# VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELGAUM, KARNATAKA

**MINOR-PROJECT-II REPORT**

**ON**

# "AUTOMATIC RECOGNITION OF MEDICINAL PLANTS USING MACHINE LEARNING TECHNIQUES"

***Submitted in partial fulfillment of the requirement for the award of the degree of***

**BACHELOR OF ENGINEERING**
**IN**
**COMPUTER SCIENCE AND ENGINEERING**

## Submitted by


| USN | NAME |
|---|---|
| 2SD18CS105 | SMITA S HEGDE |
| 2SD18CS106 | SMRUTI DESHPANDE |
| 2SD18CS129 | PRABHA H B |
| 2SD18CS135 | T BHARGAVI |


## Under the Guidance of


**Prof. / Dr. RAMACHANDRA YADAWAD**
**Dept. of CSE, SDMCET, Dharwad**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**
**S.D.M. COLLEGE OF ENGINEERING & TECHNOLOGY,**
**DHARWAD-580002**


**2021**

## S.D.M COLLEGE OF ENGINEERING & TECHNOLOGY,

## DHARWAD –580002

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

# CERTIFICATE

*Certified that the Minor-Project-1 work and presentation entitled* **"AUTOMATIC RECOGNITION OF MEDICINAL PLANTS USING MACHINE LEARNING TECHNIQUES"** *is a bonafide work carried out by* **SMITA HEGDE (2SD18CS105)**, **SMRUTI DESHPANDE (2SD18CS106)**, **PRABHA H B (2SD18CS129)**, and **T BHARGAVI (2SD18CS135),** *students of S. D. M. College of Engineering & Technology, Dharwad, in partial fulfillment for the award of* **Bachelor of Engineering in Computer Science and Engineering of** *Visvesvaraya* ***Technological University, Belgaum****, during the year 2020-2021. It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report deposited in the department library. The Minor-Project-1 has been approved, as it satisfies the academic requirements in respect of project report prescribed for the said degree.*

----------------------- ------------------------

**Dr. / Prof. Ramchandra Yadawad** **Dr. U P Kulkarni**

Project Guide HOD-CSE

# ABSTRACT

*A fully automated method for the recognition of medicinal plants using computer vision and machine learning techniques has been presented. A large number of features were extracted from each leaf such as its length, width, perimeter, and area, number of vertices, colour, perimeter and area of hull. Several derived features were then computed from these attributes. The best results were obtained from a SVM classifier It is anticipated that a web-based or mobile computer system for the automatic recognition of the medicinal plants will help the local population to improve their knowledge on medicinal plants , help taxonomists to develop more efficient species identification techniques and will also contribute significantly in the protection of endangered species*

# Working and Methodology:

*1. Pre-processing*

*The following steps were followed for pre-processing the image:*

- *Conversion of RGB to Grayscale image*
- *Smoothing image using Guassian filter*
- *Adaptive image thresholding using Otsu's thresholding method*
- *Closing of holes using Morphological Transformation*
- *Boundary extraction using contours*

*2. Feature extraction*

*Various types of leaf features were extracted from the pre-processed image are listed below:*

- *Shape based features : physiological length, physiological width, area, perimeter, aspect ratio, rectangularity, circularity*
- *Color based features : mean and standard deviations of R,G and B channels*
- *Texture based features : contrast, correlation, inverse difference moments, entropy*

*3. Model building and testing*

*(a) Support Vector Machine Classifier was used as the model to classify the plant species*

*(b) Features were then scaled using StandardScaler*

*(c) Also parameter tuning was done to find the appropriate hyperparameters of the model using GridSearchCV*

# Table of Contents

## PROBLEM STATEMENT

Plants are considered as one of the greatest assets in the field of Indian Science of Medicine called Ayurveda. The innovation in the allopathic medicines has degraded the significance of these therapeutic plants. People failed to have their medications at their door step instead went behind the fastest cure unaware of its side effects. The reason is the lack of knowledge about identifying medicinal plants among the normal ones.

In earlier days, people were good enough to identify the medicinal aspects of the plants in curing various diseases. As the days pass it is becoming difficult for the people to identify the medicinal plants. Many are unaware of these plants.

# CHAPTER 1: INTRODUCTION

The plants which are around us play a major part in framing our ecosystem. In earlier days, people were good enough to identify the medicinal aspects of these plants in curing various diseases. These plants were the ones that normally grow in our backyards or the ones that we find along the roadsides. As the days pass it is becoming difficult for the people to identify the existence of the medicinal plants. Many are unaware of these plants. So, to identify a plant first we consider the leaves of that plant to classify them. Leaves can be classified based on various features like texture, shape and color. Image processing plays a major role in the identification of medicinal plants by extracting the features of herbal leaf and authenticating its medicinal traits. Due to the two dimensional representation of leaves, the medicinal plants are easily identified and recognized by analyzing the shape, texture, color, aspect ratio, vein structure of leaves rather than fruits, flowers etc. Since manual recognition requires expert botanist, an image processing technique and Support Vector Machine (SVM) classifier is used to recognize the medicinal plants

## CHAPTER 2: LITERATURE SURVEY

**Kayathiri M, Krishnaveni K and Ponmalar K,** "Medicinal Plant Leaf Image Classification and Analysis using Svm Classifier Kernel Function", vol.6, issue 6, June 2019.

A data set containing 760 medicinal plant leaf images of thirty different classes are pre-processed using image processing techniques, morphological shape, texture and color features of the leaf images are extracted and stored as a feature dataset to train the classifier. The feature set for input test image is created, mapped with the training feature vector for classification and the medicinal plant class and its scientific name are displayed. Finally the classification results obtained by various kernel functions are analyzed with different performance metrics.

**Venitha Kowlessur, Upasana Singh ,Sameerchand Pudaruth and Fawzi Mahomoodally,** "Automatic Recognition of Medicinal Plants using Machine Learning Techniques", vol.8, No. 4, 2017

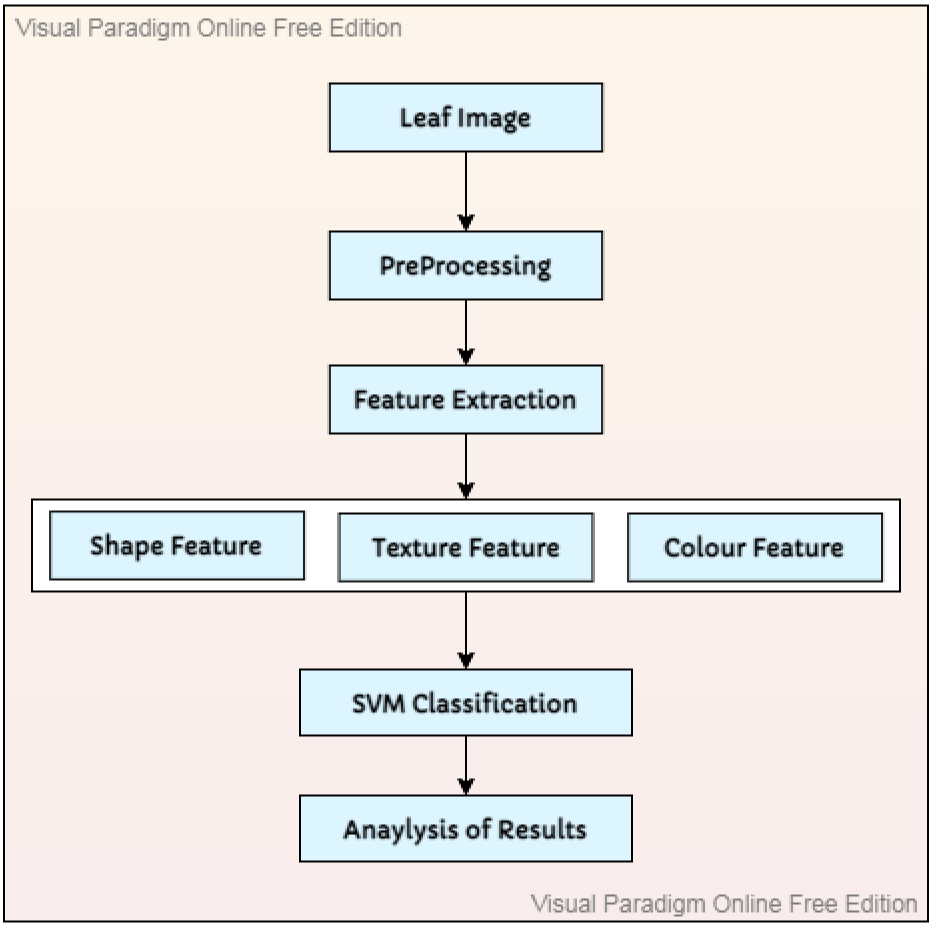
A new dataset on medicinal plants of Mauritius has been made publicly available on the machine learning repository portal. Machine learning algorithms were then used to classify the leaves from 24 different plant species into their appropriate categories. The highest accuracy of 90.1% was obtained

**D Venkataraman, Mangayarkarasi N** "Computer Vision Based Feature Extraction of Leaves for Identification of Medicinal Valus of Plant" vol . 2,2016

The features can be calculated for different types of herbal leaves and can be stored in the database which will be the trained result values.

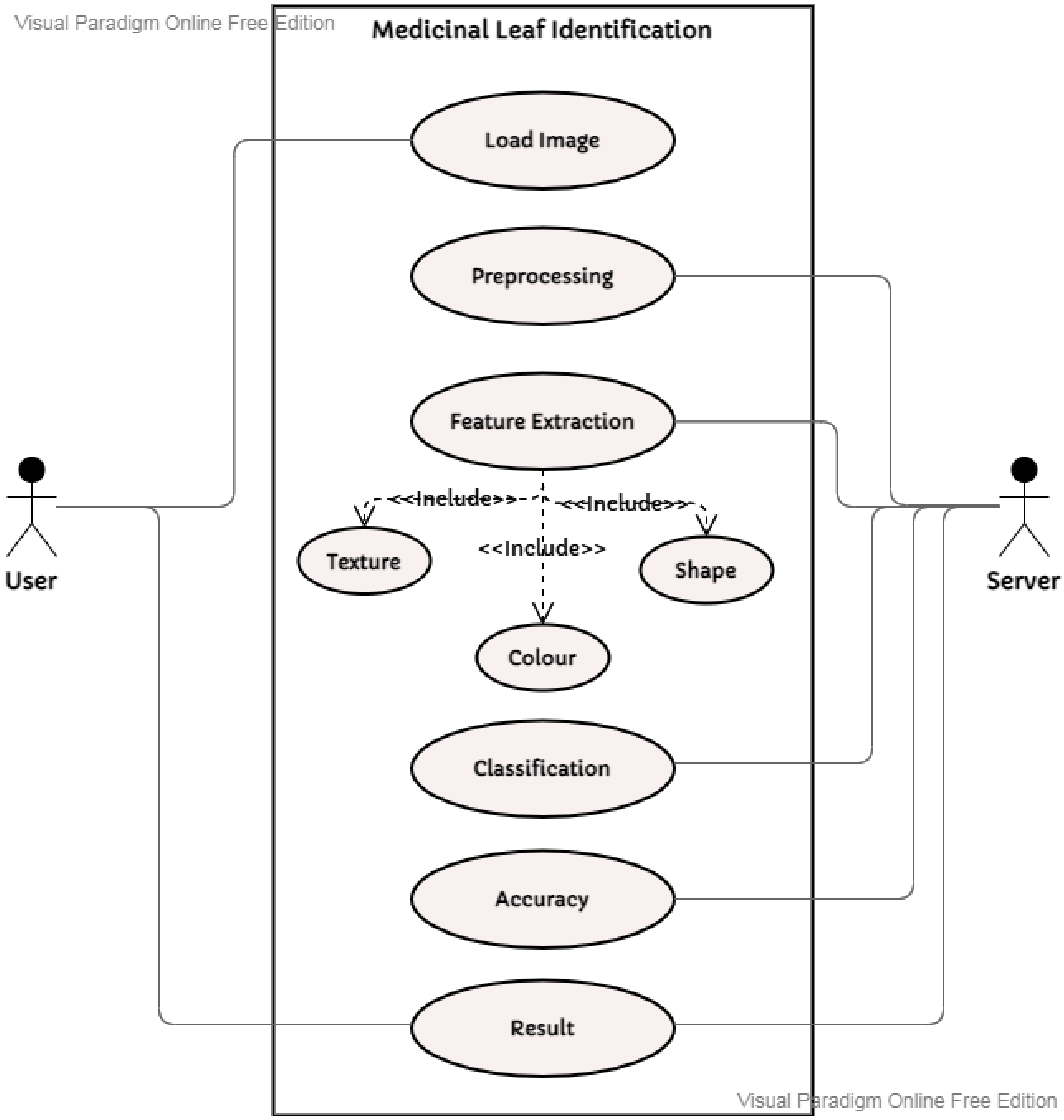
# CHAPTER 3: DETAILED DESIGN

**Schematic Diagram**




Tool Used: Visual Paradigm Online Free Edition

## Use Case Diagram




Tool Used: Visual Paradigm Online Free Edition

# CHAPTER 4: PROJECT SPECIFIC REQUIREMENTS

**HARDWARE REQUIREMENTS:**

- Processor Type : Pentium –IV and above
- Speed : 2.4 GHZ
- Ram : 128 MB RAM
- Hard disk : 20 GB HD

**SOFTWARE REQUIREMENTS:**

- Operating System : Windows 7 and above
- Software Programming Package : PYTHON

# CHAPTER 5: IMPLEMENTATION

- **Single image preprocessing and feature extraction - Testfile**

```
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline

ds_path = "Flavia leaves dataset"

test_img_path = ds_path + "\\2546.jpg"
test_img_path

main_img = cv2.imread(test_img_path)
img = cv2.cvtColor(main_img, cv2.COLOR_BGR2RGB)
plt.imshow(img)

gs = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
plt.imshow(gs,cmap='Greys_r')

gs.shape

blur = cv2.GaussianBlur(gs, (25,25),0)
plt.imshow(blur,cmap='Greys_r')

ret_otsu,im_bw_otsu = cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
plt.imshow(im_bw_otsu,cmap='Greys_r')

kernel = np.ones((50,50),np.uint8)
closing = cv2.morphologyEx(im_bw_otsu, cv2.MORPH_CLOSE, kernel)

plt.imshow(closing,cmap='Greys_r')

sobelx64f = cv2.Sobel(closing,cv2.CV_64F,1,0,ksize=5)
abs_sobel64f = np.absolute(sobelx64f)
sobel_8u = np.uint8(abs_sobel64f)
plt.imshow(abs_sobel64f,cmap='Greys_r')

ret_sobel,im_bw_sobel = cv2.threshold(sobel_8u,1,255,cv2.THRESH_BINARY)
plt.imshow(im_bw_sobel,cmap='Greys_r')

kernel_edge = np.ones((15,15),np.uint8)
closing_edge = cv2.morphologyEx(im_bw_sobel, cv2.MORPH_CLOSE, kernel_edge)
plt.imshow(closing_edge,cmap='Greys_r')

plt.imshow(closing,cmap="Greys_r")
```

```
contours, hierarchy = cv2.findContours(closing,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

len(contours)

cnt = contours[0]
len(cnt)

plottedContour = cv2.drawContours(gs,contours,-1,(0,255,0),10)
plt.imshow(plottedContour,cmap="Greys_r")

M = cv2.moments(cnt)
M

area = cv2.contourArea(cnt)
area

perimeter = cv2.arcLength(cnt,True)
perimeter

rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.int0(box)
contours_im = cv2.drawContours(closing,[box],0,(255,255,255),2)
plt.imshow(contours_im,cmap="Greys_r")

ellipse = cv2.fitEllipse(cnt)
im = cv2.ellipse(closing,ellipse,(255,255,255),2)
plt.imshow(closing,cmap="Greys_r")

x,y,w,h = cv2.boundingRect(cnt)
aspect_ratio = float(w)/h
aspect_ratio

rectangularity = w*h/area
rectangularity

circularity = ((perimeter)**2)/area
circularity

equi_diameter = np.sqrt(4*area/np.pi)
equi_diameter

(x,y),(MA,ma),angle = cv2.fitEllipse(cnt)

plt.imshow(img,cmap="Greys_r")

red_channel = img[:,:,0]
plt.imshow(red_channel,cmap="Greys_r")

green_channel = img[:,:,1]
plt.imshow(green_channel,cmap="Greys_r")

blue_channel = img[:,:,2]
plt.imshow(blue_channel,cmap="Greys_r")
```

```
np.mean(blue_channel)

blue_channel[blue_channel == 255] = 0
green_channel[green_channel == 255] = 0
red_channel[red_channel == 255] = 0

red_mean = np.mean(red_channel)
red_mean

green_mean = np.mean(green_channel)
green_mean

blue_mean = np.mean(blue_channel)
blue_mean

red_var = np.std(red_channel)
red_var

import mahotas as mt

textures = mt.features.haralick(gs)
ht_mean = textures.mean(axis=0)
ht_mean

print(ht_mean[1]) #contrast
print(ht_mean[2]) #correlation
print(ht_mean[4]) #inverse difference moments
print(ht_mean[8]) #entropy
```

- **Preprocess and Feature Extraction - Flavia dataset**

```
import os
import cv2
import numpy as np
import pandas as pd
import mahotas as mt
from matplotlib import pyplot as plt
%matplotlib inline

ds_path = "..\\Flavia leaves dataset"
img_files = os.listdir(ds_path)

def create_dataset():
names = ['area','perimeter','physiological_length','physiological_width','aspect_ratio'
          'mean_r','mean_g','mean_b','stddev_r','stddev_g','stddev_b', \
          'contrast','correlation','inverse_difference_moments','entropy'
         ]
df = pd.DataFrame([], columns=names)
        for file in img_files:
                imgpath = ds_path + "\\" + file
                main_img = cv2.imread(imgpath)
```

```
        #Preprocessing
        img = cv2.cvtColor(main_img, cv2.COLOR_BGR2RGB)
        gs = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
        blur = cv2.GaussianBlur(gs, (25,25),0)
        ret_otsu,im_bw_otsu = cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTS
        U)
        kernel = np.ones((50,50),np.uint8)
        closing = cv2.morphologyEx(im_bw_otsu, cv2.MORPH_CLOSE, kernel)

        #Shape features
        contours,image = cv2.findContours(closing,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        #print(contours[0])
        cnt = contours[0]
        M = cv2.moments(cnt)
        area = cv2.contourArea(cnt)
        perimeter = cv2.arcLength(cnt,True)
        x,y,w,h = cv2.boundingRect(cnt)
        aspect_ratio = float(w)/h
        rectangularity = w*h/area
        circularity = ((perimeter)**2)/area

        #Color features
        red_channel = img[:,:,0]
        green_channel = img[:,:,1]
        blue_channel = img[:,:,2]
        blue_channel[blue_channel == 255] = 0
        green_channel[green_channel == 255] = 0
        red_channel[red_channel == 255] = 0
        red_mean = np.mean(red_channel)
        green_mean = np.mean(green_channel)
        blue_mean = np.mean(blue_channel)
        red_std = np.std(red_channel)
        green_std = np.std(green_channel)
        blue_std = np.std(blue_channel)

        #Texture features
        textures = mt.features.haralick(gs)
        ht_mean = textures.mean(axis=0)
        contrast = ht_mean[1]
        correlation = ht_mean[2]
        inverse_diff_moments = ht_mean[4]
        entropy = ht_mean[8]

        vector = [area,perimeter,w,h,aspect_ratio,rectangularity,circularity,\
                  red_mean,green_mean,blue_mean,red_std,green_std,blue_std,\
                  contrast,correlation,inverse_diff_moments,entropy
                 ]

        df_temp = pd.DataFrame([vector],columns=names)
        df = df.append(df_temp)
        print(file)
    return df
```

```
#contours[0]
dataset = create_dataset()

dataset.shape

type(dataset)

dataset.to_csv("Flavia_features.csv")
```

- **Plant Leaf Classification**

```
import numpy as np
import pandas as pd
import os
import string

dataset = pd.read_csv("Flavia_features.csv")

dataset.head(1960)

type(dataset)

#maindir = r'Plant-Leaf-Identification'
ds_path = "..\\Flavia leaves dataset"
img_files = os.listdir(ds_path)

breakpoints=[1001,1059,1060,1122,1552,1616,1123,1194,1195,1267,1268,1323,1324,1385,1386,1437,1497,
1551,1438,1496,2001,2050,2051,2113,2114,2165,2166,2230,2231,2290,2291,2346,2347,2423,2424,2485,24
86,2546,2547,2612,2616,2675,3001,3055,3056,3110,3111,3175,3176,3229,3230,3281,3282,3334,3335,3389
,3390,3446,3447,3510,3511,3563,3566,3621,3622,3683,3684,3741,3742,3784,3785,3824]

target_list = []
	for file in img_files:
		target_num = int(file.split(".")[0])
		flag = 0
		i = 0
		for i in range(0,len(breakpoints),2):
			if((target_num >= breakpoints[i]) and (target_num <= breakpoints[i+1])):
				flag = 1
				break
		if(flag==1):
			target = int((i/2))
			target_list.append(target)

y = np.array(target_list)
y
```

```
X = dataset.iloc[:,1:]

X.head(5)

y[0:5]

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state = 142)

X_train.head(5)

y_train[0:5]

from sklearn.preprocessing import StandardScaler

sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)

X_train[0:2]

y_train[0:2]

from sklearn import svm

clf = svm.SVC()
clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)

from sklearn import metrics

metrics.accuracy_score(y_test, y_pred)

print(metrics.classification_report(y_test, y_pred))

from sklearn.model_selection import GridSearchCV

parameters = [{'kernel': ['rbf'],
               'gamma': [1e-4, 1e-3, 0.01, 0.1, 0.2, 0.5],
               'C': [1, 10, 100, 1000]},
               {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}
          ]

svm_clf = GridSearchCV(svm.SVC(decision_function_shape='ovr'), parameters, cv=5)

svm_clf.fit(X_train, y_train)
```

```
svm_clf.best_params_

means = svm_clf.cv_results_['mean_test_score']
stds = svm_clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, svm_clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))

y_pred_svm = svm_clf.predict(X_test)

metrics.accuracy_score(y_test, y_pred_svm)

print(metrics.classification_report(y_test, y_pred_svm))

from sklearn.decomposition import PCA

pca = PCA()

pca.fit(X)

var= pca.explained_variance_ratio_
var

import matplotlib.pyplot as plt
%matplotlib inline

var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)

plt.plot(var1)

import os
import cv2

def bg_sub(filename):
        test_img_path = '..\\mobile captures\\' + filename
        main_img = cv2.imread(test_img_path)
        img = cv2.cvtColor(main_img, cv2.COLOR_BGR2RGB)
        resized_image = cv2.resize(img, (1600, 1200))
        size_y,size_x,_ = img.shape
        gs = cv2.cvtColor(resized_image,cv2.COLOR_RGB2GRAY)
        blur = cv2.GaussianBlur(gs, (55,55),0)
        ret_otsu,im_bw_otsu = cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_
        kernel = np.ones((50,50),np.uint8)
        closing = cv2.morphologyEx(im_bw_otsu, cv2.MORPH_CLOSE, kernel)

        contours,hierarchy = cv2.findContours(closing,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIM

        contains = []
        y_ri,x_ri,_ = resized_image.shape
        for cc in contours:
                print(x_ri,y_ri)
                yn = cv2.pointPolygonTest(cc,(x_ri//2,y_ri//2),False)
```

```
                print(yn)
                contains.append(yn)
                print(contains)
        val = [contains.index(temp) for temp in contains if temp>-9999999999999]
        #val = [contains[0]]
        #val[0] = contains[0]
        print(val)
        index = val[0]

        black_img = np.empty([1200,1600,3],dtype=np.uint8)
        black_img.fill(0)

        cnt = contours[index]
        mask = cv2.drawContours(black_img, [cnt] , 0, (255,255,255), -1)
        print(len(mask))
        print(len(resized_image))
        maskedImg = cv2.bitwise_and(resized_image,mask)

        white_pix = [255,255,255]
        black_pix = [0,0,0]
        final_img = maskedImg
        h,w,channels = final_img.shape
        for x in range(0,w):
                for y in range(0,h):
                        channels_xy = final_img[y,x]
                        if all(channels_xy == black_pix):
                                final_img[y,x] = white_pix
        return final_img

filename = 'Test.jpg'
bg_rem_img = bg_sub(filename)

plt.imshow(bg_rem_img)

import mahotas as mt

def feature_extract(img):
        names = ['area','perimeter','pysiological_length',\
                 'pysiological_width','aspect_ratio',\
                 'rectangularity','circularity', \
                 'mean_r','mean_g','mean_b','stddev_r',\
                 'stddev_g','stddev_b',\
                 'contrast','correlation',\
                 'inverse_difference_moments','entropy'
                ]
        df = pd.DataFrame([], columns=names)

        #Preprocessing
        gs = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
        blur = cv2.GaussianBlur(gs, (25,25),0)
        ret_otsu,im_bw_otsu =
        cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
```

```
        kernel = np.ones((50,50),np.uint8)
        closing = cv2.morphologyEx(im_bw_otsu, cv2.MORPH_CLOSE, kernel)

        #Shape features
        contours,image = cv2.findContours(closing,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        cnt = contours[0]
        M = cv2.moments(cnt)
        area = cv2.contourArea(cnt)
        perimeter = cv2.arcLength(cnt,True)
        x,y,w,h = cv2.boundingRect(cnt)
        aspect_ratio = float(w)/h
        rectangularity = w*h/area
        circularity = ((perimeter)**2)/area

        #Color features
        red_channel = img[:,:,0]
        green_channel = img[:,:,1]
        blue_channel = img[:,:,2]
        blue_channel[blue_channel == 255] = 0
        green_channel[green_channel == 255] = 0
        red_channel[red_channel == 255] = 0
        red_mean = np.mean(red_channel)
        green_mean = np.mean(green_channel)
        blue_mean = np.mean(blue_channel)
        red_std = np.std(red_channel)
        green_std = np.std(green_channel)
        blue_std = np.std(blue_channel)

        #Texture features
        textures = mt.features.haralick(gs)
        ht_mean = textures.mean(axis=0)
        contrast = ht_mean[1]
        correlation = ht_mean[2]
        inverse_diff_moments = ht_mean[4]
        entropy = ht_mean[8]

        vector = [area,perimeter,w,h,aspect_ratio,rectangularity,circularity,\
                  red_mean,green_mean,blue_mean,red_std,green_std,blue_std,\
                  contrast,correlation,inverse_diff_moments,entropy
                 ]

        df_temp = pd.DataFrame([vector],columns=names)
        df = df.append(df_temp)

        return df

features_of_img = feature_extract(bg_rem_img)
features_of_img

scaled_features = sc_X.transform(features_of_img)
print(scaled_features)
# y_pred_mobile = svm_clf.predict(features_of_img)
y_pred_mobile = svm_clf.predict(scaled_features)
```

```
y_pred_mobile[0]

common_names = ['pubescent bamboo','Chinese horse chestnut','Anhui Barberry', \
                'Chinese redbud','true indigo','Japanese maple','Nanmu',\
                'castor aralia', 'Chinese cinnamon','goldenrain tree',\
                'Big-fruited Holly','Japanese cheesewood', \
                'wintersweet','camphortree','Japan Arrowwood',\
                'sweet osmanthus','deodar','ginkgo, maidenhair tree', \
                'Crape myrtle, Crepe myrtle','oleander','yew plum pine',\
                'Japanese Flowering Cherry','Glossy Privet',\
                'Chinese Toon','peach','Ford Woodlotus','trident maple',\
                'Beales barberry','southern magnolia',\
                'Canadian poplar','Chinese tulip tree','tangerine',\
                'ocimum tenuiflorum:-Tulsi','santalum Album:-Sandalwood',\
                'hibiscus Rosa-sinensis',\
                'nyctanthes arbor-tristis:-Night Flowering Jasmine'
               ]
common_names[y_pred_mobile[0]]
```

- **Image background subtraction – Testfile**

```
import os
import cv2
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline

test_img_path = 'mobile captures\\' + 'Test.jpg'

main_img = cv2.imread(test_img_path)
img = cv2.cvtColor(main_img, cv2.COLOR_BGR2RGB)
plt.imshow(img,cmap="Greys_r")

resized_image = cv2.resize(img, (1600, 1200))
plt.imshow(resized_image,cmap="Greys_r")

y,x,_ = img.shape

gs = cv2.cvtColor(resized_image,cv2.COLOR_RGB2GRAY)
plt.imshow(gs,cmap="Greys_r")

blur = cv2.GaussianBlur(gs, (55,55),0)
plt.imshow(blur,cmap="Greys_r")

ret_otsu,im_bw_otsu = cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU
plt.imshow(im_bw_otsu,cmap='Greys_r')

kernel = np.ones((50,50),np.uint8)
closing = cv2.morphologyEx(im_bw_otsu, cv2.MORPH_CLOSE, kernel)
```

```
plt.imshow(closing,cmap="Greys_r")

contours, hierarchy = cv2.findContours(closing,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)

len(contours)

def find_contour(cnts):
        contains = []
        y_ri,x_ri, _ = resized_image.shape
        for cc in cnts:
                yn = cv2.pointPolygonTest(cc,(x_ri//2,y_ri//2),False)
                contains.append(yn)

        val = [contains.index(temp) for temp in contains if temp>-9999999999]
        print(contains)
        return val[0]

black_img = np.empty([1200,1600,3],dtype=np.uint8)
black_img.fill(0)
plt.imshow(black_img,cmap="Greys_r")

index = find_contour(contours)
cnt = contours[index]
mask = cv2.drawContours(black_img, [cnt] , 0, (255,255,255), -1)
plt.imshow(mask)

maskedImg = cv2.bitwise_and(resized_image, mask)

white_pix = [255,255,255]
black_pix = [0,0,0]

final_img = maskedImg
h,w,channels = final_img.shape
for x in range(0,w):
        for y in range(0,h):
                channels_xy = final_img[y,x]
                if all(channels_xy == black_pix):
                        final_img[y,x] = white_pix

plt.imshow(final_img)
```

## CHAPTER 6: RESULTS

# Single image preprocessing and feature extraction - Testfile

This file explores the techniques to be used for preprocessing and feature extraction for the Flavia leaves dataset images.

### Importing necessary libraries

In [1]:
```
import os
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

### Reading the image

Note: 'Flavia leaves dataset' should be in the project root containing Flavia images.

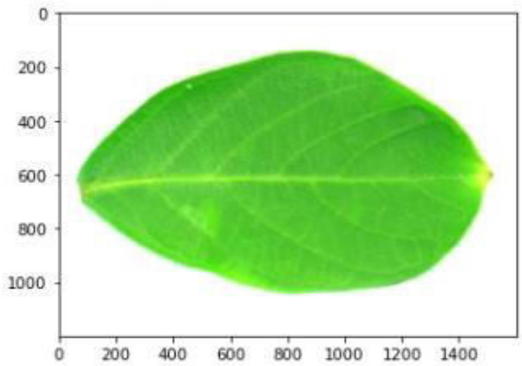
In [2]:
```
ds_path = "Flavia leaves dataset"
```

In [3]:
```
test_img_path = ds_path + "\\2546.jpg"
test_img_path
```

Out[3]:
```
'Flavia leaves dataset\\2546.jpg'
```

In [4]:
```
main_img = cv2.imread(test_img_path)
img = cv2.cvtColor(main_img, cv2.COLOR_BGR2RGB)
plt.imshow(img)
```

Out[4]:
```
<matplotlib.image.AxesImage at 0x70c6988>
```

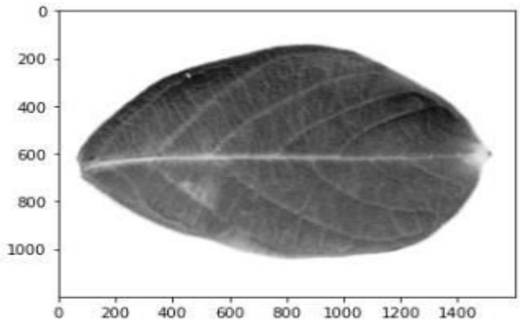



### Converting image to grayscale

In [5]:
```
gs = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
plt.imshow(gs,cmap='Greys_r')
```

```
<matplotlib.image.AxesImage at 0x7271958>
```

Out[5]:



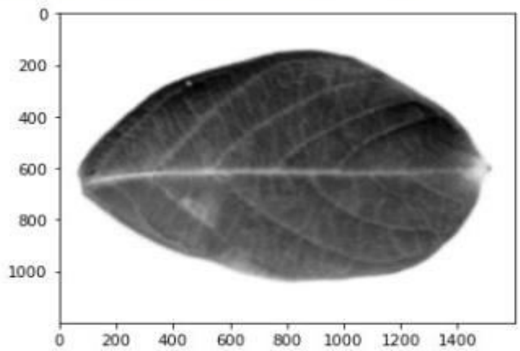

In [6]:
```
gs.shape
```

Out[6]: (1200, 1600)

### Smoothing image using Guassian filter of size (25,25)

In [7]:
```
blur = cv2.GaussianBlur(gs, (25,25),0)
plt.imshow(blur,cmap='Greys_r')
```

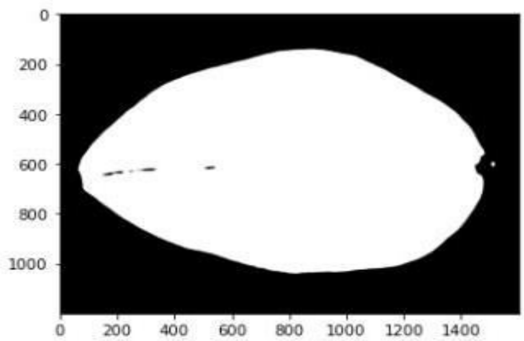
Out[7]: <matplotlib.image.AxesImage at 0x72b63d0>




### Adaptive image thresholding using Otsu's thresholding method

In [8]:
```
ret_otsu,im_bw_otsu = cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU
plt.imshow(im_bw_otsu,cmap='Greys_r')
```

Out[8]: <matplotlib.image.AxesImage at 0x72e4c88>



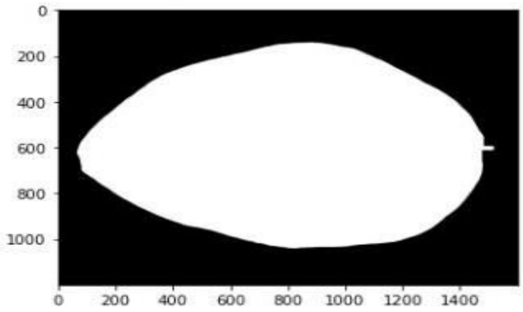

### Closing of holes using Morphological Transformation

Performed so as to close any holes present in the leaf

In [9]:
```
kernel = np.ones((50,50),np.uint8)
closing = cv2.morphologyEx(im_bw_otsu, cv2.MORPH_CLOSE, kernel)
```

In [10]:
```
plt.imshow(closing,cmap='Greys_r')
```

Out[10]: <matplotlib.image.AxesImage at 0x73843a0>



## Boundary extraction

Boundary extraction is needed which will be used in calculation of shape features.

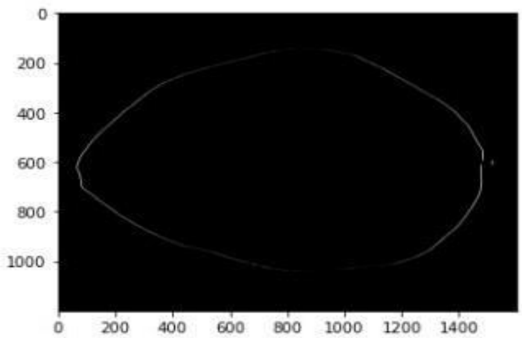
### Boundary extraction using sobel filters - Not effective

Trying to extract the boundary of the leaf using sobel filters. The image after edge extraction is thresholded using Otsu's method. Then the gaps were closed using Closing operation of Morphological Transformation.

This method is not effective as even after performing morphological transformation, gaps still persist.

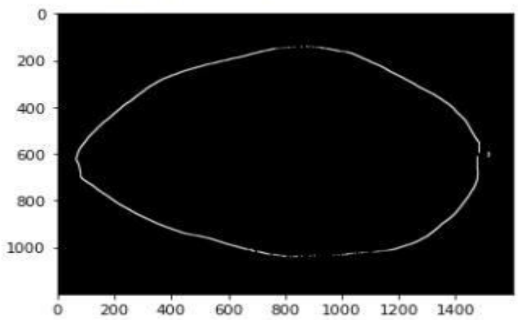
In [11]:
```
sobelx64f = cv2.Sobel(closing,cv2.CV_64F,1,0,ksize=5)
abs_sobel64f = np.absolute(sobelx64f)
sobel_8u = np.uint8(abs_sobel64f)
plt.imshow(abs_sobel64f,cmap='Greys_r')
```

Out[11]: <matplotlib.image.AxesImage at 0x7611e20>



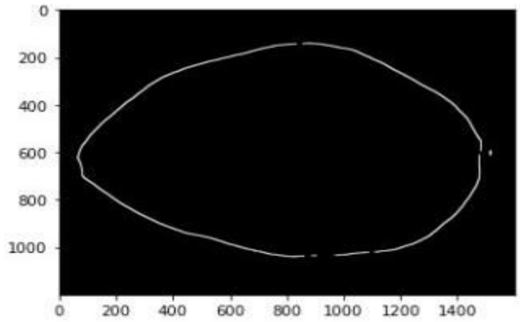
In [12]:
```
ret_sobel,im_bw_sobel = cv2.threshold(sobel_8u,1,255,cv2.THRESH_BINARY)
plt.imshow(im_bw_sobel,cmap='Greys_r')
```

Out[12]: <matplotlib.image.AxesImage at 0x79a4d48>



In [13]:
```
kernel_edge = np.ones((15,15),np.uint8)
closing_edge = cv2.morphologyEx(im_bw_sobel, cv2.MORPH_CLOSE, kernel_edge)
plt.imshow(closing_edge,cmap='Greys_r')
```

Out[13]: <matplotlib.image.AxesImage at 0x79dfe20>

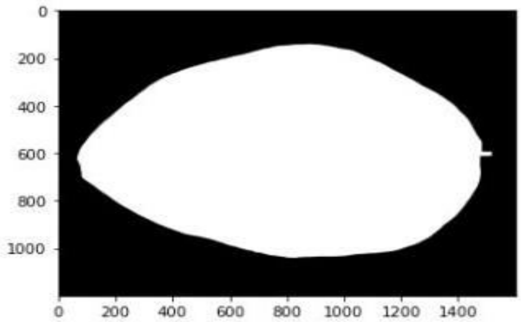



### Boundary extraction using contours - Effective

Contours are used to extract leaf boundaries. They are continous, sharp and there are no gaps between the boundary pixels

In [14]:
```
plt.imshow(closing,cmap="Greys_r")
```

Out[14]: <matplotlib.image.AxesImage at 0x8180538>




In [15]:
```
contours, hierarchy = cv2.findContours(closing,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPL
```

In [16]:
```
len(contours)
```

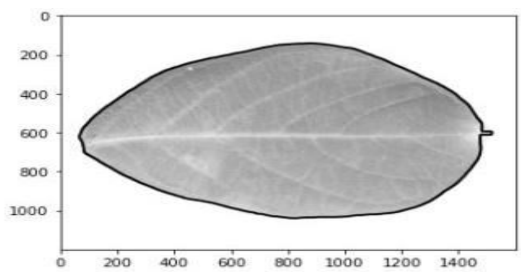
Out[16]: 1

In [17]:
```
cnt = contours[0]
len(cnt)
```

Out[17]: 1584

In [18]:
```
plottedContour = cv2.drawContours(gs,contours,-1,(0,255,0),10)
plt.imshow(plottedContour,cmap="Greys_r")
```

Out[18]: <matplotlib.image.AxesImage at 0x81b0df0>




## Morphological processing

## 1. Shape based features

### Calculating moments using contours

In [19]:
```
M = cv2.moments(cnt)
M
```

Out[19]:
```
{'m00': 935247.5,
 'm10': 756581477.6666666,
 'm01': 565084685.3333333,
 'm20': 723289661044.75,
 'm11': 460385663606.625,
 'm02': 385636777043.0833,
 'm30': 761929179121276.5,
 'm21': 443601913788318.6,
 'm12': 317109999618216.2,
 'm03': 285945100266709.94,
 'mu20': 111242548009.66907,
 'mu11': 3252545115.213257,
 'mu02': 44207688381.23468,
 'mu30': -3168469618100.5,
 'mu21': 1321617615148.461,
 'mu12': 1213331106627.2031,
 'mu03': -481339969483.8125,
 'nu20': 0.12717970576430193,
 'nu11': 0.0037185208190483455,
 'nu02': 0.050541100518092144,
 'nu30': -0.0037457014335213947,
 'nu21': 0.001562389920783421,
 'nu12': 0.0014343757754427236,
 'nu03': -0.0005690304882227447}
```

In [20]:
```
area = cv2.contourArea(cnt)
area
```

Out[20]:
```
935247.5
```

In [21]:
```
perimeter = cv2.arcLength(cnt,True)
perimeter
```

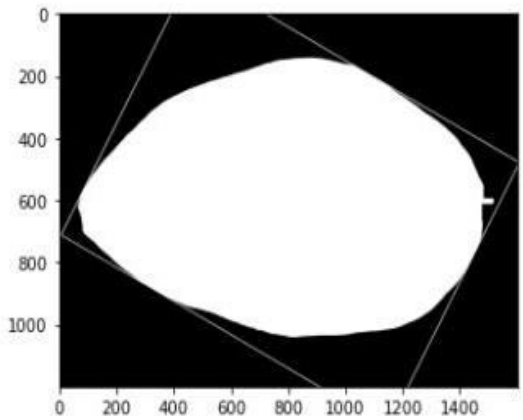
Out[21]:
```
3879.6563143730164
```

## Fitting in the best-fit rectangle and ellipse

The best-fit rectangle is chosen and not ellipse as removes (leaves out) some portion at the extreme ends of the leaf image.

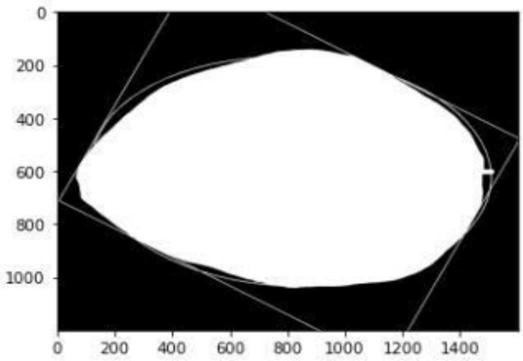
In [22]:
```
rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.int0(box)
contours_im = cv2.drawContours(closing,[box],0,(255,255,255),2)
plt.imshow(contours_im,cmap="Greys_r")
```

Out[22]:
```
<matplotlib.image.AxesImage at 0x84ed7f0>
```




In [23]:
```
ellipse = cv2.fitEllipse(cnt)
im = cv2.ellipse(closing,ellipse,(255,255,255),2)
plt.imshow(closing,cmap="Greys_r")
```

Out[23]: <matplotlib.image.AxesImage at 0x7288ee0>




### Shape based features calculated - Aspect ratio, rectangularity, circularity etc.

In [24]:
```
x,y,w,h = cv2.boundingRect(cnt)
aspect_ratio = float(w)/h
aspect_ratio
```

Out[24]: 1.6158129175946547

In [25]:
```
rectangularity = w*h/area
rectangularity
```

Out[25]: 1.3932119572626498

In [26]:
```
circularity = ((perimeter)**2)/area
circularity
```

Out[26]: 16.09385014945714

In [27]:
```
equi_diameter = np.sqrt(4*area/np.pi)
equi_diameter
```

Out[27]: 1091.2351264116726

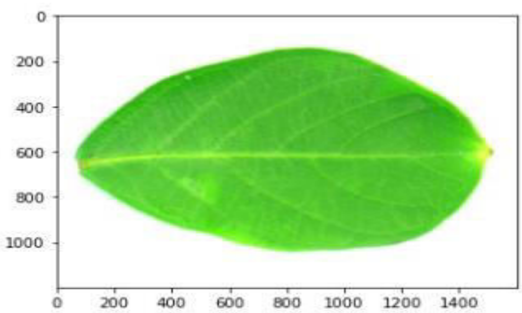
In [28]:
```
(x,y),(MA,ma),angle = cv2.fitEllipse(cnt)
```

## 2. Color based features

### Calculating color based features - mean, std-dev of the RGB channels

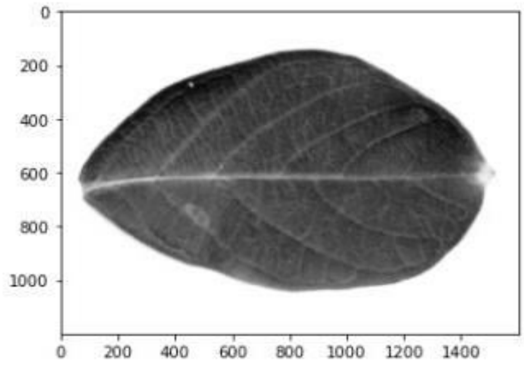
In [29]:
```
plt.imshow(img,cmap="Greys_r")
```

Out[29]: <matplotlib.image.AxesImage at 0x8e93bb0>



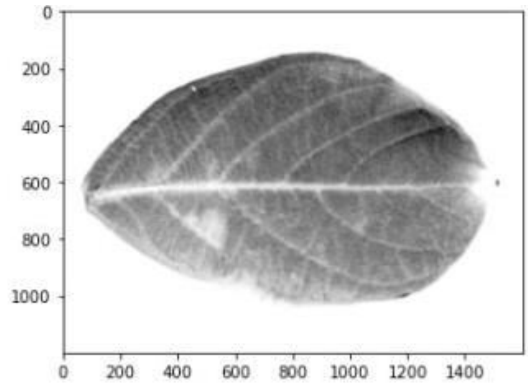

In [30]:
```
red_channel = img[:,:,0]
plt.imshow(red_channel,cmap="Greys_r")
```

Out[30]: <matplotlib.image.AxesImage at 0x8f11838>



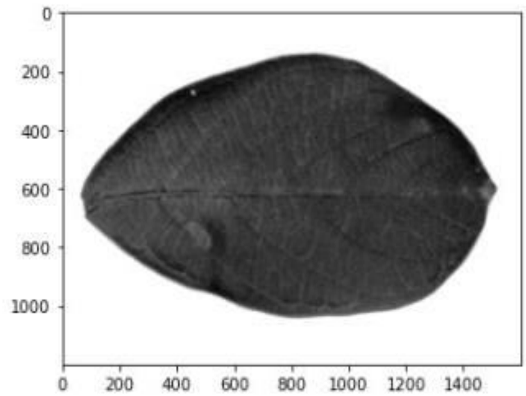

In [31]:
```
green_channel = img[:,:,1]
plt.imshow(green_channel,cmap="Greys_r")
```

Out[31]: <matplotlib.image.AxesImage at 0x8f4e070>




In [32]:
```
blue_channel = img[:,:,2]
plt.imshow(blue_channel,cmap="Greys_r")
```

Out[32]: <matplotlib.image.AxesImage at 0x8f7d970>




In [33]:
```
np.mean(blue_channel)
```

```
Out[33]: 153.04465729166665
```

```
In [34]:
blue_channel[blue_channel == 255] = 0
green_channel[green_channel == 255] = 0
red_channel[red_channel == 255] = 0
```

```
In [35]:
red_mean = np.mean(red_channel)
red_mean
```

```
Out[35]: 46.30409322916667
```

```
In [36]:
green_mean = np.mean(green_channel)
green_mean
```

```
Out[36]: 99.32073958333334
```

```
In [37]:
blue_mean = np.mean(blue_channel)
blue_mean
```

```
Out[37]: 27.54534479166667
```

```
In [38]:
red_var = np.std(red_channel)
red_var
```

```
Out[38]: 50.659233268268196
```

# 3. Texture based features

## Using Haralick moments - calculating texture based features such as contrast, correlation, entropy

```
In [39]:
import mahotas as mt
import mahotas as mt

#import mahotas.features
```

```
In [40]:
textures = mt.features.haralick(gs)
ht_mean = textures.mean(axis=0)
ht_mean
```

```
Out[40]: array([ 2.41129832e-01,  1.28949304e+02,  9.82976382e-01,  3.78732411e+03,
                 7.17433249e-01,  3.96336062e+02,  1.50203472e+04,  4.52835136e+00,
                 5.63434228e+00,  1.58917324e-03,  1.86351267e+00, -5.97338685e-01,
                 9.95789285e-01])
```

```
In [41]:
print(ht_mean[1]) #contrast
print(ht_mean[2]) #correlation
print(ht_mean[4]) #inverse difference moments
print(ht_mean[8]) #entropy
```

```
128.94930395301319
0.9829763821281734


0.7174332494974124
5.634342275636059
```

# Preprocess and Feature Extraction - Flavia dataset

Extracted features are saved in file named "Flavia_features.csv"

In [1]:

```
import os
import cv2
import numpy as np
import pandas as pd
import mahotas as mt
from matplotlib import pyplot as plt
%matplotlib inline
```

In [2]:

```
ds_path = "..\\Flavia leaves dataset"
img_files = os.listdir(ds_path)
```

In [4]:

```
def create_dataset():
    names = ['area','perimeter','physiological_length','physiological_width','aspect_ratio'
             'mean_r','mean_g','mean_b','stddev_r','stddev_g','stddev_b', \
             'contrast','correlation','inverse_difference_moments','entropy'
            ]
    df = pd.DataFrame([], columns=names)
    for file in img_files:
        imgpath = ds_path + "\\" + file
        main_img = cv2.imread(imgpath)

        #Preprocessing
        img = cv2.cvtColor(main_img, cv2.COLOR_BGR2RGB)
        gs = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
        blur = cv2.GaussianBlur(gs, (25,25),0)
        ret_otsu,im_bw_otsu = cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTS
        kernel = np.ones((50,50),np.uint8)
        closing = cv2.morphologyEx(im_bw_otsu, cv2.MORPH_CLOSE, kernel)

        #Shape features
        contours,image = cv2.findContours(closing,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        #print(contours[0])
        cnt = contours[0]
        M = cv2.moments(cnt)
        area = cv2.contourArea(cnt)
        perimeter = cv2.arcLength(cnt,True)
        x,y,w,h = cv2.boundingRect(cnt)
        aspect_ratio = float(w)/h
        rectangularity = w*h/area
        circularity = ((perimeter)**2)/area
```

```
        #Color features
        red_channel = img[:,:,0]
        green_channel = img[:,:,1]
        blue_channel = img[:,:,2]
        blue_channel[blue_channel == 255] = 0
        green_channel[green_channel == 255] = 0
        red_channel[red_channel == 255] = 0

        red_mean = np.mean(red_channel)
        green_mean = np.mean(green_channel)
        blue_mean = np.mean(blue_channel)

        red_std = np.std(red_channel)
        green_std = np.std(green_channel)
        blue_std = np.std(blue_channel)

        #Texture features
        textures = mt.features.haralick(gs)
        ht_mean = textures.mean(axis=0)
        contrast = ht_mean[1]
        correlation = ht_mean[2]
        inverse_diff_moments = ht_mean[4]
        entropy = ht_mean[8]

        vector = [area,perimeter,w,h,aspect_ratio,rectangularity,circularity,\
                  red_mean,green_mean,blue_mean,red_std,green_std,blue_std,\
                  contrast,correlation,inverse_diff_moments,entropy
                 ]

        df_temp = pd.DataFrame([vector],columns=names)
        df = df.append(df_temp)
        print(file)
    return df
```

In [5]:

```
#contours[0]
dataset = create_dataset()
```

```
3730.jpg
3731.jpg
3732.jpg
3733.jpg
3734.jpg
3735.jpg
3736.jpg
3737.jpg
3738.jpg
3739.jpg
3740.jpg
3741.jpg
3742.jpg
3743.jpg
3744.jpg
3745.jpg
3746.jpg
3747.jpg
3748.jpg
3749.jpg
```

In [6]:
```
dataset.shape
```
Out[6]:
```
(2100, 17)
```

In [7]:
```
type(dataset)
```
Out[7]:
```
pandas.core.frame.DataFrame
```

In [8]:
```
dataset.to_csv("Flavia_features.csv")
```

# Plant Leaf Classification

Applying machine learning models for classification of plant leaf images

## Importing necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import os
import string
```

## Reading the dataset

In [2]:
```
dataset = pd.read_csv("Flavia_features.csv")
```

In [3]:
```
dataset.head(1960)
```

Out[3]:

| | Unnamed: 0 | area | perimeter | physiological_length | physiological_width | aspect_ratio | rect |
|---|---|---|---|---|---|---|---|
| **0** | 0 | 197787.0 | 3479.036035 | 1416 | 759 | 1.865613 | |
| **1** | 0 | 101297.0 | 2491.210239 | 1191 | 130 | 9.161538 | |
| **2** | 0 | 86626.5 | 2291.511754 | 1096 | 119 | 9.210084 | |
| **3** | 0 | 190481.0 | 2858.479352 | 1319 | 254 | 5.192913 | |
| **4** | 0 | 228035.0 | 2920.420478 | 1325 | 286 | 4.632867 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **1955** | 0 | 496875.5 | 3071.176960 | 933 | 839 | 1.112038 | |
| **1956** | 0 | 387666.5 | 2600.868304 | 805 | 739 | 1.089310 | |
| **1957** | 0 | 409365.0 | 2567.596227 | 727 | 782 | 0.929668 | |
| **1958** | 0 | 440020.5 | 2866.240886 | 924 | 727 | 1.270977 | |
| **1959** | 0 | 804440.0 | 3750.727379 | 1313 | 962 | 1.364865 | |

1960 rows × 18 columns

```
In [4]: type(dataset)

Out[4]: pandas.core.frame.DataFrame

In [5]: #maindir = r'Plant-Leaf-Identification'
        ds_path =  "..\\Flavia leaves dataset"
        img_files = os.listdir(ds_path)
```

## Creating target labels

Breakpoints are used alongside the image file to create a vector of target labels. The breakpoints are specified in Flavia leaves dataset website.

```
In [6]: breakpoints = [1001,1059,1060,1122,1552,1616,1123,1194,1195,1267,1268,1323,1324,1385

In [7]: target_list = []
        for file in img_files:
            target_num = int(file.split(".")[0])
            flag = 0
            i = 0
            for i in range(0,len(breakpoints),2):
                if((target_num >= breakpoints[i]) and (target_num <= breakpoints[i+1])):
                    flag = 1
                    break
            if(flag==1):
                target = int((i/2))
                target_list.append(target)

In [8]: y = np.array(target_list)
        y

Out[8]: array([ 0,  0,  0, ..., 35, 35, 35])

In [9]: X = dataset.iloc[:,1:]

In [10]: X.head(5)
```

Out[10]:

| | area | perimeter | physiological_length | physiological_width | aspect_ratio | rectangularity | circı |
|---|---|---|---|---|---|---|---|
| 0 | 197787.0 | 3479.036035 | 1416 | 759 | 1.865613 | 5.433846 | 61.1 |
| 1 | 101297.0 | 2491.210239 | 1191 | 130 | 9.161538 | 1.528476 | 61.2 |
| 2 | 86626.5 | 2291.511754 | 1096 | 119 | 9.210084 | 1.505590 | 60.6 |
| 3 | 190481.0 | 2858.479352 | 1319 | 254 | 5.192913 | 1.758842 | 42.8 |
| 4 | 228035.0 | 2920.420478 | 1325 | 286 | 4.632867 | 1.661806 | 37.4 |

In [11]:
```
y[0:5]
```

Out[11]: array([0, 0, 0, 0, 0])

## Train test split

In [12]:
```
from sklearn.model_selection import train_test_split
```

In [13]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_stat
```

In [14]:
```
X_train.head(5)
```

Out[14]:

| | area | perimeter | physiological_length | physiological_width | aspect_ratio | rectangularity |
|---|---|---|---|---|---|---|
| 1033 | 60599.0 | 3468.329528 | 1360 | 894 | 1.521253 | 20.063697 |
| 1029 | 55247.5 | 3542.797555 | 1391 | 914 | 1.521882 | 23.012335 |
| 636 | 816617.5 | 3863.524283 | 1523 | 765 | 1.990850 | 1.426733 |
| 177 | 847927.5 | 3711.847224 | 1083 | 1099 | 0.985441 | 1.403678 |
| 2080 | 425087.0 | 3090.189153 | 720 | 956 | 0.753138 | 1.619245 |

In [15]:
```
y_train[0:5]
```

Out[15]: array([16, 16, 10,  3, 35])

## Feature Scaling

In [16]:
```
from sklearn.preprocessing import StandardScaler
```

In [17]:
```
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

In [18]:
```
X_train[0:2]
```

Out[18]:
```
array([[-2.0677066 , -0.29379584,  0.26994771,  0.04726532, -0.08473214,
         3.90559211,  3.92727426, -0.85270595, -1.19555962, -0.65110409,
        -1.20276985, -1.93680559, -0.64416071, -0.86213271, -0.46986853,
         2.20571733, -2.31594841],
       [-2.08802027, -0.19772303,  0.38762054,  0.13262687, -0.083729  ,
         4.57571744,  4.60789112, -0.84868006, -1.19472003, -0.64392645,
        -1.1270634 , -1.89982244, -0.53497064, -0.88361381, -0.46170211,
         2.21749523, -2.33431548]])
```

In [19]:
```
y_train[0:2]
```

Out[19]: array([16, 16])

## Applying SVM classifier model

In [20]:
```
from sklearn import svm
```

In [21]:
```
clf = svm.SVC()
clf.fit(X_train,y_train)
```

Out[21]: SVC()

In [22]:
```
y_pred = clf.predict(X_test)
```

In [23]:
```
from sklearn import metrics
```

In [24]:
```
metrics.accuracy_score(y_test, y_pred)
```

Out[24]:
```
0.8015873015873016
```

In [25]:
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.50      0.33      0.40        18
           1       0.61      0.78      0.68        18
           2       0.89      0.81      0.85        21
           3       0.81      1.00      0.90        22
           4       1.00      0.96      0.98        25
           5       0.95      1.00      0.97        18
           6       0.72      0.95      0.82        22
           7       0.93      0.76      0.84        17
           8       0.29      0.42      0.34        12
           9       0.88      0.93      0.90        15
          10       0.79      0.58      0.67        19
          11       0.90      0.60      0.72        15
          12       0.65      0.72      0.68        18
          13       0.90      0.75      0.82        24
          14       0.80      0.50      0.62        16
          15       0.50      0.54      0.52        13
          16       1.00      1.00      1.00        23
          17       0.90      0.95      0.92        19
          18       1.00      0.75      0.86        12
          19       0.90      0.95      0.93        20
          20       0.65      0.88      0.75        17
          21       0.81      0.68      0.74        19
          22       0.91      1.00      0.95        10
          23       0.72      0.87      0.79        15
          24       0.94      0.79      0.86        19
          25       0.88      0.75      0.81        20
          26       0.95      0.95      0.95        19
          27       0.71      0.83      0.77        12
          28       0.67      0.82      0.73        22
          29       0.75      0.88      0.81        17
          30       0.94      0.80      0.86        20
          31       0.76      0.87      0.81        15
          32       0.90      0.60      0.72        15
          33       0.90      0.95      0.92        19
          34       0.57      0.67      0.62        12
          35       1.00      0.83      0.91        12

    accuracy                           0.80       630
   macro avg       0.81      0.79      0.79       630
weighted avg       0.82      0.80      0.80       630
```

## Performing parameter tuning of the model

In [26]:
```
from sklearn.model_selection import GridSearchCV
```

In [27]:
```
parameters = [{'kernel': ['rbf'],
               'gamma': [1e-4, 1e-3, 0.01, 0.1, 0.2, 0.5],
               'C': [1, 10, 100, 1000]},
              {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}
             ]
```

In [28]:
```
svm_clf = GridSearchCV(svm.SVC(decision_function_shape='ovr'), parameters, cv=5)
```

```
svm_clf.fit(X_train, y_train)
```

Out[28]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid=[{'C': [1, 10, 100, 1000],
                          'gamma': [0.0001, 0.001, 0.01, 0.1, 0.2, 0.5],
                          'kernel': ['rbf']},
                         {'C': [1, 10, 100, 1000], 'kernel': ['linear']}])
```

In [29]:
```
svm_clf.best_params_
```

Out[29]:
```
{'C': 100, 'kernel': 'linear'}
```

In [30]:
```
means = svm_clf.cv_results_['mean_test_score']
stds = svm_clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, svm_clf.cv_results_['params']):
    print("%0.3f (+/-%0.03f) for %r" % (mean, std * 2, params))
```

```
0.041 (+/-0.020) for {'C': 1, 'gamma': 0.0001, 'kernel': 'rbf'}
0.188 (+/-0.036) for {'C': 1, 'gamma': 0.001, 'kernel': 'rbf'}
0.598 (+/-0.046) for {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
0.826 (+/-0.050) for {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
0.853 (+/-0.032) for {'C': 1, 'gamma': 0.2, 'kernel': 'rbf'}
0.861 (+/-0.030) for {'C': 1, 'gamma': 0.5, 'kernel': 'rbf'}
0.188 (+/-0.036) for {'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
0.598 (+/-0.050) for {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
0.829 (+/-0.047) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.891 (+/-0.024) for {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
0.887 (+/-0.032) for {'C': 10, 'gamma': 0.2, 'kernel': 'rbf'}
0.872 (+/-0.050) for {'C': 10, 'gamma': 0.5, 'kernel': 'rbf'}
0.596 (+/-0.049) for {'C': 100, 'gamma': 0.0001, 'kernel': 'rbf'}
0.819 (+/-0.069) for {'C': 100, 'gamma': 0.001, 'kernel': 'rbf'}
0.889 (+/-0.035) for {'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
0.893 (+/-0.039) for {'C': 100, 'gamma': 0.1, 'kernel': 'rbf'}
0.883 (+/-0.037) for {'C': 100, 'gamma': 0.2, 'kernel': 'rbf'}
0.871 (+/-0.041) for {'C': 100, 'gamma': 0.5, 'kernel': 'rbf'}
0.816 (+/-0.065) for {'C': 1000, 'gamma': 0.0001, 'kernel': 'rbf'}
0.886 (+/-0.036) for {'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
0.899 (+/-0.036) for {'C': 1000, 'gamma': 0.01, 'kernel': 'rbf'}
0.893 (+/-0.036) for {'C': 1000, 'gamma': 0.1, 'kernel': 'rbf'}
0.884 (+/-0.033) for {'C': 1000, 'gamma': 0.2, 'kernel': 'rbf'}
0.871 (+/-0.041) for {'C': 1000, 'gamma': 0.5, 'kernel': 'rbf'}
0.871 (+/-0.056) for {'C': 1, 'kernel': 'linear'}
0.896 (+/-0.039) for {'C': 10, 'kernel': 'linear'}
0.907 (+/-0.044) for {'C': 100, 'kernel': 'linear'}
0.901 (+/-0.041) for {'C': 1000, 'kernel': 'linear'}
```

In [31]:
```
y_pred_svm = svm_clf.predict(X_test)
```

In [32]:
```
metrics.accuracy_score(y_test, y_pred_svm)
```

Out[32]: 0.9

In [33]:
```
print(metrics.classification_report(y_test, y_pred_svm))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.84 | 0.89 | 0.86 | 18 |
| 1 | 0.83 | 0.83 | 0.83 | 18 |
| 2 | 1.00 | 0.90 | 0.95 | 21 |
| 3 | 0.96 | 1.00 | 0.98 | 22 |
| 4 | 1.00 | 1.00 | 1.00 | 25 |
| 5 | 0.95 | 1.00 | 0.97 | 18 |
| 6 | 0.83 | 0.91 | 0.87 | 22 |
| 7 | 1.00 | 0.88 | 0.94 | 17 |
| 8 | 0.50 | 0.67 | 0.57 | 12 |
| 9 | 1.00 | 1.00 | 1.00 | 15 |
| 10 | 0.89 | 0.89 | 0.89 | 19 |
| 11 | 0.73 | 0.73 | 0.73 | 15 |
| 12 | 0.87 | 0.72 | 0.79 | 18 |
| 13 | 0.71 | 0.83 | 0.77 | 24 |
| 14 | 0.83 | 0.94 | 0.88 | 16 |
| 15 | 0.78 | 0.54 | 0.64 | 13 |
| 16 | 1.00 | 1.00 | 1.00 | 23 |
| 17 | 0.90 | 1.00 | 0.95 | 19 |
| 18 | 0.92 | 1.00 | 0.96 | 12 |
| 19 | 0.95 | 1.00 | 0.98 | 20 |
| 20 | 1.00 | 0.88 | 0.94 | 17 |
| 21 | 0.88 | 0.79 | 0.83 | 19 |
| 22 | 0.91 | 1.00 | 0.95 | 10 |
| 23 | 0.87 | 0.87 | 0.87 | 15 |
| 24 | 0.86 | 0.95 | 0.90 | 19 |
| 25 | 1.00 | 0.85 | 0.92 | 20 |
| 26 | 0.90 | 1.00 | 0.95 | 19 |
| 27 | 0.91 | 0.83 | 0.87 | 12 |
| 28 | 0.91 | 0.91 | 0.91 | 22 |
| 29 | 0.89 | 0.94 | 0.91 | 17 |
| 30 | 1.00 | 0.90 | 0.95 | 20 |
| 31 | 0.87 | 0.87 | 0.87 | 15 |
| 32 | 1.00 | 0.93 | 0.97 | 15 |
| 33 | 1.00 | 1.00 | 1.00 | 19 |
| 34 | 0.91 | 0.83 | 0.87 | 12 |
| 35 | 1.00 | 0.83 | 0.91 | 12 |
| accuracy | | | 0.90 | 630 |
| macro avg | 0.90 | 0.89 | 0.89 | 630 |
| weighted avg | 0.91 | 0.90 | 0.90 | 630 |

### Dimensionality Reduction using PCA

In [34]:
```
from sklearn.decomposition import PCA
```

In [35]:
```
pca = PCA()
```

In [36]:
```
pca.fit(X)
```

Out[36]: PCA()

In [37]:
```
var= pca.explained_variance_ratio_
var
```

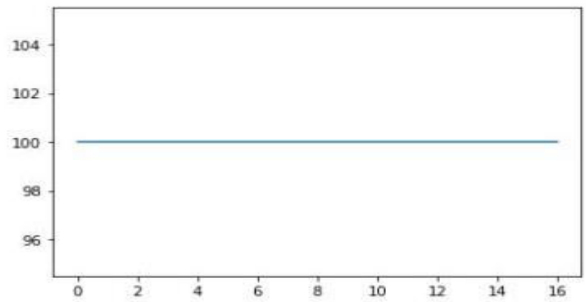
Out[37]:
```
array([9.99992628e-01, 6.56201418e-06, 5.08348962e-07, 2.15851091e-07,
       5.90241634e-08, 1.32846295e-08, 9.12039062e-09, 3.29323627e-09,
       8.99334259e-10, 2.05826615e-10, 1.83429601e-10, 1.43969145e-11,
       2.69007806e-12, 1.49779609e-12, 5.56715258e-13, 4.03320280e-15,
       1.02879613e-17])
```

In [38]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [39]:
```
var1=np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
```

```
plt.plot(var1)
```

Out[39]: [<matplotlib.lines.Line2D at 0x9cc1688>]




### Testing with mobile captured leaves which are not classified

In [40]:
```
import os
import cv2
```

In [41]:
```
def bg_sub(filename):
    test_img_path = '..\\mobile captures\\' + filename
    main_img = cv2.imread(test_img_path)
    img = cv2.cvtColor(main_img, cv2.COLOR_BGR2RGB)
    resized_image = cv2.resize(img, (1600, 1200))
    size_y,size_x,_ = img.shape
    gs = cv2.cvtColor(resized_image,cv2.COLOR_RGB2GRAY)
    blur = cv2.GaussianBlur(gs, (55,55),0)
    ret_otsu,im_bw_otsu = cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_
    kernel = np.ones((50,50),np.uint8)
    closing = cv2.morphologyEx(im_bw_otsu, cv2.MORPH_CLOSE, kernel)

    contours,hierarchy = cv2.findContours(closing,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIM

    contains = []
    y_ri,x_ri,_ = resized_image.shape
    for cc in contours:
        print(x_ri,y_ri)
        yn = cv2.pointPolygonTest(cc,(x_ri//2,y_ri//2),False)
        print(yn)
        contains.append(yn)

        print(contains)
    val = [contains.index(temp) for temp in contains if temp>-9999999999999]
    #val = [contains[0]]
    #val[0] = contains[0]
    print(val)
    index = val[0]

    black_img = np.empty([1200,1600,3],dtype=np.uint8)
    black_img.fill(0)

    cnt = contours[index]
    mask = cv2.drawContours(black_img, [cnt] , 0, (255,255,255), -1)
    print(len(mask))
    print(len(resized_image))
    maskedImg = cv2.bitwise_and(resized_image,mask)
```

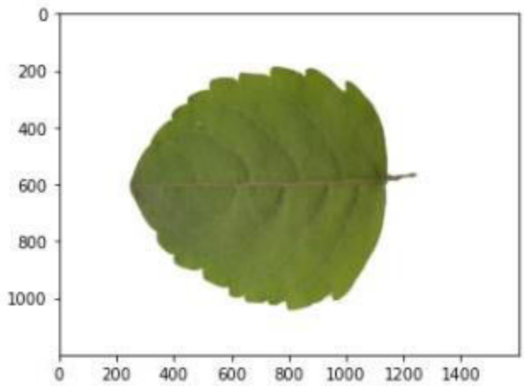
```
    white_pix = [255,255,255]
    black_pix = [0,0,0]

    final_img = maskedImg
    h,w,channels = final_img.shape
    for x in range(0,w):
        for y in range(0,h):
            channels_xy = final_img[y,x]
            if all(channels_xy == black_pix):
                final_img[y,x] = white_pix

    return final_img
```

In [42]:
```
filename = 'Test.jpg'
bg_rem_img = bg_sub(filename)
```

```
1600 1200
1.0
[1.0]
[0]
1200
1200
```

In [43]:
```
plt.imshow(bg_rem_img)
```

Out[43]: <matplotlib.image.AxesImage at 0x9dde910>




In [44]:
```
import mahotas as mt
```

In [45]:
```
def feature_extract(img):
    names = ['area','perimeter','pysiological_length',\
             'pysiological_width','aspect_ratio',\
             'rectangularity','circularity', \
             'mean_r','mean_g','mean_b','stddev_r',\
             'stddev_g','stddev_b',\
             'contrast','correlation',\
             'inverse_difference_moments','entropy'
            ]
    df = pd.DataFrame([], columns=names)

    #Preprocessing
    gs = cv2.cvtColor(img,cv2.COLOR_RGB2GRAY)
    blur = cv2.GaussianBlur(gs, (25,25),0)
    ret_otsu,im_bw_otsu = cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_
```

```
    kernel = np.ones((50,50),np.uint8)
    closing = cv2.morphologyEx(im_bw_otsu, cv2.MORPH_CLOSE, kernel)

    #Shape features
    contours,image = cv2.findContours(closing,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
    cnt = contours[0]
    M = cv2.moments(cnt)
    area = cv2.contourArea(cnt)
    perimeter = cv2.arcLength(cnt,True)
    x,y,w,h = cv2.boundingRect(cnt)
    aspect_ratio = float(w)/h
    rectangularity = w*h/area
    circularity = ((perimeter)**2)/area

    #Color features
    red_channel = img[:,:,0]
    green_channel = img[:,:,1]
    blue_channel = img[:,:,2]
    blue_channel[blue_channel == 255] = 0
    green_channel[green_channel == 255] = 0
    red_channel[red_channel == 255] = 0

    red_mean = np.mean(red_channel)
    green_mean = np.mean(green_channel)
    blue_mean = np.mean(blue_channel)

    red_std = np.std(red_channel)
    green_std = np.std(green_channel)
    blue_std = np.std(blue_channel)

    #Texture features
    textures = mt.features.haralick(gs)
    ht_mean = textures.mean(axis=0)
    contrast = ht_mean[1]
    correlation = ht_mean[2]
    inverse_diff_moments = ht_mean[4]
    entropy = ht_mean[8]

    vector = [area,perimeter,w,h,aspect_ratio,rectangularity,circularity,\
              red_mean,green_mean,blue_mean,red_std,green_std,blue_std,\
              contrast,correlation,inverse_diff_moments,entropy
             ]

    df_temp = pd.DataFrame([vector],columns=names)
    df = df.append(df_temp)

    return df
```

In [46]:
```
features_of_img = feature_extract(bg_rem_img)
features_of_img
```

Out[46]:

| | area | perimeter | pysiological_length | pysiological_width | aspect_ratio | rectangularity | circula |
|---|---|---|---|---|---|---|---|
| 0 | 578918.0 | 3191.076453 | 997 | 855 | 1.166082 | 1.472462 | 17.589 |

In [47]:
```
scaled_features = sc_X.transform(features_of_img)
print(scaled_features)
# y_pred_mobile = svm_clf.predict(features_of_img)
y_pred_mobile = svm_clf.predict(scaled_features)
y_pred_mobile[0]
```

```
[[-0.10022783 -0.65148599 -1.10796313 -0.11918968 -0.65112455 -0.31956456
  -0.36637274 -0.09775246 -0.50639069 -0.32426586  0.23771225 -0.6916865
  -0.63293388 -0.24650607  0.42989084  1.04866252 -0.69766815]]
```

Out[47]: 32

In [48]:

```
common_names = ['pubescent bamboo','Chinese horse chestnut','Anhui Barberry', \
                'Chinese redbud','true indigo','Japanese maple','Nanmu',\
                'castor aralia', 'Chinese cinnamon','goldenrain tree',\
                'Big-fruited Holly','Japanese cheesewood', \
                'wintersweet','camphortree','Japan Arrowwood',\
                'sweet osmanthus','deodar','ginkgo, maidenhair tree', \
                'Crape myrtle, Crepe myrtle','oleander','yew plum pine',\
                'Japanese Flowering Cherry','Glossy Privet',\
                'Chinese Toon','peach','Ford Woodlotus','trident maple',\
                'Beales barberry','southern magnolia',\
                'Canadian poplar','Chinese tulip tree','tangerine',\
                'ocimum tenuiflorum:-Tulsi','santalum Album:-Sandalwood',\
                'hibiscus Rosa-sinensis',\
                'nyctanthes arbor-tristis:-Night Flowering Jasmine'
               ]
common_names[y_pred_mobile[0]]
```

Out[48]: 'ocimum tenuiflorum:-Tulsi'

## Image background subtraction - Testfile

This file explores the method of background subtraction of plant leaf images captured from mobile camera. Background subtracted images will then be treated as input images to the plant leaf identification system.

### Importing necessary libraries

In [1]:

```
import os
import cv2
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
%matplotlib inline
```

### Reading the image

Note: 'mobile captures' folder must be in the project root

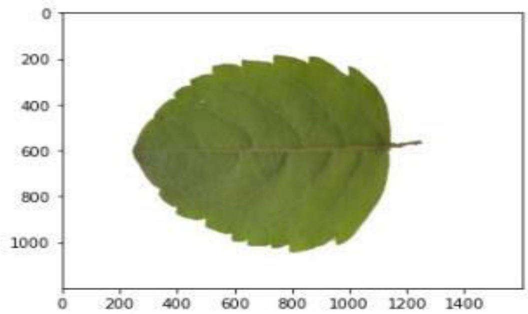
In [2]:

```
test_img_path = 'mobile captures\\' + 'Test.jpg'
```

In [3]:

```
main_img = cv2.imread(test_img_path)
img = cv2.cvtColor(main_img, cv2.COLOR_BGR2RGB)
plt.imshow(img,cmap="Greys_r")
```

Out[3]: <matplotlib.image.AxesImage at 0x897b220>



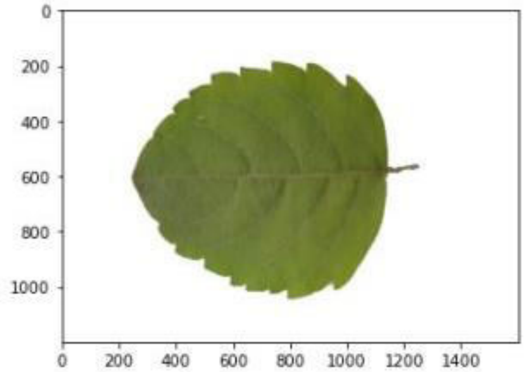

### Resizing the image to (1600,1200) - Optional

This is done as all the images in the flavia dataset were of size (1600,1200)

In [4]:

```
resized_image = cv2.resize(img, (1600, 1200))
plt.imshow(resized_image,cmap="Greys_r")
```

Out[4]: <matplotlib.image.AxesImage at 0x8aef910>

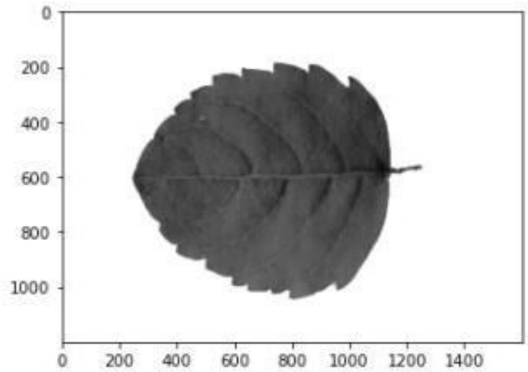


In [5]:
```
y,x,_ = img.shape
```

## Converting image to grayscale

In [6]:
```
gs = cv2.cvtColor(resized_image,cv2.COLOR_RGB2GRAY)
plt.imshow(gs,cmap="Greys_r")
```

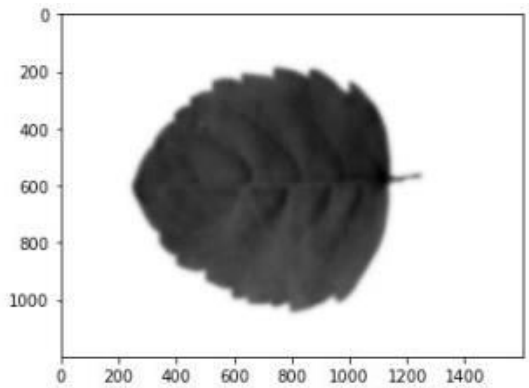
Out[6]: <matplotlib.image.AxesImage at 0x8b2f2f8>



## Smoothing image using Guassian filter of size (55,55)

In [7]:
```
blur = cv2.GaussianBlur(gs, (55,55),0)
plt.imshow(blur,cmap="Greys_r")
```

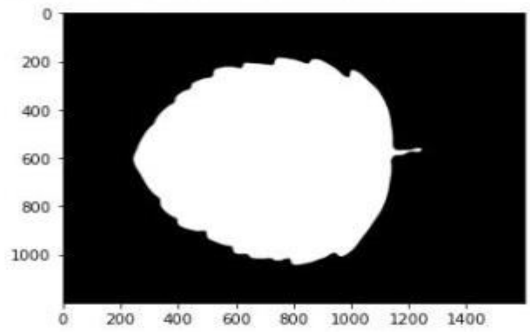
Out[7]: <matplotlib.image.AxesImage at 0x8b5fb80>



## Adaptive image thresholding using Otsu's thresholding method

In [8]:
```
ret_otsu,im_bw_otsu = cv2.threshold(blur,0,255,cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU
plt.imshow(im_bw_otsu,cmap='Greys_r')
```

Out[8]: <matplotlib.image.AxesImage at 0x8b44358>

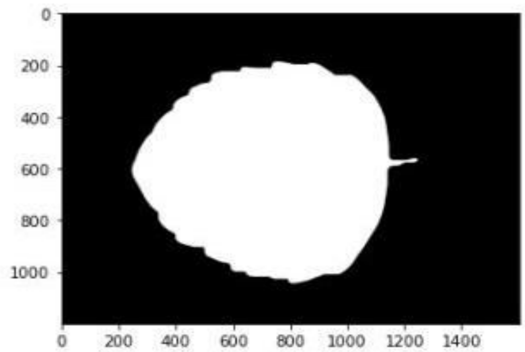


### Closing of holes using Morphological Transformation

Performed so as to close any holes present in the leaf

In [9]:
```
kernel = np.ones((50,50),np.uint8)
closing = cv2.morphologyEx(im_bw_otsu, cv2.MORPH_CLOSE, kernel)
plt.imshow(closing,cmap="Greys_r")
```

Out[9]: <matplotlib.image.AxesImage at 0x9024850>



### Finding contours

In [10]:
```
contours, hierarchy = cv2.findContours(closing,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE
```

In [11]:
```
len(contours)
```

Out[11]: 1

### Finding the correct leaf contour from the list of contours

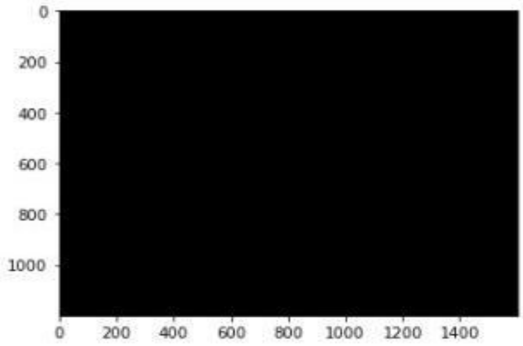
The following function finds the correct leaf contour by taking any coordinate point of the leaf (default - center point) and checks whether the current contour contains that point or not. Returns the index of the correct contour.

In [12]:
```
def find_contour(cnts):
    contains = []
    y_ri,x_ri, _ = resized_image.shape
    for cc in cnts:
        yn = cv2.pointPolygonTest(cc,(x_ri//2,y_ri//2),False)
        contains.append(yn)

    val = [contains.index(temp) for temp in contains if temp>-9999999999]
    print(contains)
    return val[0]
```

### Creating mask image for background subtraction using leaf contour

In [13]:
```
black_img = np.empty([1200,1600,3],dtype=np.uint8)
black_img.fill(0)
plt.imshow(black_img,cmap="Greys_r")
```

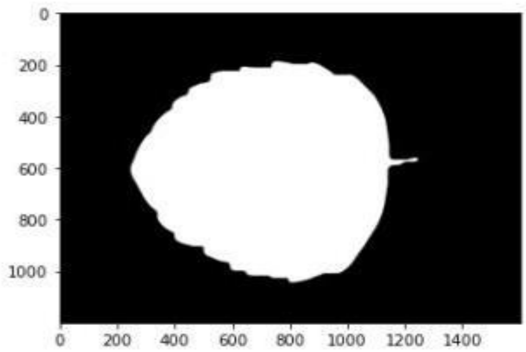
Out[13]: <matplotlib.image.AxesImage at 0x90c3940>



In [14]:
```
index = find_contour(contours)
cnt = contours[index]
mask = cv2.drawContours(black_img, [cnt] , 0, (255,255,255), -1)
plt.imshow(mask)
```

```
[1.0]
```

Out[14]: `<matplotlib.image.AxesImage at 0x91012b0>`



### Performing masking operation on the original image

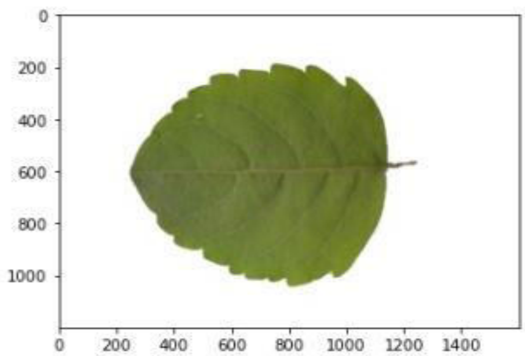
In [15]:
```
maskedImg = cv2.bitwise_and(resized_image, mask)
```

In [16]:
```
white_pix = [255,255,255]
black_pix = [0,0,0]

final_img = maskedImg
h,w,channels = final_img.shape
for x in range(0,w):
    for y in range(0,h):
        channels_xy = final_img[y,x]
        if all(channels_xy == black_pix):
            final_img[y,x] = white_pix
```

### Background subtracted image

In [17]:
```
plt.imshow(final_img)
```

Out[17]: `<matplotlib.image.AxesImage at 0x91376d0>`



# CHAPTER 7: CONCLUSION and FUTURE SCOPE

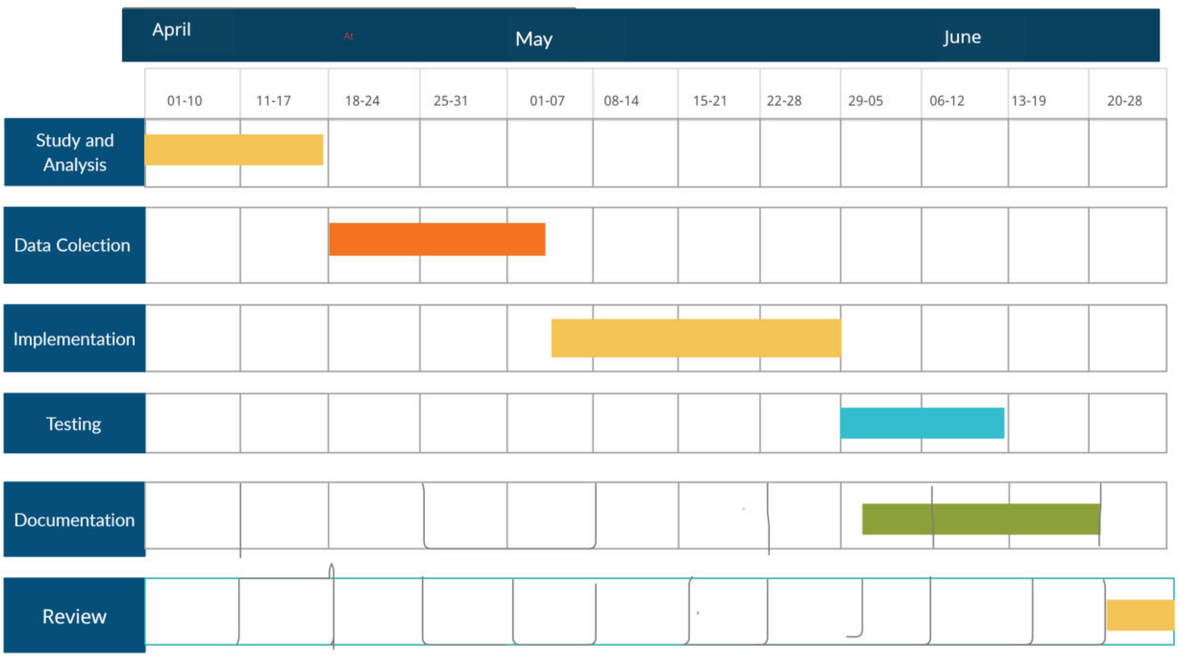
## Conclusion

Medicinal plant leaf image classification and analysis using Support Vector Machine classifier with various kernel functions is proposed .A data set containing 2100 images of plant leaf are pre-processed using image processing techniques, morphological shape, texture and color features of the leaf images are extracted and stored as a feature dataset to train the classifier. Finally the classification results obtained by various kernel functions are analyzed with different performance metrics. The experimental results show that the SVM classifier with the kernel functions yielding 90%. for the combination of shape, texture and color features.

## Future Scope

Further analyses can be conducted to improve the current feature extraction process and to include additional features such as medicinal values are displayed to the user.

## Modules and Requirements Completed are



## REFERENCES

[1]  Kayathiri M, Krishnaveni K and  Ponmalar K, “Medicinal Plant Leaf Image Classification and Analys using Svm Classifier Kernel Function”, vol.6,  pp.1-4,  June 2019.

[2] Venitha Kowlessur, Upasana Singh  ,Sameerchand Pudaruth and Fawzi Mahomoodally, “Automatic Recognition of Medicinal Plants using Machine Learning Techniques”, vol.8, pp.3-5 ,2017

[3] D Venkataraman, Mangayarkarasi N “Computer Vision Based Feature Extraction of Leaves for Identification of Medicinal Valus of Plant” vol . 2,  pp. 2-3,2016

[4] ShreyaRao, “Leaf-Classification-Using-Machine-Learning”, 20 May 2018.

[5]Aayush, “Plant Leaf Identification ”, 3 aug 2020 .