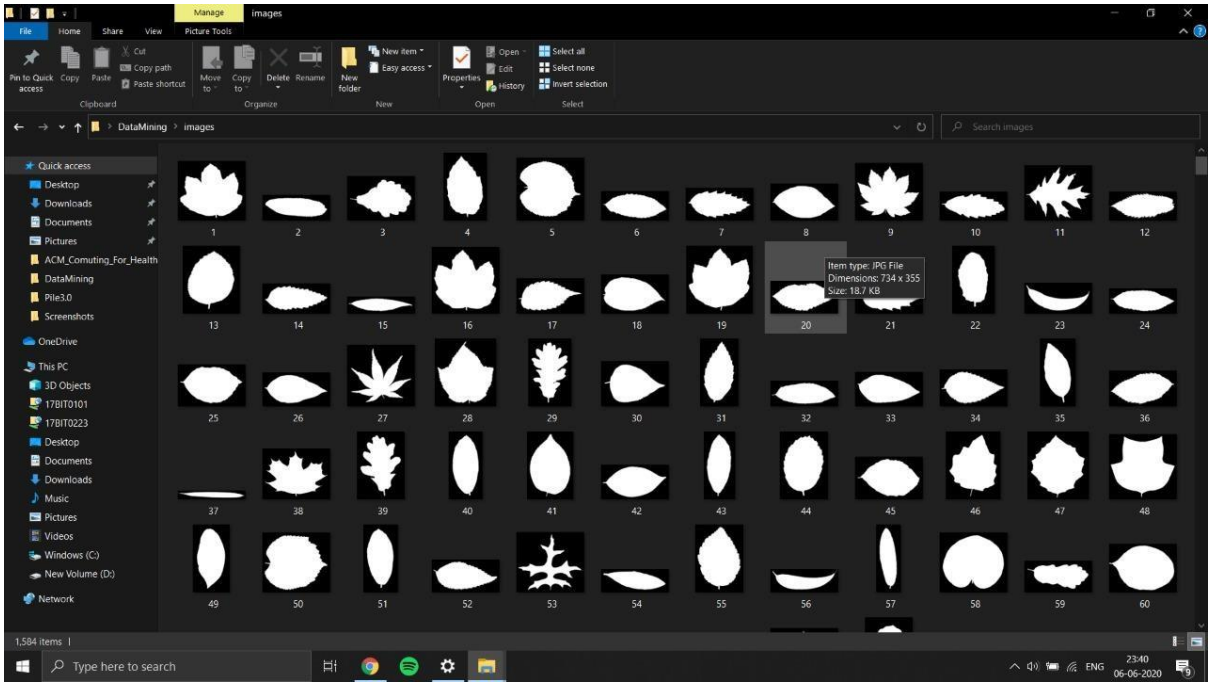
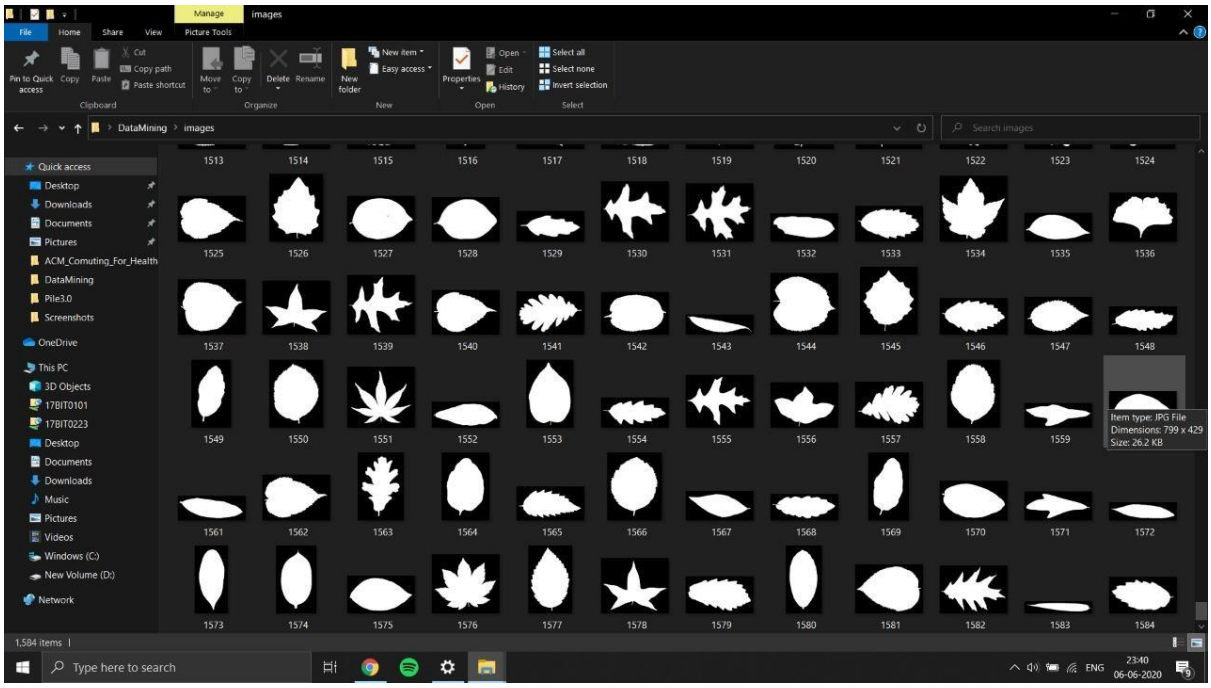
Database used:

The dataset was taken from Kaggle. Link for which is: https://www.kaggle.com/c/leaf-classification/data

Sample screenshot of dataset images:-





DBSCAN:

Density-based spatial clustering of applications with noise (DBSCAN) is a well-known data clustering algorithm that is commonly used in data mining and machine learning.

Based on a set of points (let's think in a bidimensional space as exemplified in the figure), DBSCAN groups together points that are close to each other based on a distance measurement (usually Euclidean distance) and a minimum number of points. It also marks as outliers the points that are in low-density regions.

Parameters:

The DBSCAN algorithm basically requires 2 parameters:

eps: specifies how close points should be to each other to be considered a part of a cluster. It means that if the distance between two points is lower or equal to this value (eps), these points are considered neighbors.

minPoints: the minimum number of points to form a dense region. For example, if we set the minPoints parameter as 5, then we need at least 5 points to form a dense region.

Parameter estimation:

The parameter estimation is a problem for every data mining task. To choose good parameters we need to understand how they are used and have at least a basic previous knowledge about the data set that will be used.

eps: if the eps value chosen is too small, a large part of the data will not be clustered. It will be considered outliers because don't satisfy the number of points to create a dense region. On the other hand, if the value that was chosen is too high, clusters will merge and the majority of objects will be in the same cluster. The eps should be chosen based on the distance of the dataset (we can use a k-distance graph to find it), but in general small eps values are preferable.

minPoints: As a general rule, a minimum minPoints can be derived from a number of dimensions (D) in the data set, as minPoints $\geq D + 1$. Larger values are usually better for data sets with noise and will form more significant clusters. The minimum value for the minPoints must be 3, but the larger the data set, the larger the minPoints value that should be chosen.

- In our project, we've taken a data set of 1600 images of leave. We converted the images into black and white format, so that colours distinction is reduced to two. Conversion to black and white is done by converting the colour values to binary: black and white.
- We used two approaches to proceed with DB Scan: **Image approach and CSV file approach.**

Image approach: In this approach, we utilized every pixel of the every image and assigned an intensity value to it between 0 and 1. Since images are black and white, intensity values will

be binary (either 0 or 1). Once the intensity value of every single pixel of every single image is assigned, a normalized value is given to every image according to their pixel-intensity value.

The normalized values are then plotted on a graph and allowed to form clusters with neighbouring plotted points. The epoch is calculated by applying repeated iteration on these points and after 10 epochs, accuracy came out to be ~40%. Hence the clusters are formed and these clusters can be scanned through.

In [28]:

```
train_data
```

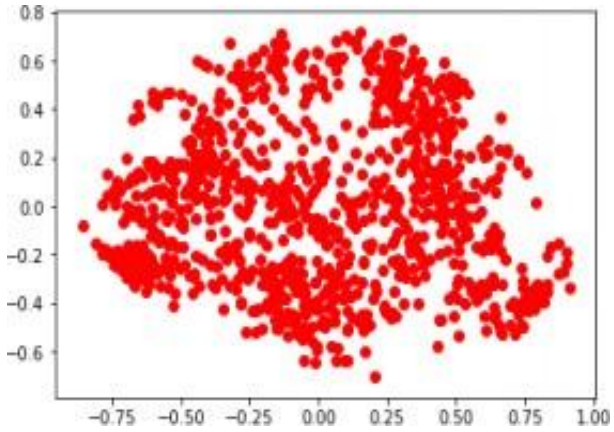
Out[28]:

	id	species	margin1	margin2	margin3	margin4	margin5	margin6
0	1	Acer_Opalus	0.007812	0.023438	0.023438	0.003906	0.011719	0.000766
1	2	Pterocarya_Stenoptera	0.005859	0.000000	0.031250	0.015625	0.025391	0.001953
2	3	Quercus_Hartwissiana	0.005859	0.009766	0.019531	0.007812	0.003906	0.005859
3	5	Tilia_Tomentosa	0.000000	0.003906	0.023438	0.005859	0.021484	0.019531
4	6	Quercus_Variabilis	0.005859	0.003906	0.048828	0.009766	0.013672	0.015625
...	...	...	...	...	...	...	...	...
985	1575	Magnolia_Salicifolia	0.060547	0.119140	0.007812	0.003906	0.000000	0.148440
986	1578	Acer_Pictum	0.001953	0.003906	0.021484	0.107420	0.001953	0.000000
987	1581	Alnus_Maximowiczii	0.001953	0.003906	0.000000	0.021484	0.078125	0.003906
988	1582	Quercus_Rubra	0.000000	0.000000	0.046875	0.056641	0.009766	0.000000
989	1584	Quercus_Afares	0.023438	0.019531	0.031250	0.015625	0.005859	0.019531

990 rows × 194 columns

CSV file approach:

In this approach, we took the data of all the 1600 images and we defined parameters: Margin and texture between 0 and 1 for every image. These values, P1 and P2 were then normalized and assigned between -1 to 1 for every image and plotted on graph. We then took the epsilon value as 0.78 by trial and error and minimum number of points: 30. The black colour is assigned -1 (outliers) and rest are RGB (Red, green, black). Using these values, DBSCAN algorithm is applied and graph is plotted.


0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

CODE AND OUTPUT:

1) USING THE CSV FILE:

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.cluster import DBSCAN
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA
```

In [19]:

```
X = pd.read_csv("C:/Users/Lenovo/Desktop/DataMining/train.csv")

# Dropping the CUST_ID column from the data
# X = X.drop('id', axis = 1)
X = X.drop('species', axis = 1)
# Handling the missing values
X.fillna(method ='ffill', inplace = True)

print(X.head())
```

```
   id   margin1   margin2   margin3   margin4   margin5   margin6   margin
7  \
0   1  0.007812  0.023438  0.023438  0.003906  0.011719  0.009766  0.02734
4
1   2  0.005859  0.000000  0.031250  0.015625  0.025391  0.001953  0.01953
1
2   3  0.005859  0.009766  0.019531  0.007812  0.003906  0.005859  0.06835
9
3   5  0.000000  0.003906  0.023438  0.005859  0.021484  0.019531  0.02343
8
4   6  0.005859  0.003906  0.048828  0.009766  0.013672  0.015625  0.00585
9

   margin8   margin9  ...  texture55  texture56  texture57  texture58  \
0      0.0  0.001953  ...   0.007812   0.000000   0.002930   0.002930
1      0.0  0.000000  ...   0.000977   0.000000   0.000000   0.000977
2      0.0  0.000000  ...   0.154300   0.000000   0.005859   0.000977
3      0.0  0.013672  ...   0.000000   0.000977   0.000000   0.000000
4      0.0  0.000000  ...   0.096680   0.000000   0.021484   0.000000

   texture59  texture60  texture61  texture62  texture63  texture64
0   0.035156        0.0        0.0   0.004883   0.000000   0.025391
1   0.023438        0.0        0.0   0.000977   0.039062   0.022461
2   0.007812        0.0        0.0   0.000000   0.020508   0.002930
3   0.020508        0.0        0.0   0.017578   0.000000   0.047852
4   0.000000        0.0        0.0   0.000000   0.000000   0.031250

[5 rows x 193 columns]
```

In [20]:

```
# Scaling the data to bring all the attributes to a comparable level
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Normalizing the data so that
# the data approximately follows a Gaussian distribution
X_normalized = normalize(X_scaled)

# Converting the numpy array into a pandas DataFrame
X_normalized = pd.DataFrame(X_normalized)
```

```
C:\Users\Lenovo\Anaconda3\envs\tensorflow\lib\site-packages\sklearn\prepro
cessing\data.py:625: DataConversionWarning: Data with input dtype int64, f
loat64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
C:\Users\Lenovo\Anaconda3\envs\tensorflow\lib\site-packages\sklearn\base.p
y:462: DataConversionWarning: Data with input dtype int64, float64 were al
l converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

In [21]:

```
pca = PCA(n_components = 2)
X_principal = pca.fit_transform(X_normalized)
X_principal = pd.DataFrame(X_principal)
X_principal.columns = ['P1', 'P2']
print(X_principal.head())
```

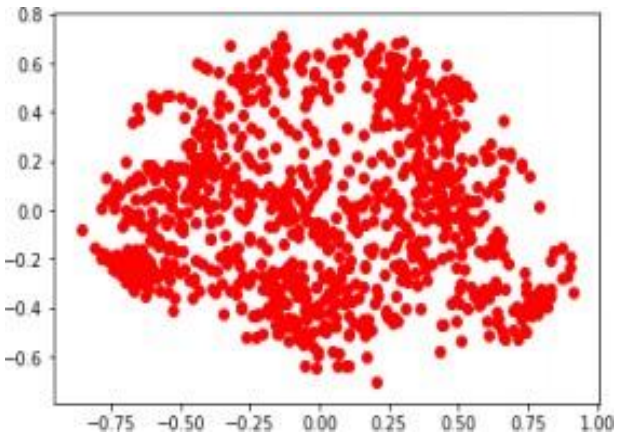
```
         P1        P2
0 -0.360054  0.309934
1  0.232383  0.501790
2  0.396450 -0.027263
3 -0.131966  0.598132
4  0.267184 -0.028494
```

In [29]:

```
plt.scatter(X_principal['P1'],X_principal['P2'],color = 'r')
```

Out[29]:

```
<matplotlib.collections.PathCollection at 0x1d969a36dd8>
```



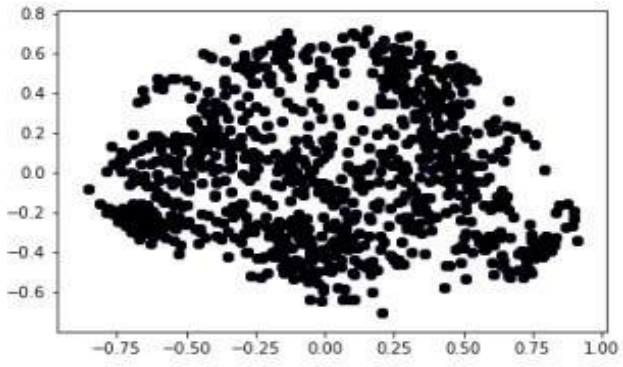
In [98]:

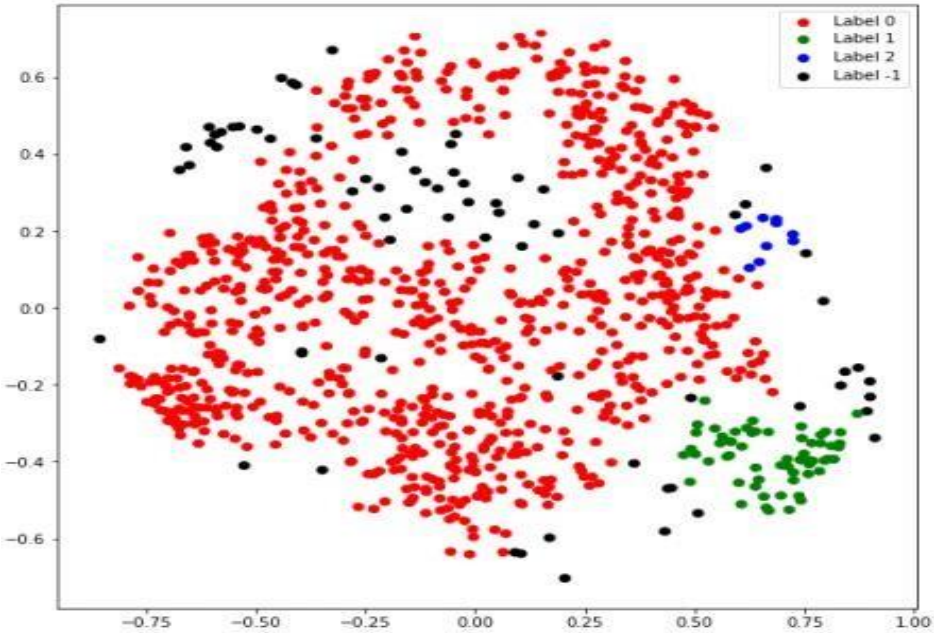
```
# Numpy array of all the cluster labels assigned to each data point
db_default = DBSCAN(eps = .079, min_samples = 10).fit(X_principal)
labels = db_default.labels_
print(labels)
```

```
[ 0  0  0  0  0  0  0  1  0  0 -1  0 -1  0 -1  0  0  0  0  0  0 -1  0  0
  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0 -1  0  0  0 -1  0  0  0 -1
  0  0  0  0  1  0  0  0 -1  0  0  0  0  1  0  0  0  0 -1  0  0  0  0  1
  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0 -1  0  0  0  0  1  0  0  0
  0  0  0  0  0  0  2  0  1  0  0  0  0  2  0  0  0  0  1  1  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  2  1  0  0  1  0  0 -1  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0
  0  0  0  0 -1  0  0 -1  0  0  0 -1  0  0 -1  0  0  1  0  0  0  0  0  0
  0  0  0  0  0  0  0  0 -1  0  0  0  1  0  1  0  1  0  1  0  0  0  1  0
  0  0  0  0 -1  0  0  0  2  0  0  0  1  0  0  0  0  0  0  0 -1  0  0  0
  0  0  0  0  0  0  0  0  1  0  0  0 -1  0  0  0  0  0  0  0  2  0  0  0
  1  0  1  0  0  1  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0 -1  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  1  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  1
  0  0  0  2  0  1  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0 -1
 -1  0  1  0  0  2  0  0  0  0  0  0  0  0  1  0 -1 -1  0  0  0  0  1  0
  0  0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  1  0  0  0  1
  0  1  0 -1  0  0  0  0 -1  0  0 -1  0  0  0  0  0  0  0  0  2  0  0  0
  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  1
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0 -1  0  0  0
  0  0  0 -1  0  0  1 -1 -1  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0
  0  0  0  1  0  0  0  0  0  0  1  0  0  0  1  0  0  1  0  0  0  0 -1  0
  0  0  0  0  0  0  0 -1  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0  0  1  0  0  0  1 -1  0
  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0 -1 -1  0  0  0  0  0  0 -1  0  0  1  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0 -1  0  0  0  0  1  0  0  1  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  1 -1  0  0  0  0  0  0 -1 -1  0  0 -1  0  0  0  0
  0  0  0  0 -1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0 -1  0  0  0  0  0  0  0  1  0  0  0  1  0  0  0 -1  0
  0  0  0  0  0  0  0  0  0  2  0  0  0  0  0  0 -1  0  0  0 -1  0  0  0
  0 -1  0  0  0  0  0  0  1  0 -1  0  0  0  0  0  0  0 -1  0  0  0  0  0
  0  1  0  0  0  0  0  0  0  0  0  1  0  0 -1  0  0  0  0  0  0  0  0  0
  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0 -1  0  0  0  0  0
  0  0 -1 -1  0  0  0  0  0  0  0  0 -1  0  0  0  0  0  1  0  0  0  0  0
  0  0  0  1  0  0  0  0 -1  0  0  0  0  0 -1  0  0  0  1  0  0  0  0  0
  0 -1  0  0  0  0 -1  0  0  0  1  0  0  0  0  0  0  0  0  0  1  0  0
  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0 -1  0  0  0 -1  0  0  0  0  1  0  0  0  0 -1  0  0  0 -1  0  0  0  0
  0  0  0  0  0  0]
```

In [99]:

```
# Building the label to colour mapping
colours = {}
colours[0] = 'r'
colours[1] = 'g'
colours[2] = 'b'
colours[-1] = 'k'

# Building the colour vector for each data point
cvec = [colours[label] for label in labels]

# For the construction of the legend of the plot
r = plt.scatter(X_principal['P1'], X_principal['P2'], color ='r');
g = plt.scatter(X_principal['P1'], X_principal['P2'], color ='g');
b = plt.scatter(X_principal['P1'], X_principal['P2'], color ='b');
k = plt.scatter(X_principal['P1'], X_principal['P2'], color ='k');

# Plotting P1 on the X-Axis and P2 on the Y-Axis
# according to the colour vector defined
plt.figure(figsize =(9, 9))
plt.scatter(X_principal['P1'], X_principal['P2'], c = cvec)

# Building the legend
plt.legend((r, g, b, k), ('Label 0', 'Label 1', 'Label 2', 'Label -1'))

plt.show()
```





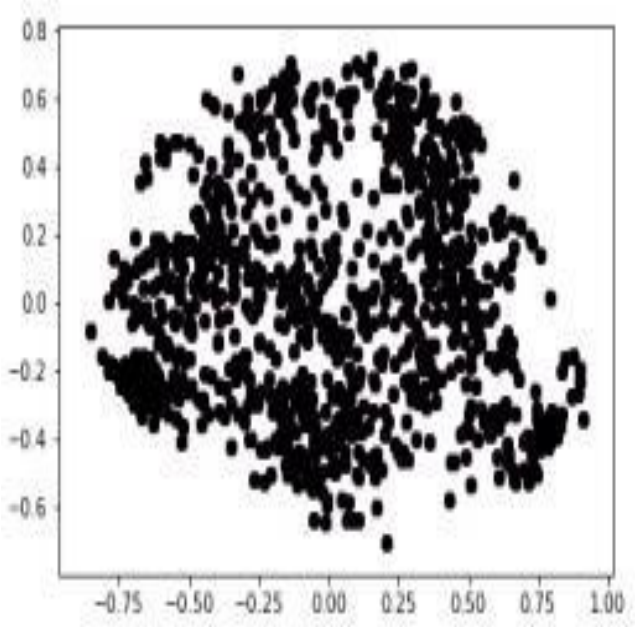
In [141]:

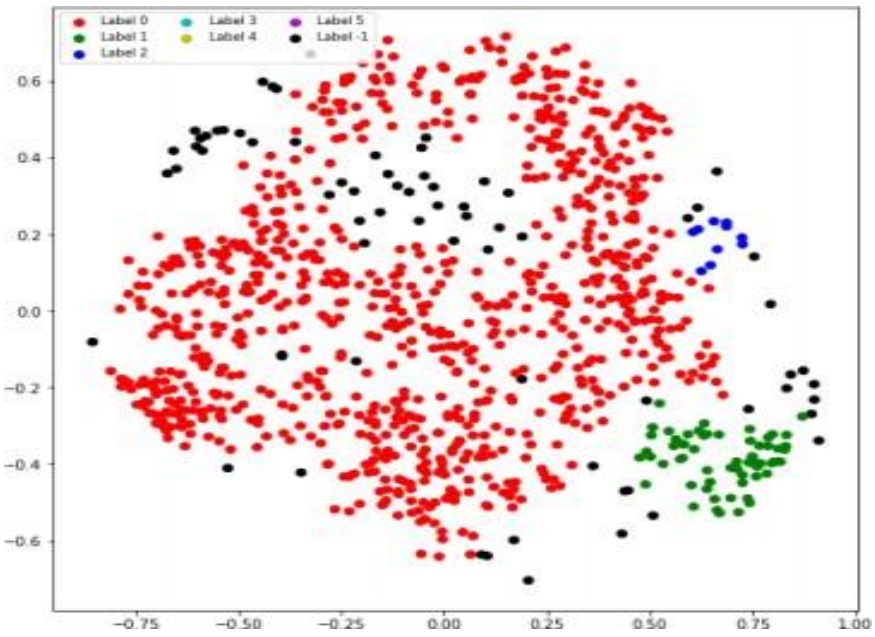
```
db = DBSCAN(eps = 0.119, min_samples = 30).fit(X_principal)
labels1 = db.labels_
labels2 = np.array(labels1)
labels2 = labels2.astype(int)
print(labels2)
```

```
[ 1  0  0  3  0  1 -1  4  0  3 -1  0 -1  1 -1  2  1  0  0  1  0  2  0  2
  2  1  1  2  1  2  1  2 -1  4  0  2  2  2  0 -1  1  0  1 -1  2  1  0 -1
  0  0  0  3  4  1  0  1 -1  2  1  0  1  4  1  0  1  1 -1  2  0  1  1  4
  2  1  1  2  0  1 -1  0  4  0  3  2  0  1  2 -1  0  0  1  0  4  0  0  1
  0  1  2  0  1  5 -1  0  4  1  2  1  2  0  2  0  1  0  4  4  3  1  1  1
  1  2  0 -1  2  1  0  0  1  1  0  1  0 -1  4  1  2  4  0  1 -1 -1  0  1
  0  1  1  2  0  1  1  1  1 -1  0  1  2  2  0  1  1  4  1  0  1  2  0  1
  1  1  2  0 -1  1  1 -1  1  2  0  1  1  0  1  1  1  0  1  1  1  1  1 -1
  1  1  0  2 -1  1 -1 -1  0 -1  1 -1  1  1 -1  1  0  4  0  0  2  2  1  0
  0  0  2  1  0  0  2  0 -1  1  0  1 -1  0  4  1 -1 -1  4 -1  0  0  4  2
  0  1  1  1 -1  2  1 -1 -1  5  0  2 -1  2  3  1 -1  1  1  0 -1  1  0  2
  1  1  0  0  1  0  2  2  4  0  2  1 -1  1  3  1 -1  1  2  2 -1  2  0  1
  4  0  4  2  1  4  1  0  1  1  0  0  0  0  0  1  0  1  1  2  0  0  1  1
  1  2  1  2 -1  1  1  0  1  2 -1  1  1  1  0  1  2  0  1  0  2  4  0  2
  2  2  2  0 -1  1  1  0  1  0  0  0  0  2  2  4  1  2  2  1  2  1  3  0
  1 -1  1 -1 -1  4  1  1  2  1  0  2 -1  1  2  1  3 -1  2  2  1  0  0 -1
 -1  2  4  2  0 -1  0  0  1  0  0  3  0  0 -1  1 -1  1  2  2  0  1  4  1
  1  0  1  1  0  2  2  1  1  4 -1  0  2  1  2  1  1  1  0  4  3 -1  0  4
  0  4  0 -1  0  0  0  2 -1  1  2 -1  0 -1  2 -1  2  2  0  2 -1  0  2  0
  1  1  2  2  1  1 -1  2  0 -1  2  0  0  0  1  1  1  3  2  1  2  1  2  4
  1  0  0  1  0  1  2  0  1  2  2 -1  2  0  1  2  0  1  1  0  1  2  1  1
  1  2  2  1  1  1  1  5 -1  0 -1  2  2  1  0 -1  1  2  3  0 -1  1  1  1
  0  1 -1  1  1  0  4  1 -1  1  0  3  1 -1  0  1  1  4  0  1  1  2  0  1
  1  2  2  4 -1  1  1  0  3  0  4  1  1  0  4  1 -1  4  1 -1  1  1 -1  2
  1  1  2  0  2  1  3 -1  4  1  2  2  1  1  1  0  1  2  3  1  1  2  1  1
  0  1  2  1  0  0  1  0 -1 -1  0  0  1  0  2  1  3  4  1  0  0  4 -1  1
  0  0 -1  1  1  2  0  0  1  1  0  0  1 -1  1  1  1  3  1  0  0  0  0  0
  0  3 -1 -1  1  2  2  1  2  2 -1  0  2  4 -1  1 -1  3  1  3  1  0  1  0
  1  3  0  3  1  0 -1  0 -1  0  1 -1  1  2 -1  0  0  2  1  1  2  2  2  0
  0  0  2  1 -1  0  1  4  0  1  2  1  3  0  1 -1 -1  1 -1 -1  1  0  0  1
  1  0  2  0  4  2  1  2  1  0  0 -1  1  0  0  2  2  0  0  2 -1  1  0  0
  5  0  1  2  0  1  2 -1  1 -1  0  0  1  1  4  1  2  1  4  2 -1 -1 -1  0
  0  1 -1  2  1  0  0  2  0 -1  1  2  2  0  0  2 -1  0  1  2 -1  2  0  1
  1 -1 -1  0  0  1  1  1  4  0 -1  0  2  1  2  2  3  0 -1  3  0 -1 -1  1
  1 -1  0  1  1  1 -1  2 -1 -1  1 -1  0  1  2  3  2 -1  3  1  1  1  2 -1
  1 -1 -1  0  4  1  2 -1  0  1  1  1  1  0  0  1  1  1 -1  0  2  0  1 -1
  1  2 -1 -1  0  2 -1  1  0  0  2  2 -1 -1  2  0  0  0  4  0  2  1 -1  0
  0  0  2  4  0  0  0  0 -1  1  0 -1  1 -1 -1  1  2  1 -1  2  2  2  1  1
  0 -1  2  3 -1  2 -1  0  0  1  4  0  1  2  1  1  1  0 -1  1 -1  4  1  2
  0  0 -1  1 -1  1  1  1  4  0  1  0  1  1  0  0  2  1  1  0  1  1  2  1
  1 -1 -1  1  0 -1  1  0  0  2  4  1  1  1 -1 -1  0 -1  0  1  1  2  2  2
  1  1  2  0 -1  1]
```

In [142]:

```
colours1 = {}
colours1[0] = 'r'
colours1[1] = 'g'
colours1[2] = 'b'
colours1[3] = 'c'
colours1[4] = 'y'
colours1[5] = 'm'
colours1[-1] = 'k'

cvec = [colours1[label] for label in labels]
colors = ['r', 'g', 'b', 'c', 'y', 'm', 'k' ]

r = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color = colors[0])
g = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color = colors[1])
b = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color = colors[2])
c = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color = colors[3])
y = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color = colors[4])
m = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color = colors[5])
k = plt.scatter(
        X_principal['P1'], X_principal['P2'], marker ='o', color = colors[6])

plt.figure(figsize =(9, 9))
plt.scatter(X_principal['P1'], X_principal['P2'], c = cvec)
plt.legend((r, g, b, c, y, m, k),
           ('Label 0', 'Label 1', 'Label 2', 'Label 3', 'Label 4','Label 5', 'Label -1'
),
           scatterpoints = 1,
           loc ='upper left',
           ncol = 3,
           fontsize = 8)
plt.show()
```





2) USING THE IMAGES

In [55]:

```
import os
import sys
import numpy as np
import pandas as pd
import PIL
from keras.preprocessing import image
from sklearn.preprocessing import LabelEncoder
from keras.utils.np_utils import to_categorical
from keras.models import Model
from keras.layers import Dense, Dropout, Activation, Convolution2D, MaxPooling2D, Flatt
en, Input
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [26]:

```
data_root = ("C:/Users/Lenovo/Desktop/DataMining/")
img_folder = data_root + 'images/'
train_data = pd.read_csv('train.csv')
train_ID = train_data['id']
train_Y = train_data['species']
test_data = pd.read_csv('test.csv')
test_ID = test_data['id']

le = LabelEncoder()
train_y = le.fit_transform(train_Y)
```

In [27]:

```
def resize_img(img, max_dim=96):
    large_axis = max((0, 1), key=lambda x: img.size[x])
    scalar = max_dim / float(img.size[large_axis])
    resized = img.resize(
        (int(img.size[0] * scalar), int(img.size[1] * scalar)))
    return resized


def load_image_data(id_list, max_dim=96, center=True):
    X = np.empty((len(id_list), max_dim, max_dim, 1))
    for i, idnum in enumerate(id_list):
        x = image.load_img(
            (img_folder + str(idnum) + '.jpg'), grayscale=True)
        x = image.img_to_array(resize_img(x, max_dim=max_dim))
        height = x.shape[0]
        width = x.shape[1]
        if center:
            h1 = int((max_dim - height) / 2)
            h2 = h1 + height
            w1 = int((max_dim - width) / 2)
            w2 = w1 + width
        else:
            h1, w1 = 0, 0
            h2, w2 = (height, width)
        X[i, h1:h2, w1:w2, :] = x
    return np.around(X / 255.0)
```

In [28]:

```
train_data
```

Out[28]:

	id	species	margin1	margin2	margin3	margin4	margin5	margin6
0	1	Acer_Opalus	0.007812	0.023438	0.023438	0.003906	0.011719	0.000766
1	2	Pterocarya_Stenoptera	0.005859	0.000000	0.031250	0.015625	0.025391	0.001953
2	3	Quercus_Hartwissiana	0.005859	0.009766	0.019531	0.007812	0.003906	0.005859
3	5	Tilia_Tomentosa	0.000000	0.003906	0.023438	0.005859	0.021484	0.019531
4	6	Quercus_Variabilis	0.005859	0.003906	0.048828	0.009766	0.013672	0.015625
...	...	...	...	...	...	...	...	...
985	1575	Magnolia_Salicifolia	0.060547	0.119140	0.007812	0.003906	0.000000	0.148440
986	1578	Acer_Pictum	0.001953	0.003906	0.021484	0.107420	0.001953	0.000000
987	1581	Alnus_Maximowiczii	0.001953	0.003906	0.000000	0.021484	0.078125	0.003906
988	1582	Quercus_Rubra	0.000000	0.000000	0.046875	0.056641	0.009766	0.000000
989	1584	Quercus_Afares	0.023438	0.019531	0.031250	0.015625	0.005859	0.019531

990 rows × 194 columns

In [33]:

```
def AlexNet(input_layer):
    conv_1 = Convolution2D(96, 11, 11, activation='relu', input_shape=(
        96, 96, 1), border_mode='same', name='conv1')(input_layer)
    max_pool_1 = MaxPooling2D((3, 3), strides=(2, 2))(conv_1)

    conv_2 = Convolution2D(256, 5, 5, border_mode='same',
                           activation='relu')(max_pool_1)
    max_pool_2 = MaxPooling2D((3, 3), strides=(2, 2))(conv_2)

    conv_3 = Convolution2D(384, 3, 3, border_mode='same',
                           activation='relu')(max_pool_2)
    conv_4 = Convolution2D(384, 3, 3, border_mode='same',
                           activation='relu')(conv_3)

    conv_5 = Convolution2D(256, 3, 3, border_mode='same',
                           activation='relu')(conv_4)
    max_pool_5 = MaxPooling2D((3, 3), strides=(2, 2))(conv_5)

    flat = Flatten()(max_pool_5)
    dense_1 = Dense(4096, init='glorot_normal', activation='relu')(flat)
    drop_1 = Dropout(0.5)(dense_1)

    dense_2 = Dense(4096, init='glorot_normal', activation='relu')(drop_1)
    drop_2 = Dropout(0.5)(dense_2)

    output_layer = Dense(99, activation='softmax')(drop_2)

    model = Model(input_layer, output_layer)
    return model
```

In [34]:

```
def NaiveCovNet(input_layer):
    x = Convolution2D(8, 5, 5, input_shape=(96, 96, 1),
                      border_mode='same')(input_layer)
    x = (Activation('relu'))(x)
    x = (MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))(x)

    # Now through the second convolutional layer
    x = (Convolution2D(32, 5, 5, border_mode='same'))(x)
    x = (Activation('relu'))(x)
    x = (MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))(x)

    # Flatten our array
    x = Flatten()(x)
    dense_1 = Dense(1024, init='glorot_normal', activation='relu')(x)
    drop_1 = Dropout(0.5)(dense_1)

    dense_2 = Dense(99, init='glorot_normal', activation='relu')(drop_1)
    drop_2 = Dropout(0.5)(dense_2)

    output_layer = Dense(99, activation='softmax')(drop_2)
    model = Model
    return model
```

In [31]:

```
input_layer = Input(shape=(96, 96, 1), name='image')
model = AlexNet(input_layer)
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])

trian_X = load_image_data(train_ID)
train_y = to_categorical(train_y)
```

```
C:\Users\Lenovo\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_laun
cher.py:3: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Con
v2D(96, (11, 11), activation="relu", input_shape=(96, 96, 1..., name="conv
1", padding="same")`
  This is separate from the ipykernel package so we can avoid doing import
s until
C:\Users\Lenovo\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_laun
cher.py:7: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Con
v2D(256, (5, 5), activation="relu", padding="same")`
  import sys
C:\Users\Lenovo\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_laun
cher.py:11: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Co
nv2D(384, (3, 3), activation="relu", padding="same")`
  # This is added back by InteractiveShellApp.init_path()
C:\Users\Lenovo\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_laun
cher.py:13: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Co
nv2D(384, (3, 3), activation="relu", padding="same")`
  del sys.path[0]
C:\Users\Lenovo\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_laun
cher.py:16: UserWarning: Update your `Conv2D` call to the Keras 2 API: `Co
nv2D(256, (3, 3), activation="relu", padding="same")`
  app.launch_new_instance()
C:\Users\Lenovo\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_laun
cher.py:20: UserWarning: Update your `Dense` call to the Keras 2 API: `Den
se(4096, activation="relu", kernel_initializer="glorot_normal")`
C:\Users\Lenovo\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_laun
cher.py:23: UserWarning: Update your `Dense` call to the Keras 2 API: `Den
se(4096, activation="relu", kernel_initializer="glorot_normal")`
C:\Users\Lenovo\Anaconda3\envs\tensorflow\lib\site-packages\keras_preproce
ssing\image\utils.py:104: UserWarning: grayscale is deprecated. Please use
color_mode = "grayscale"
  warnings.warn('grayscale is deprecated. Please use '
```

In [32]:

```
history = model.fit(trian_X, train_y, nb_epoch=10, batch_size=128)
```

```
C:\Users\Lenovo\Anaconda3\envs\tensorflow\lib\site-packages\ipykernel_laun
cher.py:1: UserWarning: The `nb_epoch` argument in `fit` has been renamed
`epochs`.
  """Entry point for launching an IPython kernel.

Epoch 1/10
990/990 [==============================] - 295s 297ms/step - loss: 5.2209
- accuracy: 0.0071
Epoch 2/10
990/990 [==============================] - 239s 241ms/step - loss: 4.5967
- accuracy: 0.0061
Epoch 3/10
990/990 [==============================] - 251s 254ms/step - loss: 4.5822
- accuracy: 0.0172
Epoch 4/10
990/990 [==============================] - 233s 235ms/step - loss: 4.3668
- accuracy: 0.0253
Epoch 5/10
990/990 [==============================] - 226s 228ms/step - loss: 3.8555
- accuracy: 0.0434
Epoch 6/10
990/990 [==============================] - 230s 232ms/step - loss: 3.3027
- accuracy: 0.1212
Epoch 7/10
990/990 [==============================] - 235s 238ms/step - loss: 2.8716
- accuracy: 0.2030
Epoch 8/10
990/990 [==============================] - 202s 204ms/step - loss: 2.5582
- accuracy: 0.2364
Epoch 9/10
990/990 [==============================] - 200s 202ms/step - loss: 2.2008
- accuracy: 0.3202
Epoch 10/10
990/990 [==============================] - 225s 227ms/step - loss: 1.9564
- accuracy: 0.3929
```