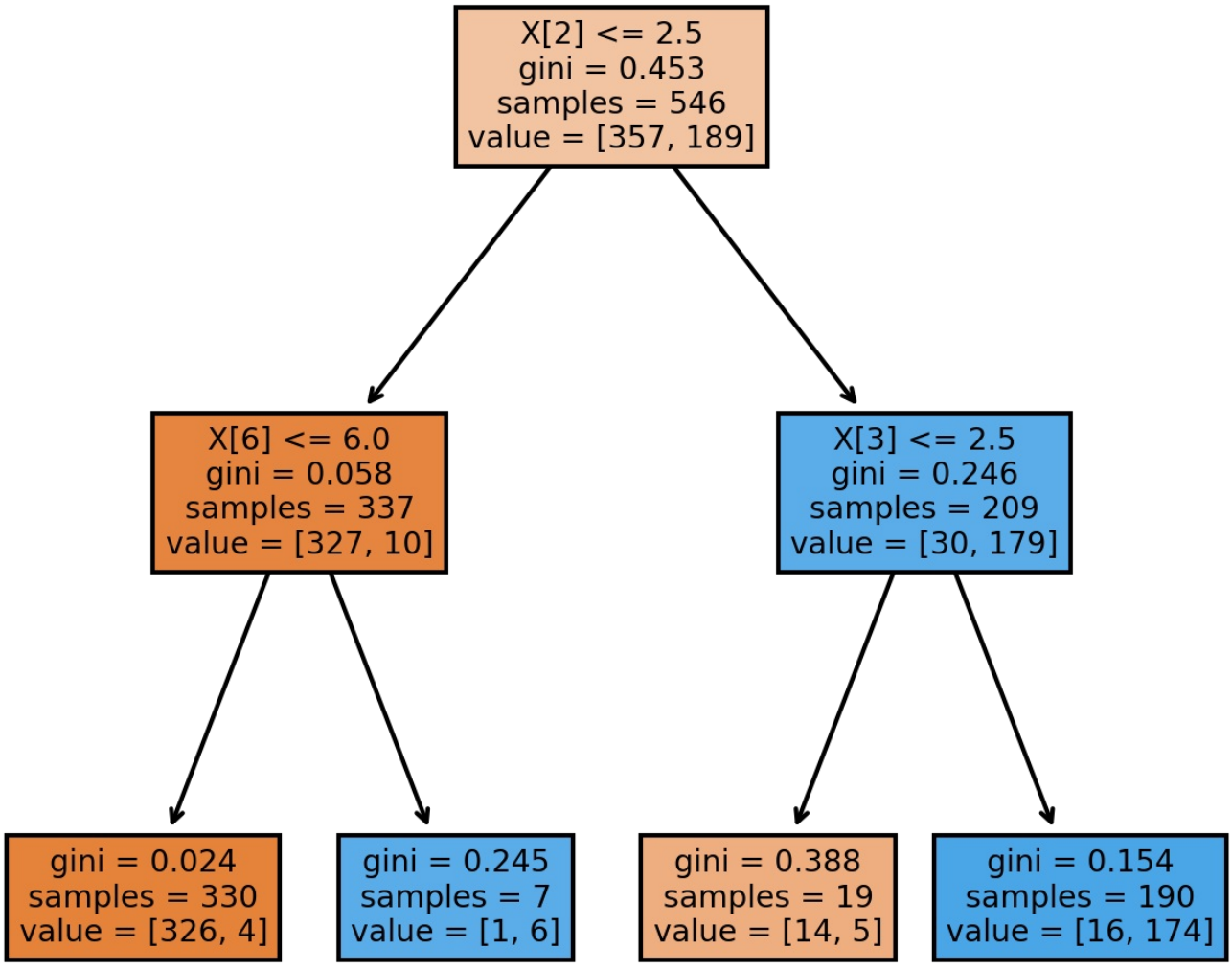
	Sample Code Number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1.0	3	1	1	2
1	1002945	5	4	4	5	7	10.0	3	2	1	2
2	1015425	3	1	1	1	2	2.0	3	1	1	2
3	1016277	6	8	8	1	3	4.0	3	7	1	2
4	1017023	4	1	1	3	2	1.0	3	1	1	2
...	...	...	...	...	...	...	...	...	...	...	...
694	776715	3	1	1	1	3	2.0	1	1	1	2
695	841769	2	1	1	1	2	1.0	1	1	1	2
696	888820	5	10	10	3	7	3.0	8	10	2	4
697	897471	4	8	6	4	3	4.0	10	6	1	4
698	897471	4	8	8	5	4	5.0	10	4	1	4

683 rows × 11 columns

In [15]:
```
# prepare data
X_train,X_test,y_train,y_test = model_selection.train_test_split(bc_df.iloc[:,:-1],bc_df.iloc[:,-1],test_size=0.2

# fit decision tree
clf = DecisionTreeClassifier(min_samples_split=5, min_samples_leaf=2, max_depth=2)
clf = clf.fit(X_train, y_train)

# check metrics
y_hat = clf.predict(X_test)
```

```
# print metrics
print(f'Classifier Report:\n\n{metrics.classification_report(y_test, y_hat)}\n')
print(f'Confusion Matrix:\n\n{metrics.confusion_matrix(y_test, y_hat)}')

# Plot Decision Tree
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (5,5), dpi=300)
_ = tree.plot_tree(clf, filled=True)
```

```
Classifier Report:

              precision    recall  f1-score   support

           2       0.98      0.95      0.97        87
           4       0.92      0.96      0.94        50

    accuracy                           0.96       137
   macro avg       0.95      0.96      0.95       137
weighted avg       0.96      0.96      0.96       137


Confusion Matrix:

[[83  4]
 [ 2 48]]
```

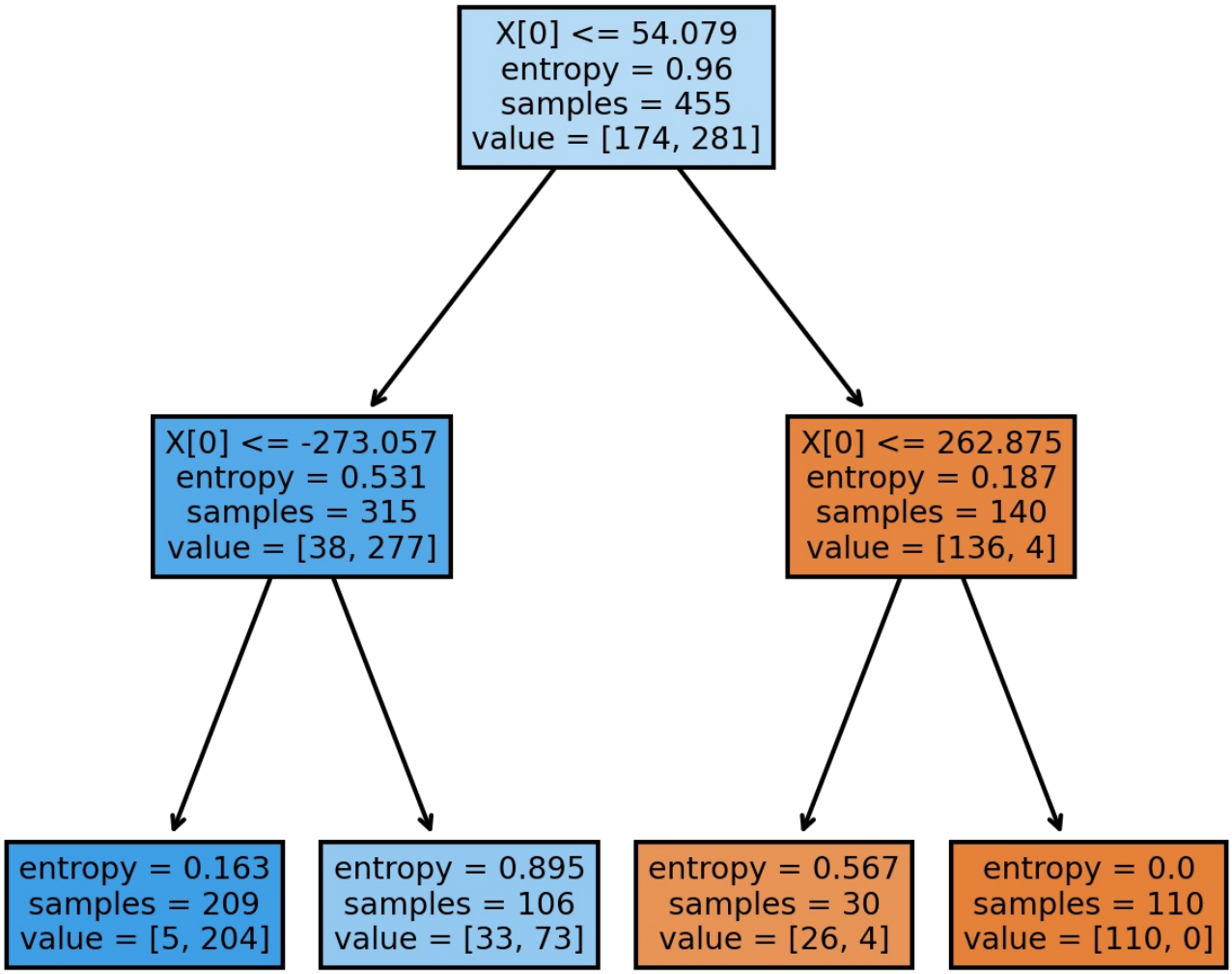


In [29]:

```
# fit decision tree (entropy criteria)
clf = DecisionTreeClassifier(criterion = 'entropy', min_samples_split=5, min_samples_leaf=2, max_depth=2)
clf = clf.fit(X_train, y_train)

# check metrics
```

```
y_hat = clf.predict(X_test)

# print metrics
print(f'Classifier Report:\n\n{metrics.classification_report(y_test, y_hat)}\n')
print(f'Confusion Matrix:\n\n{metrics.confusion_matrix(y_test, y_hat)}')

# Plot Decision Tree
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (5,5), dpi=300)
_ = tree.plot_tree(clf, filled=True)
```

```
Classifier Report:

              precision    recall  f1-score   support

           1       0.91      0.84      0.88        38
           2       0.92      0.96      0.94        76

    accuracy                           0.92       114
   macro avg       0.92      0.90      0.91       114
weighted avg       0.92      0.92      0.92       114


Confusion Matrix:

[[32  6]
 [ 3 73]]
```



Conclusion

Feature selected for first split is x[2] i.e. 'Uniformity of Cell Size':

Gini Index = 1 - $(357/546)^2$ - $(189/546)^2$ = 0.4526

Misclassification error = 1 - (357/546) = 0.3462

Entropy = 0.96

Entropy(Left child) = 0.531

Entropy(right child) = 0.187

Information Gain = 0.96 - (174/455)(0.531) - (281/455)(0.187) = 0.6414

Splitting value is 2.5 (with gini)

Problem 3

In [16]:
```
bc_df = pd.read_csv("./data/wdbc.data", header=None)
bc_df.columns = ['ID number','Diagnosis','mean radius', 'mean texture', 'mean perimeter' ,'mean area', 'mean smoo
 'mean concave points', 'mean symmetry' ,'mean fractal dimension','radius error' ,'texture error', 'perimeter err
 'smoothness error', 'compactness error', 'concavity error','concave points error' ,'symmetry error', 'fractal di
 'worst smoothness', 'worst compactness','worst concavity','worst concave points' ,'worst symmetry' ,'worst fract
target = bc_df[['Diagnosis']].copy()
target = target.replace('M',1)
target = target.replace('B',2)
```

In [17]:
```
bc_df.head()
```

Out[17]:

	ID number	Diagnosis	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	...	worst radius	worst texture	worst perimeter	wors are
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	25.38	17.33	184.60	2019.
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	24.99	23.41	158.80	1956.
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	23.57	25.53	152.50	1709.
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	14.91	26.50	98.87	567.
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	22.54	16.67	152.20	1575.

5 rows × 32 columns

In [18]:
```
#
### Original Decision Tree
#

# prepare data
X_train,X_test,y_train,y_test = model_selection.train_test_split(bc_df.iloc[:,2:],target,test_size=0.2)

# fit decision tree
clf = DecisionTreeClassifier(min_samples_split=5, min_samples_leaf=2, max_depth=2)
clf = clf.fit(X_train, y_train)

# check metrics
y_hat = clf.predict(X_test)

# print metrics
print(f'Classifier Report:\n\n{metrics.classification_report(y_test, y_hat)}\n')
print(f'Confusion Matrix:\n\n{metrics.confusion_matrix(y_test, y_hat)}')
cm = metrics.confusion_matrix(y_test,y_hat)
tp, fp, fn, tn = cm[0][0],cm[0][1],cm[1][0],cm[1][1]
print(f'\nTP :{tp}')
print(f'FP :{fp}')
print(f'FPR :{fp/(fp+tn)}')
print(f'TPR :{tp/(tp+fp)}')
```

```
Classifier Report:

              precision    recall  f1-score   support

           1       0.98      0.86      0.92        51
           2       0.90      0.98      0.94        63

    accuracy                           0.93       114
   macro avg       0.94      0.92      0.93       114
weighted avg       0.93      0.93      0.93       114
```

```
Confusion Matrix:

[[44  7]
 [ 1 62]]

TP :44
FP :7
FPR :0.10144927536231885
TPR :0.8627450980392157
```

In [19]:
```
#
### PCA1 Decision Tree
#

# PCA Decomposition
pca = PCA(n_components=1)
pca_df = pd.DataFrame(pca.fit_transform(bc_df.iloc[:,2:]))
print(pca.explained_variance_ratio_)
```

```
[0.98204467]
```

In [20]:
```
# prepare data
X_train,X_test,y_train,y_test = model_selection.train_test_split(pca_df,target,test_size=0.2)

# fit decision tree
clf = DecisionTreeClassifier(min_samples_split=5, min_samples_leaf=2, max_depth=2)
clf = clf.fit(X_train, y_train)

# check metrics
y_hat = clf.predict(X_test)

# print metrics
print(f'Classifier Report:\n\n{metrics.classification_report(y_test, y_hat)}\n')
print(f'Confusion Matrix:\n\n{metrics.confusion_matrix(y_test, y_hat)}')
cm = metrics.confusion_matrix(y_test,y_hat)
tp, fp, fn, tn = cm[0][0],cm[0][1],cm[1][0],cm[1][1]
print(f'\nTP :{tp}')
print(f'FP :{fp}')
print(f'FPR :{fp/(fp+tn)}')
print(f'TPR :{tp/(tp+fp)}')
```

```
Classifier Report:

              precision    recall  f1-score   support

           1       0.97      0.81      0.88        42
           2       0.90      0.99      0.94        72

    accuracy                           0.92       114
   macro avg       0.94      0.90      0.91       114
weighted avg       0.93      0.92      0.92       114


Confusion Matrix:

[[34  8]
 [ 1 71]]

TP :34
FP :8
FPR :0.10126582278481013
TPR :0.8095238095238095
```

In [21]:
```
#
### PCA2 Decision Tree
#

# PCA Decomposition
pca = PCA(n_components=2)
pca_df = pd.DataFrame(pca.fit_transform(bc_df.iloc[:,2:]))
print(pca.explained_variance_ratio_)
```

```
[0.98204467 0.01617649]
```

In [22]:
```
# prepare data
X_train,X_test,y_train,y_test = model_selection.train_test_split(pca_df,target,test_size=0.2)
```

```
# fit decision tree
clf = DecisionTreeClassifier(min_samples_split=5, min_samples_leaf=2, max_depth=2)
clf = clf.fit(X_train, y_train)

# check metrics
y_hat = clf.predict(X_test)

# print metrics
print(f'Classifier Report:\n\n{metrics.classification_report(y_test, y_hat)}\n')
print(f'Confusion Matrix:\n\n{metrics.confusion_matrix(y_test, y_hat)}')
cm = metrics.confusion_matrix(y_test,y_hat)
tp, fp, fn, tn = cm[0][0],cm[0][1],cm[1][0],cm[1][1]
print(f'\nTP :{tp}')
print(f'FP :{fp}')
print(f'FPR :{fp/(fp+tn)}')
print(f'TPR :{tp/(tp+fp)}')
```

```
Classifier Report:

              precision    recall  f1-score   support

           1       0.91      0.84      0.88        38
           2       0.92      0.96      0.94        76

    accuracy                           0.92       114
   macro avg       0.92      0.90      0.91       114
weighted avg       0.92      0.92      0.92       114


Confusion Matrix:

[[32  6]
 [ 3 73]]

TP :32
FP :6
FPR :0.0759493670886076
TPR :0.8421052631578947
```

Conclusion

It is observed that by using the principal components, we are able to get better precision and better recall values in averages in comparison of original continues data. Therefore using PCA based data is better than normal continuous data. This happens because the PCA axis are well adjusted in the new dimension thus giving better classification of data.

Problem 4

In [23]:
```
# Generate DataFrame

np_array1 = np.random.normal(5,2,1000)
np_array1 = np_array1[...,np.newaxis]
df_class1 = pd.DataFrame(np_array1)
df_class1['Class'] = np.repeat('c1',1000)
df_class1.columns = ['Numbers','Class']

np_array2 = np.random.normal(-5,2,1000)
np_array2 = np_array2[...,np.newaxis]
df_class2 = pd.DataFrame(np_array2)
df_class2['Class'] = np.repeat('c2',1000)
df_class2.columns = ['Numbers','Class']

df_full = df_class1.append(df_class2, ignore_index = True)
```

In [24]:
```
df_full
```

Out[24]:

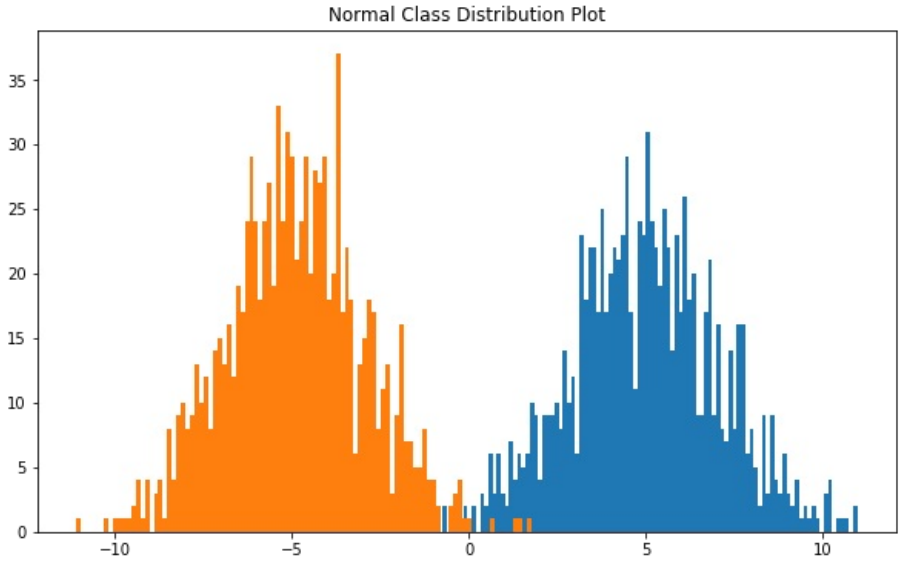
	Numbers	Class
0	7.416078	c1
1	3.599961	c1
2	10.215792	c1
3	7.804494	c1
4	2.772446	c1
...	...	...
1995	-6.001731	c2
1996	-4.604181	c2

1997	-1.957377	c2
1998	-4.260302	c2
1999	-7.739767	c2

2000 rows × 2 columns

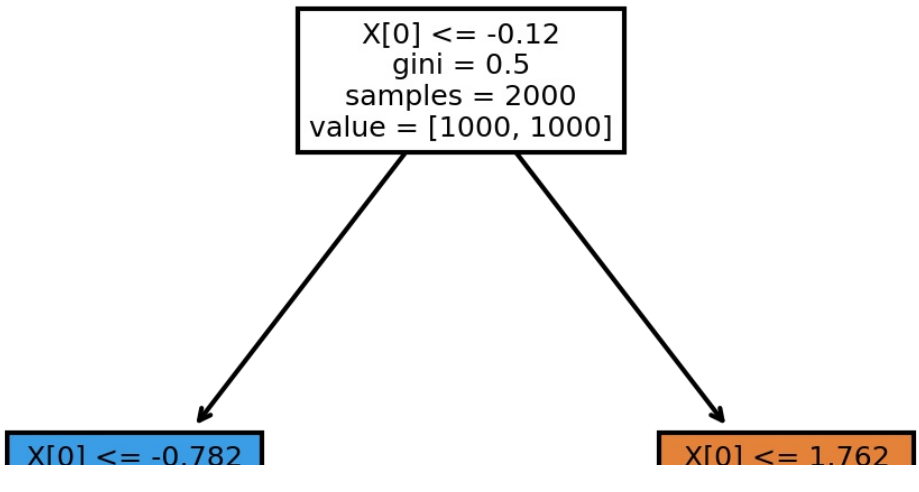
In [25]:
```
# Visualize our Class Distributions
plt.figure(figsize=(10,6))
_ = plt.hist(np_array1,bins=100)
_ = plt.hist(np_array2,bins=100)
plt.title('Normal Class Distribution Plot')
```

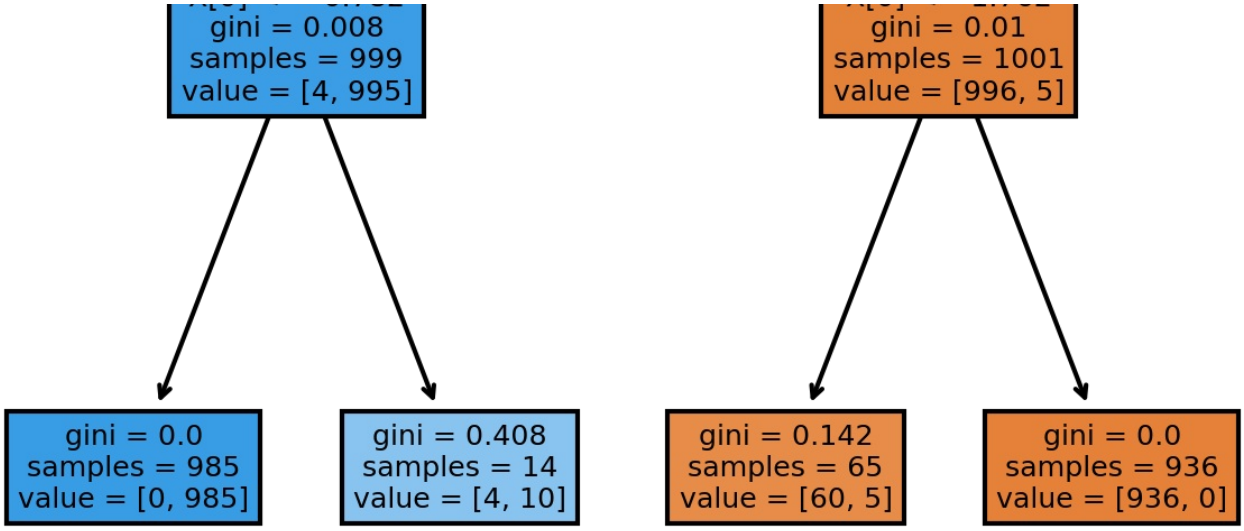
Out[25]:
```
Text(0.5, 1.0, 'Normal Class Distribution Plot')
```



In [26]:
```
# Make Decision Tree

clf = DecisionTreeClassifier(max_depth = 2)
feature = df_full['Numbers']
target = df_full['Class']
clf = clf.fit(feature[...,np.newaxis],target)
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (5,5), dpi=300)
tree.plot_tree(clf, filled=True)
```

Out[26]:
```
[Text(581.25, 943.75, 'X[0] <= -0.12\ngini = 0.5\nsamples = 2000\nvalue = [1000, 1000]'),
 Text(290.625, 566.25, 'X[0] <= -0.782\ngini = 0.008\nsamples = 999\nvalue = [4, 995]'),
 Text(145.3125, 188.75, 'gini = 0.0\nsamples = 985\nvalue = [0, 985]'),
 Text(435.9375, 188.75, 'gini = 0.408\nsamples = 14\nvalue = [4, 10]'),
 Text(871.875, 566.25, 'X[0] <= 1.762\ngini = 0.01\nsamples = 1001\nvalue = [996, 5]'),
 Text(726.5625, 188.75, 'gini = 0.142\nsamples = 65\nvalue = [60, 5]'),
 Text(1017.1875, 188.75, 'gini = 0.0\nsamples = 936\nvalue = [936, 0]')]
```





In [27]:
```
# Emperical distribution
df_full.describe()
```

Out[27]:

	Numbers
count	2000.000000
mean	0.034125
std	5.332324
min	-11.090201
25%	-4.972743
50%	-0.114037
75%	4.984962
max	10.983598

Conclusion: The threshold value for features first split is -0.12; Comparing this to the empirical distribution, we observe that the threshold value is very close to the median. This can be better understood by observing the visualized normal distributions of our input features. We can observe a clear distinction in the value and frequencies of the two classes, this inturn brings the threshold value close to median. The reason it is not exactly equal to median is because some values in a small portion of the range might overlap.
