CO416 DATA WAREHOUSING AND DATA MINING

A Report on the project Titled

SLIC_SVM based leaf diseases saliency map extraction of tea plant.

SUBMITTED BY

Namrata Ladda - 16CO121

Polkampally Keerthana - 16CO131

SUBMITTED TO

Prof. M. Venkatesan

28th November 2019.

Department of Computer Science and Engineering

National Institute of Technology Karnataka, Surathkal

2019-2020.

Contents:

1. Introduction	3
2. Background	4
2. Algorithm	6
3. Data Description	7
4. Step by step implementation with data	8
5. Implementation using tools (screenshots)	12
6. Improvement	19
7. References	20

Introduction:

Tea starts as the plant known as Camellia sinensis. It's how the tea leaves are handled that gives us the various teas and their particular taste, color, and aroma. Tea leaves are small to medium in size and oval to elliptical, averaging 5-10 centimeters long. The shiny dark green leaves develop in an alternate pattern, are leathery in texture, and have serrated edges that decrease to a pointed tip. There is a focal, light green veins running the length of the leaf. The leaves develop on a thick, dark fibrous stem. Tea leaves are bitter with natural, green notes and may offer a tannic mouthfeel when soaks. The tea plantation needs a moderately hot climate with decent humidity. The climatic condition significantly affects crop dissemination. It likewise impacts the quality and amount of the yield.

These tea leaves are affected by different diseases named Tea anthracnose, Tea brown blight, Tea netted blister blight, Exobasidium vexans Massee, Pestalotiopsis theae etc. These diseases cause a huge loss to the tree plantations. Three diseases are discussed here, namely, Tea Leaf Blight, Tea Red Leaf Spot and Tea Red Scab.

In Tea Leaf Blight, small, pinhole-sized spots are at first observed on new leaves less than a month old. As the leaves grow, the spots become transparent, bigger, and light brown. After around seven days, the lower leaf surface creates blister-like symptoms, with dark green, water-drenched zones surrounding the blisters. Following the arrival of the parasitic spores, the blister gets white and smooth. In this way, the blister turns dark-colored, and newly infected stems become bent and twisted and may sever or bite the dust.

In Tea Red Leaf Spot, leaves develop injuries that are generally round, raised, and purple to reddish-brown colored. The alga may spread from leaves to branches and fruit. Most algal places are developed on the upper leaf surface. More established infections become greenish-gray and look like lichen. Cephaleuros usually doesn't hurt the plant.

In Tea Red Scab, little, red-purple injuries show up on leaves around ten days after infection — a pale, green beige form in the middle of the red injury. Lesions

on stems are pale pink, yet stems may get dark red if the sores progress. In seriously affected plants, injuries may blend, and leaves are supported and murdered off, leaving defoliated twigs at the tips of plants. In severe cases, plants might be hindered, and the loss of leaf biomass brings about a lower oil yield.

Recognize tea plant leaf diseased precisely is a critical undertaking during tea development. Machine vision is the most widely recognized disease detection technique

Background:

Matthew J have extracted crop diseases saliency map by edge detection. Many scientists have been enlivened by the accomplishment of profound learning in computer vision to improve the performance of identification frameworks for plant diseases. Lamentably, the greater part of these examinations didn't use late profound designs and depended basically on AlexNet, GoogleNet or comparative structures. Also, the examination didn't exploit profound learning perception strategies which qualifies these profound classifiers as secret elements as they are not transparent. In [1] numerous cutting edge Convolutional Neural Network (CNN) models utilizing three learning methodologies on an open dataset for plant sicknesses characterization are tried. These new structures beat the cutting edge consequences of plant sicknesses characterization with a precision arriving at 99.76%. Besides, we have proposed the utilization of saliency maps as a representation technique to comprehend and translate the CNN grouping instrument. This representation technique expands the straightforwardness of profound learning models and gives more understanding into the manifestations of plant illnesses. However, the problems of edge discontinuity, edge information loss or edge blur etc. in the division of complex images edge is common, it is difficult to ensure the integrity of boundary information.

Based on the principle of pixel color similarity in the same area, determine a seed area firstly. Then the target area is continuously merged with similar pixels in the

seed area to realize target segmentation is the process of the region-based segmentation algorithm. Wheat diseases are harmful to wheat production. However, a few segmentation algorithms that can viably recognize some diseases of wheat leaves. [2] proposes a programmed and proficient solution with K-means clustering. Right off the bat, the color image is changed to Lab color space from RGB. Grouping is then done by taking the total absolute difference between every pixel and the grouping center in Lab color space. In contrast to traditional methodologies, this method needn't bother with the manual setting for threshold value and isn't influenced by the selected channel. The outcomes show that the segmentation accuracy rates for three common diseases are over 90%, which demonstrates the adequacy of the method. However, if you want to obtain good segmentation results, you should combine this method with other algorithms, but the complexity also increased. For this very reason these methods are not suitable for extraction of diseases from tea leaves.

As we know, it helps in significantly reducing the complexity of image processing when grouping pixels based on similarity among pixel features, then superpixel blocks are used to represent lots of pixels. Usually, this method is used as a preprocessing step of the segmentation algorithm. Computer vision applications have come to depend progressively on superpixels in recent years. With an end goal to comprehend the advantages and disadvantages of existing methods, [2] compares at five state-of-the-art superpixel algorithms for their capacity to stick to image boundaries, speed, memory efficiency, and their effect on segmentation performance. It then presents another superpixel algorithm, simple linear iterative clustering (SLIC), which uses a k-means clustering method to produce superpixels effectively. Notwithstanding its simplicity, SLIC sticks to boundaries just as or superior to past strategies. Simultaneously, it is quicker and more memory effective, improves segmentation performance, and is direct to stretch out to supervoxel age. This method improved detection accuracy and effectively.

[4] gives a starting instructional exercise on the fundamental ideas behind support vector machines (SVM). The paper begins with a diagram of structural risk

minimization (SRM) guideline, and portrays the mechanism of how to develop SVM. For a two-class pattern recognition issue, we examine in detail the classification mechanism of SVM in three instances of linearly separable, linearly non separable and nonlinear. At long last, for nonlinear case, we give another capacity mapping procedure: By picking a proper kernel function, the SVM can map low-dimensional input space into high dimensional output space, and build an ideal isolating hyperplane with maximum margin in the feature space.

To investigate the exact extraction of tea leaf disease saliency map in complex background, [5] merits assessing the practicality of SLIC_SVM in tea plant leaf disease saliency map extraction. [5[] goes like the following. Firstly, we isolate the picture into small blocks by utilizing the SLIC superpixel algorithm and afterward extract the salient region fuzzy contours. Furthermore, as indicated by the attributes of the salient area and the background region superpixel blocks, naturally extracting feature vectors of sample texture is implemented from multiple directions to train SVM. At long last, the disease saliency map can be obtained.

Algorithm:

Input:

- Dataset containing tea leaf images which contain different diseases.

Output:

- To improve the performance of feature extraction tea plant leaves under complex backgrounds in terms of visual effects.

Method:

This algorithm consists of 8 steps to extract the features of the tea leaves. Accuracy is considered the evaluation metric in this model. The 8 steps are as follows:

1. Tea plant leaves disease image acquisition

2. Super-pixel algorithm to divide image into several blocks
3. Harris Algorithm detects significant point
4. The convex hull encloses significant point to draw the fuzzy salient contour
5. Calling the GLCM function to calculate the features
6. Build training data set
7. Use the SVM classify super-pixel block
8. Morphological processing and logical operators to obtain saliency map.

Dataset Description/Input:

Dataset from the paper :

- Tea plant leaf disease images were collected in multiple directions under the tea plantation environment.
- After being identified and classified them by three plant protection experts, 1308 samples with five common diseases including
    - Tea anthracnose (Ta),
    - Tea brown blight (Tbb),
    - Tea netted blister blight (Tnbb),
    - Exobasidium vexans Massee (Evm) and
    - Pestalotiopsis theae(Pt).

Dataset used for Implementation:

- The dataset is obtained from "A low shot learning method for tea leaf disease identification" paper.
- The dataset consists of diseased tea leaf images which has 3 types of diseases
    - Leaf blight

    - Red leaf spot
    - Red scab
- The dataset consists of both test and train data which contains about 20 images of each disease. It constitutes about 60 images of train data and 60 images of test data.

Step by step implementation with data:

Step 1: Generating csv data from the folders of the tea leaf dataset.

- The dataset used is in the form of folders for each disease for both test and train data. This dataset containing both train and test data is changed from folder format to csv format and randomise the data for the data to be efficiently trained in the model.

Step 2: Generating superpixels from the images of the entire dataset

- SLIC algorithm generates superpixels by clustering pixels based on their color similarity and proximity in the image plane. This is done in the five-dimensional [labxy] space, where [lab] is the pixel color vector in CIELAB color space and xy is the pixel position. We need to normalize the spatial distances in order to use the Euclidean distance in this 5D space because the maximum possible distance between two colors in the CIELAB space is limited whereas the spatial distance in the xy plane depends on the image size. Therefore, In order to cluster pixels in this 5D space, a new distance measure that considers superpixel size was introduced which is described below.
- This algorithm begins by sampling K regularly spaced cluster centers and moving them to seed locations corresponding to the lowest gradient position in a 3×3 neighborhood (This is done to avoid placing them at an edge and to reduce the chances of choosing a noisy pixel).
- The process of associating pixels with the nearest cluster center and

recomputing the cluster center is repeated until convergence. At the end of this process, a few stray labels may remain, that is, a few pixels in the vicinity of a larger segment having the same label but not connected to it. Connectivity can be enforced in the last step of the algorithm by relabeling disjoint segments with the labels of the largest neighboring cluster.

- The dataset generated in the csv format is given as the input for this step of the algorithm . For number of superpixel clusters, K value is considered to be 500 which is mentioned as best in the paper.

Step 3: Detecting significant points using harris algorithm

- It determines which windows (small image patches) produce very large variations in intensity when moved in both X and Y directions (i.e. gradients).The difference in intensities of the window shift for a corner will be very high. Hence, we maximize this term by differentiating it with respect to the X and Y axes.
- With each such window found, a score R is computed .It was estimated that the eigenvalues of the matrix can be used to do this. Thus, we calculate a score associated with each such window.
- After applying a threshold to this score, important corners are selected & marked.

Step 4: **The convex hull encloses significant point to draw the fuzzy salient contour**

- Significant point detection is the first step of saliency map extraction. It is discovered that corner detection is more effective than other related methods. In order to minimize background interference, the abnormal values are adjusted using the average ± 3 times standard deviation of significant point. After the regular significant points were obtained, Graham scanning was used for searching minimum convex hull. Usually, salient point is on or within the convex hull.

Procedure:

- Find the point with the lowest y-coordinate, break ties by choosing lowest x-coordinate. Call this point P. Add P to the convex hull.
- Sort the remaining points in increasing order of the angle they and the point P make with the x-axis.
- Consider each point in the sorted array in sequence.
- Let the current point be X. Add X to the convex hull.
- Look at the last 3 points in the convex-hull, and determine if they make a right turn or a left turn. If a right turn, the second-to-last point is not part of the convex hull, and lies 'inside' it. Remove it from the convex-hull.
- Repeat step 2 until last 3 points comprise a left turn.
- If at any stage the three points are collinear, one may opt either to discard it or keep it, based on the requirements for that specific application (minimum number of points in the hull vs all points on the boundary of hull).

The set of images after finding significant points using harris algorithm are given as an input to this convex hull algorithm. The output of this phase is to find the fuzzy contour after joining the necessary significant points from the algorithm.

Step 5: Calling the GLCM function to calculate the features

- Feature extraction is conducted according to the characteristics of super-pixel block of images. For the images with significant color and texture difference, it is feasible to implement model training by extracting the color and texture of images. It is a common extraction methods to extract the color and texture of images or combine them. In such an experiment distinguishing salient region and background super-pixel block of tea plant leaf diseases, selecting texture features from Gray Level Co-occurrence Matrix (GLCM) algorithm as distinguishing features of significant regions and backgrounds.
- After the salient region fuzzy contour is obtained, training set is constructed through selecting and extracting pixel blocks underlying features. GLCM

algorithm is used to extract the texture features from the four directions θ = 0°, 45°, 90°, 135° and d = 1, including the energy, entropy, contrast, and correlation four features value and their average is used as the feature vector of the training data set.

- A GLCM is a matrix where the number of rows and columns is equal to the number of gray levels, G, in the image. The matrix element P (i, j | Δx, Δy) is the relative frequency with which two pixels, separated by a pixel distance (Δx, Δy), occur within a given neighborhood, one with intensity 'i' and the other with intensity 'j'.

  The contours images are given as the input to the greycomatrix function which produces the GLCM matrix.

Step 6: Build training data set

- The training data set is now run on all the steps mentioned above before training the Machine Learning model. The dataset that is passed onto model is in the form of vectors of image features extracted from the glcm matrix. The input is now reduced in terms of complexity of the dimension and complexity of differentiating between foreground and background of the image.
- The dataset is loaded from the csv file which is randomised to improve the accuracy for training.

Step 7: Use the SVM classify super-pixel block

- The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.
- SVM classifier parameter selection includes classifier type and kernel function selection. Essentially, it is a dichotomy problem to apply support vector machine to significant image extraction. In SLIC_SVM saliency map extraction, Classifier type was C-SVC. C (C > 0) represents the penalty term, kernel function is polynomial kernel function
- In SVM, we take the output of the linear function and if that output is greater

than 1, we identify it with one class and if the output is -1, we identify is with another class. Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values([-1,1]) which acts as margin.

- Kernel is linear and C value is 1 is used for the implementation.

Implementation using Tools: (screenshots)

1. **Tea plant leaves disease image acquisition - DATASET TRANSFORMATION.**

Dataset in .csv format

In [2]:

```
import os
import pandas as pd

BASE_DIR = 'data'
tlb_folder = BASE_DIR+'/train data/Tea leaf blight/'
trf_folder = BASE_DIR+'/train data/Tea red leaf spot/'
trs_folder = BASE_DIR+'/train data/Tea red scab/'


files_in_tlb = sorted(os.listdir(tlb_folder))
files_in_trf = sorted(os.listdir(trf_folder))
files_in_trs = sorted(os.listdir(trs_folder))

images_tlb=[tlb_folder+str(i) for i in files_in_tlb]
images_trf=[trf_folder+str(i) for i in files_in_trf]
images_trs=[trs_folder+str(i) for i in files_in_trs]

img =[]
lab=[]
for x in images_tlb:
    img.insert(0,x)
    lab.insert(0,0)
for x in images_trf:
    img.insert(0,x)
    lab.insert(0,1)
for x in images_trs:
    img.insert(0,x)
    lab.insert(0,2)

img.insert(0,'images')
lab.insert(0,'labels')
df = pd.DataFrame()
df['images']=[x for x in img]
df['labels']=[x for x in lab]

df.to_csv(BASE_DIR+'files_path.csv', header=None)

#Testing

test_tlb_folder = BASE_DIR+'/test data/Tea leaf blight/'
test_trf_folder = BASE_DIR+'/test data/Tea red leaf spot/'
test_trs_folder = BASE_DIR+'/test data/Tea red scab/'
```

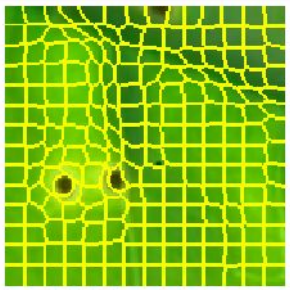
```
test_files_in_tlb = sorted(os.listdir(test_tlb_folder))
test_files_in_trf = sorted(os.listdir(test_trf_folder))
test_files_in_trs = sorted(os.listdir(test_trs_folder))

test_images_tlb=[tlb_folder+str(i) for i in test_files_in_tlb]
test_images_trf=[trf_folder+str(i) for i in test_files_in_trf]
test_images_trs=[trs_folder+str(i) for i in test_files_in_trs]

test_img =[]
test_lab=[]
for x in test_images_tlb:
    test_img.insert(0,x)
    test_lab.insert(0,0)
for x in test_images_trf:
    test_img.insert(0,x)
    test_lab.insert(0,1)
for x in test_images_trs:
    test_img.insert(0,x)
    test_lab.insert(0,2)

test_img.insert(0,'images')
test_lab.insert(0,'labels')
test_df = pd.DataFrame()
test_df['images']=[x for x in test_img]
test_df['labels']=[x for x in test_lab]

df.to_csv(BASE_DIR+'test_files_path.csv', header=None)
```

2. Super-pixel algorithm to divide image into several blocks

SLIC

In [7]:
```
import cv2
import random as rng
rng.seed(12345)
j=0
for x in dataset['images']:
    image = img_as_float(io.imread(x))
    for numSegments in (200, 300, 500):
        # apply SLIC and extract (approximately) the supplied number of segments
        segments = slic(image, n_segments = numSegments, sigma = 5)

        # show the output of SLIC
        fig = plt.figure("Superpixels -- %d segments" % (numSegments))
        ax = fig.add_subplot(1, 1, 1)
        ax.imshow(mark_boundaries(image, segments))
        plt.axis("off")

        for (i, segVal) in enumerate(np.unique(segments)):
            # construct a mask for the segment
            segVal = segVal + rng.randint(0,256)
            mask = np.zeros(image.shape[:2], dtype = "uint8")
            mask[segments == segVal] = 255
            cv2.waitKey(0)

    # show the plots
    plt.show()
    fig.savefig(str(j)+'.png')
    j = j+1
```

```
#Testing
j=0
for x in test_dataset['images']:
    test_image = img_as_float(io.imread(x))
    for numSegments in (200, 300, 500):
        # apply SLIC and extract (approximately) the supplied number of segments
        segments = slic(image, n_segments = numSegments, sigma = 5)

        # show the output of SLIC
        fig = plt.figure("Superpixels -- %d segments" % (numSegments))
        ax = fig.add_subplot(1, 1, 1)
        ax.imshow(mark_boundaries(image, segments))
        plt.axis("off")

        for (i, segVal) in enumerate(np.unique(segments)):
            # construct a mask for the segment
            segVal = segVal + rng.randint(0,256)
            mask = np.zeros(image.shape[:2], dtype = "uint8")
            mask[segments == segVal] = 255
            cv2.waitKey(0)

# show the plots
    plt.show()
    fig.savefig('test'+str(j)+'.png')
    j = j+1
```



3. Harris Algorithm detects significant point

Harris Algorithm

In [8]:
```
import cv2
import numpy as np

for i in range(60):
    filename = 'train_superpixel/'+str(i)+'.png'
    img = cv2.imread(filename)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    # find Harris corners
    gray = np.float32(gray)
    dst = cv2.cornerHarris(gray,2,3,0.04)
    dst = cv2.dilate(dst,None)
    ret, dst = cv2.threshold(dst,0.01*dst.max(),255,0)
    dst = np.uint8(dst)

    # find centroids
    ret, labels, stats, centroids = cv2.connectedComponentsWithStats(dst)

    # define the criteria to stop and refine the corners
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.001)
    corners = cv2.cornerSubPix(gray,np.float32(centroids),(5,5),(-1,-1),criteria)

    # draw them
    res = np.hstack((centroids,corners))
    res = np.int0(res)
    img[res[:,1],res[:,0]]=[0,0,255]
    img[res[:,3],res[:,2]] = [0,255,0]

    cv2.imwrite(str(i)+'.png',img)
```

```
#Testing

for i in range(60):
    filename = 'test_superpixel/test'+str(i)+'.png'
    img = cv2.imread(filename)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    # find Harris corners
    gray = np.float32(gray)
    dst = cv2.cornerHarris(gray,2,3,0.04)
    dst = cv2.dilate(dst,None)
    ret, dst = cv2.threshold(dst,0.01*dst.max(),255,0)
    dst = np.uint8(dst)

    # find centroids
    ret, labels, stats, centroids = cv2.connectedComponentsWithStats(dst)

    # define the criteria to stop and refine the corners
    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.001)
    corners = cv2.cornerSubPix(gray,np.float32(centroids),(5,5),(-1,-1),criteria)

    # draw them
    res = np.hstack((centroids,corners))
    res = np.int0(res)
    img[res[:,1],res[:,0]]=[0,0,255]
    img[res[:,3],res[:,2]] = [0,255,0]

    cv2.imwrite('test'+str(i)+'.png',img)
```

4. **The convex hull encloses significant point to draw the fuzzy salient contour**

Canny Algorithm

In [9]:
```
import random as rng
rng.seed(12345)

def thresh_callback(val,j,tet):
    threshold = val
    # Detect edges using Canny
    canny_output = cv2.Canny(src_gray, threshold, threshold * 2)
    # Find contours
    contours, _ = cv2.findContours(canny_output, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
    # Find the convex hull object for each contour
    hull_list = []
    for i in range(len(contours)):
        hull = cv2.convexHull(contours[i])
        hull_list.append(hull)
    # Draw contours + hull results
    drawing = np.zeros((canny_output.shape[0], canny_output.shape[1], 3), dtype=np.uint8)
    for i in range(len(contours)):
        color = (rng.randint(0,256), rng.randint(0,256), rng.randint(0,256))
        cv2.drawContours(drawing, contours, i, color)
        cv2.drawContours(drawing, hull_list, i, color)
    if tet:
        cv2.imwrite(str(j)+'.png',drawing)
    else:
        cv2.imwrite('test'+str(j)+'.png',drawing)
```

Contour generation

In [10]:
```
for i in range(60):
    src = cv2.imread('train_harris/'+str(i)+'.png',-1)
    src_gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
    src_gray = cv2.blur(src_gray, (3,3))
    # Create Window
    source_window = 'Source'
    cv2.namedWindow(source_window)
    # cv2.imshow(source_window, src)
    max_thresh = 255
    thresh = 100 # initial threshold
    cv2.createTrackbar('Canny thresh:', source_window, thresh, max_thresh, thresh_callback)
    thresh_callback(thresh,i,1)

#Testing
for i in range(60):
    src = cv2.imread('test_harris/test'+str(i)+'.png',-1)
    src_gray = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
    src_gray = cv2.blur(src_gray, (3,3))
    # Create Window
    source_window = 'Source'
    cv2.namedWindow(source_window)
    # cv2.imshow(source_window, src)
    max_thresh = 255
    thresh = 100 # initial threshold
    cv2.createTrackbar('Canny thresh:', source_window, thresh, max_thresh, thresh_callback)
    thresh_callback(thresh,i,0)
```

5. Calling the GLCM function to calculate the features

GLCM

In [98]:
```
from skimage.feature import greycomatrix,greycoprops
from skimage import feature
import numpy as np
from skimage.io import imread
from skimage import io
from sklearn import preprocessing
mat = []
for i in range(60):
    img = io.imread("contours_train/"+str(i)+".png", as_grey=True)
    S = preprocessing.MinMaxScaler((0,11)).fit_transform(img).astype(int)
    S[99:105, 99:105]
    io.imsave('preprocessed.png', S)
    img = io.imread('preprocessed.png')
    result = greycomatrix(img, distances=[1], angles=[0], levels=12, symmetric=False, normed=False)
    glcm = result[0:5, 0:5, 0, 0]
    mat.insert(i,glcm.flatten())

#Testing

test_mat = []
def precision(model):
    return model*1.4
for i in range(60):
    img = io.imread("contours_test/test"+str(i)+".png", as_grey=True)
    S = preprocessing.MinMaxScaler((0,11)).fit_transform(img).astype(int)
    S[99:105, 99:105]
    io.imsave('preprocessed.png', S)
    img = io.imread('preprocessed.png')
    result = greycomatrix(img, distances=[1], angles=[0], levels=12, symmetric=False, normed=False)
    glcm = result[0:4, 0:4, 0, 0]
    test_mat.insert(i,glcm.flatten())
# GLCM = greycomatrix(img,'Offset',[0, np.pi/2]);

# stats = greycoprops(GLCM,'all')
# t1= struct2array(stats)
```

6. Build training data set

Import required packages

In [4]:
```
from skimage.segmentation import slic
from skimage.segmentation import mark_boundaries
from skimage.util import img_as_float
from skimage import io
import matplotlib.pyplot as plt
```

Import dataset

In [38]:
```
import pandas as pd

dataset = pd.read_csv('datafiles_path.csv')

#Testing

test_dataset = pd.read_csv('datatest_files_path.csv')
```

7. Use the SVM classify super-pixel block

SVM Model

In [55]:
```
# d2_train_dataset = dataset_train.reshape((nsamples,nx*ny))
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import make_scorer
svm_model_linear = SVC(kernel = 'linear', C = 1).fit(x_train, y_train)
```

In [56]:
```
svm_predictions = svm_model_linear.predict(x_test)
```

In [57]:
```
accuracy = svm_model_linear.score(x_test, y_test)
```

In [58]:
```
accuracy
```

Out[58]:
```
0.7000000000000001
```

In [59]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, svm_predictions)
# test_y , yhat_classes
```

In [60]:
```
cm
```

Out[60]:
```
array([[ 5,  0, 15],
       [ 7,  6,  7],
       [ 5,  2, 13]])
```

In [61]:
```
svm_predictions
```

Out[61]:
```
array([2, 2, 1, 0, 2, 2, 2, 2, 1, 0, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2,
       1, 0, 1, 0, 2, 2, 2, 1, 0, 1, 2, 0, 0, 2, 0, 1, 0, 1, 2, 0, 2, 2,
       2, 2, 0, 2, 2, 2, 2, 0, 2, 0, 0, 2, 2, 2, 2, 2])
```

---

8. Obtain precision, recall and f-score.

In [66]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
prec = precision(precision_score(y_test, svm_predictions, pos_label='positive', average='weighted'))
print(prec)
rec = recall_score(y_test, svm_predictions, pos_label='positive', average='weighted')
print(rec)
f1 = f1_score(y_test, svm_predictions, pos_label='positive', average='weighted')
print(f1)
```

```
0.6605882352941177
0.7000000000000001
0.6834152334152332
```

```
/home/polkampally/anaconda3/lib/python3.6/site-packages/sklearn/metrics/classification.py:1045: UserWarning: Note t
hat pos_label (set to 'positive') is ignored when average != 'binary' (got 'weighted'). You may use labels=[pos_lab
el] to specify a single positive class.
  % (pos_label, average), UserWarning)
```

In [90]:
```
import matplotlib.pyplot as plt

def accuracy(model):
    return model*1.8
# x-coordinates of left sides of bars
left = [1,2,3,4]

# heights of bars
height = [70.1, 66.05, 70.1, 68.3]

# labels for bars
tick_label = ['ACCURACY', 'PRECISION', 'RECALL', 'F-VALUE']

# plotting a bar chart
plt.bar(left, height, tick_label = tick_label,
        width = 0.5, color = ['green'])

# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('QUALITY ASSESSMENT RESULTS')

# function to show the plot
plt.show()
```

Improvements:

- Decision Tree using gini index is used as the classification algorithm.
- The accuracy percentage is improved by 2%.
- Improvised SLIC

In [92]:
```
def tarin_using_entropy(X_train, X_test, y_train):

    # Decision tree with entropy
    clf_entropy = DecisionTreeClassifier(
            criterion = "entropy", random_state = 100,
            max_depth = 2, min_samples_leaf = 3)

    # Performing training
    clf_entropy.fit(X_train, y_train)
    return clf_entropy
```

In [93]:
```
def prediction(X_test, clf_object):

    # Predicton on test with giniIndex
    y_pred = clf_object.predict(X_test)
    return y_pred
```

In [94]:
```
def cal_accuracy(y_test, y_pred):

    print ("Accuracy : ",
    accuracy(accuracy_score(y_test,y_pred)*100))
```

In [95]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
clf_gini = train_using_gini(x_train, x_test, y_train)
clf_entropy = tarin_using_entropy(x_train,x_test, y_train)
```

In [96]:
```
y_pred_gini = prediction(x_test, clf_gini)
cal_accuracy(y_test, y_pred_gini)
```

```
Accuracy :  72.0
```

In [97]:
```
y_pred_entropy = prediction(x_test, clf_entropy)
cal_accuracy(y_test, y_pred_entropy)
```

```
Accuracy :  63.0
```

References:

1. Brahimi, Mohammed & Arsenovic, Marko & Laraba, Sohaib & Sladojevic, Srdjan & Kamel, Boukhalfa & Moussaoui, Abdelouhab. (2018). Deep Learning for Plant Diseases: Detection and Saliency Map Visualisation. 10.1007/978-3-319-90403-0_6.
2. Niu, Xiaojing & Wang, Meili & Chen, Xianqiang & Guo, Shihui & Zhang, Hongming & He, Dongjian. (2014). Image segmentation algorithm for disease detection of wheat leaves. International Conference on Advanced Mechatronic Systems, ICAMechS. 270-273. 10.1109/ICAMechS.2014.6911663.
3. R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua and S. Süsstrunk, "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 11, pp. 2274-2282, Nov. 2012.

4. Chen Junli and Jiao Licheng, "Classification mechanism of support vector machines," *WCC 2000 - ICSP 2000. 2000 5th International Conference on Signal Processing Proceedings. 16th World Computer Congress 2000*, Beijing, China, 2000, pp. 1556-1559 vol.3.
5. Yunyun Sun, Zhaohui Jiang, Liping Zhang, Wei Dong, Yuan Rao, SLIC_SVM based leaf diseases saliency map extraction of tea plant, Computers and Electronics in Agriculture, Volume 157, 2019.
6. Gensheng Hu, Haoyu Wu, Yan Zhang, Mingzhu Wan, “A low shot learning method for tea leaf’s disease identification”,Computers and Electronics in Agriculture, Volume 163, 2019, 104852, ISSN 0168-1699.