**SEGMENTATION CODE:**

def img\_segmentation(img):

rgb\_img=cv2.cvtColor(img,cv2.COLOR\_BGR2RGB)

lab\_img=cv2.cvtColor(rgb\_img,cv2.COLOR\_RGB2LAB)

g\_m=cv2.inRange(lab\_img,np.array([0,130,0]),np.array([250,255,255]))

g\_m=~g\_m

d\_m=cv2.inRange(lab\_img,np.array([0,130,135]),np.array([255,255,235]))

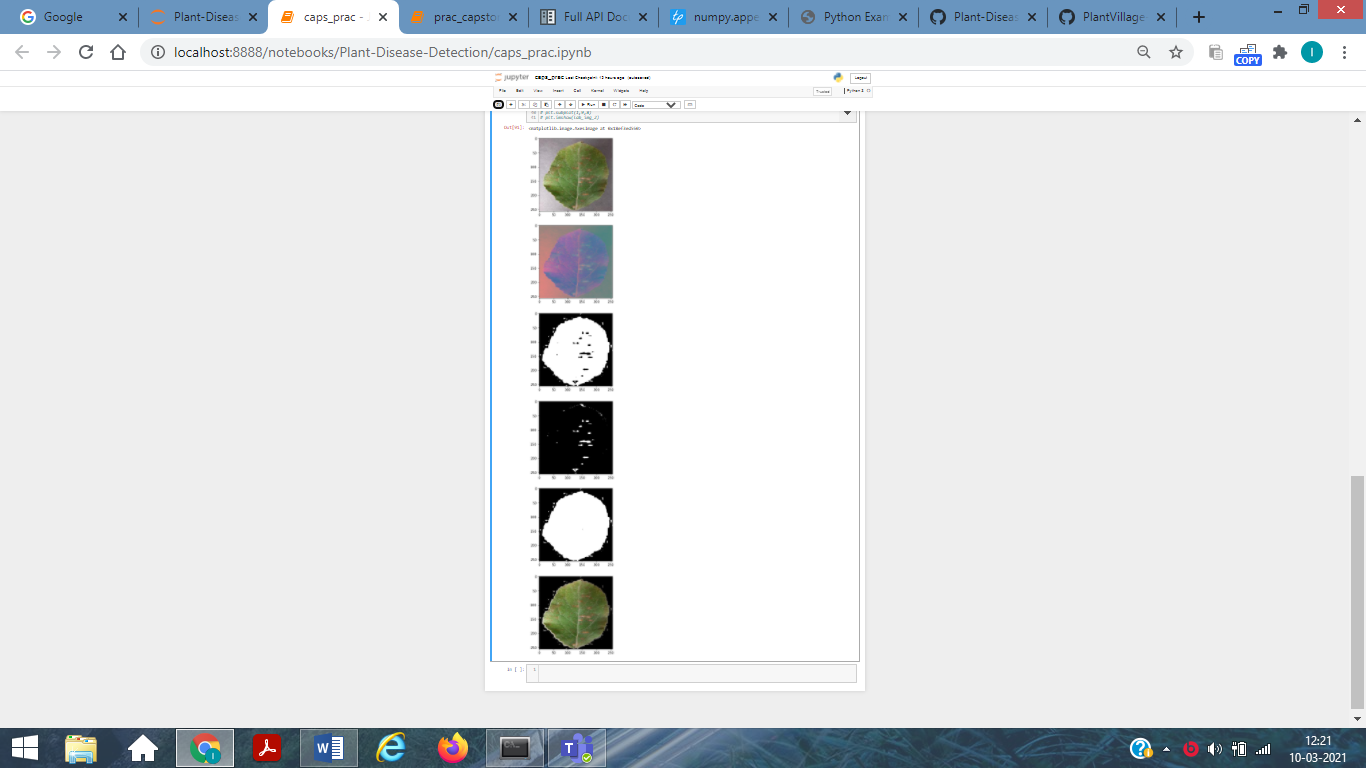
d\_m=d\_m

f\_m=d\_m+g\_m

seg\_img=cv2.bitwise\_and(rgb\_img,rgb\_img,mask=f\_m)

return seg\_img

**Result of segmentation**



**Features extracted**

**1)** **Hu moments**

**Code:**

def fd\_hu\_moments(image):

image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

feature = cv2.HuMoments(cv2.moments(image)).flatten()

return feature

2) **Haralick features**

**Code:**

def fd\_haralick(image):

gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

haralick = mahotas.features.haralick(gray).mean(axis=0)

return haralick

3) **Color histogram**

def fd\_histogram(image, mask=None):

image = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

hist = cv2.calcHist([image], [0, 1, 2], None, [bins, bins, bins], [0, 256, 0, 256, 0, 256])

cv2.normalize(hist, hist)

return hist.flatten()

4**) Color features**

**Code:**

def fd\_color(image):

import mahotas

import cv2

import numpy as np

from skimage.feature.texture import greycomatrix

# test\_img=cv2.imread(image)

test\_hsv=cv2.cvtColor(image,cv2.COLOR\_BGR2HSV)

h\_mean,h\_std=cv2.meanStdDev(h)

s\_mean,s\_std=cv2.meanStdDev(s)

color\_fea=np.array([h\_mean,h\_std,s\_mean,s\_std])

haralick\_fea = mahotas.features.haralick(v).mean(axis=0)

color\_fea=np.append(color\_fea,haralick\_fea)

return color\_fea

**Extracting Features and Organizing it for training data**

**Code:**

ptr=1

# data=pd.dataframe()

for training\_name in train\_labels:

# join the training data path and each species training folder

dir = os.path.join(train\_path, training\_name)

current\_label = training\_name

if(ptr==1):

images\_per\_class=805

else:

images\_per\_class=904

for x in range(1,images\_per\_class+1):

file = dir + "/" + str(x) + ".jpg"

image = cv2.imread(file)

image = cv2.resize(image, fixed\_size)

IMG\_SEGMENT = img\_segmentation(image)

fv\_hu\_moments = fd\_hu\_moments(IMG\_SEGMENT)

fv\_haralick = fd\_haralick(IMG\_SEGMENT)

fv\_histogram = fd\_histogram(IMG\_SEGMENT)

fv\_color\_fea= fd\_color(IMG\_SEGMENT)

global\_feature = np.hstack([fv\_histogram, fv\_haralick, fv\_hu\_moments,fv\_color\_fea])

labels.append(current\_label)

global\_features.append(global\_feature)

print("[STATUS] processed folder: {}".format(current\_label))

ptr+=1

print(fv\_color\_fea)

print(len(fv\_histogram) )

print("[STATUS] completed Global Feature Extraction...")

import pandas as pd

import numpy as np

from matplotlib import pyplot as plt

from sklearn.preprocessing import MinMaxScaler

scaler= MinMaxScaler(feature\_range=(0, 1))

rescaled\_features=scaler.fit\_transform(global\_features)

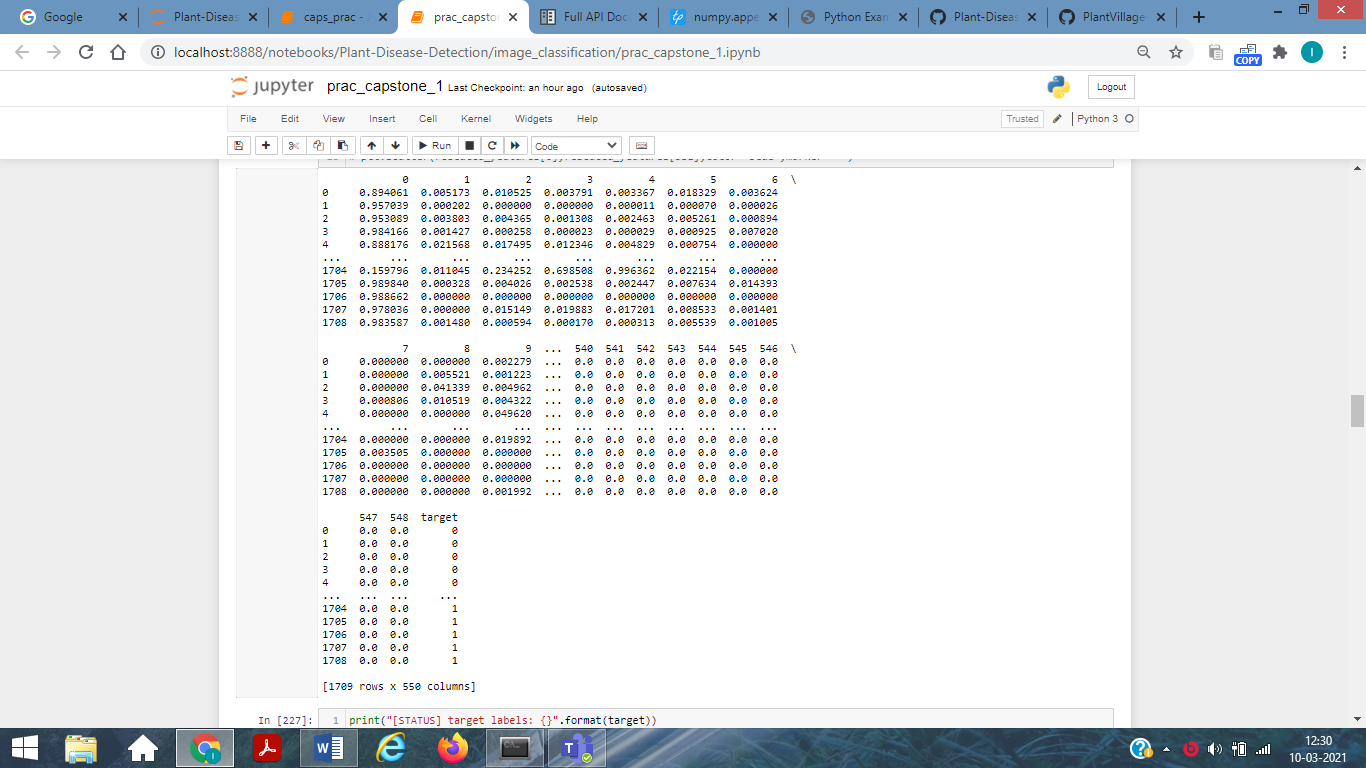
# data=pd.DataFrame(rescaled\_features,columns=[np.arange(0,532,1)])

data=pd.DataFrame(rescaled\_features)

# target=labels;

data["target"]=target

print(data)



**Random forest**

While building subsets of data for trees, the word “random” comes into the picture. A subset of data is made by randomly selecting x number of features (columns) and y number of examples (rows) from the original dataset of n features and m examples.

Entropy is a measure of the randomness of a system. The entropy of sample space S is the expected number of bits needed to encode the class of a randomly drawn member of S.  Here we have 14 rows in our data so 14 members.

*Entropy E(S)= -∑p(x)\*log2(p(x))*

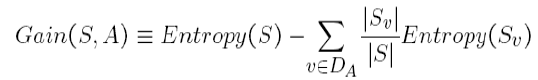
The entropy of the system is calculated by the above formula where p(x) is the probability of getting class x from those 14 members(assuming). We have two classes here one is Yes and the Other is No in target column.

*E(S)= -[p(Yes)\*log(p(Yes))+ p(No)\*log(p(No))];*

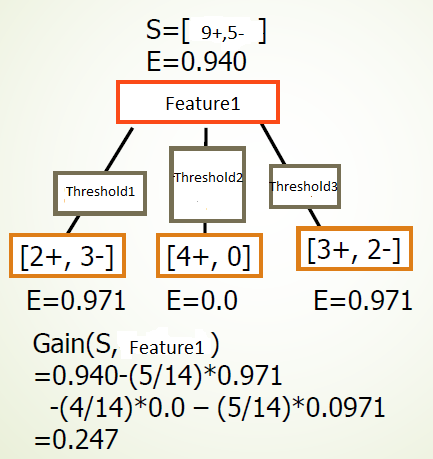
*Where yes means diseased otherwise healthy*

***Information Gain***

The information gain is the amount by which the Entropy of the system reduces due to the split that we have done. We have created the tree using observations. But how come we decided that we should first split the data based on a particular feature and not on any other feature? The reason is that this split was reducing the entropy by the maximum amount and gain is maximum.



For ex

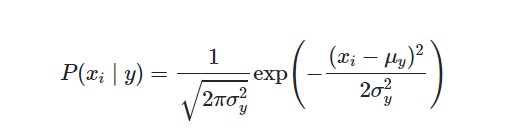


**Guassian Naïve Bayes**

Bayes’ theorem states the following relationship, given class variable y and dependent feature vector x(1) through x(n), :

Since P(x(1),…,x(n)) is constant given the input, we can use the following classification rule:

When working with continuous data, an assumption often taken is that the continuous values associated with each class are distributed according to a normal (or Gaussian) distribution. The likelihood of the features is assumed to be-



Where μ=mean of feature x given class Y or in this case diseased plant

X*i= ith feature value for the sample test*

*σy2* =variance of feature x given class Y or in this case diseased plant

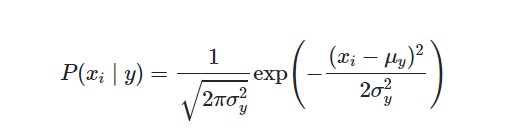
**Step wise Algo**

1) From Training set find the prior probability P(diseased)=(diseased)/(diseased + healthy) P (healthy)=(healthy)/(diseased + healthy).

2) Find the mean, variance for the all the 550 feature given the target is plant is healthy or diseased separately from the test data, i.e. if plant is healthy calculate the mean and variance of the given feature.

3) From the test data, for all the leaves we now predict whether that belongs to which class i.e healthy or diseased.

Using below mentioned formulas



P(y)=prior probability

Where μ=mean of feature x given class Y or in this case diseased plant or healthy one (from training data)

X*i= ith feature value for the test data*

*σy2* =variance of feature x given class Y or in this case diseased plant or healthy one (from training data)

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Algorithm | Type | Accuracy | Interpretabilty | Data Size Handling | Resistant to Multicollinearity | Susceptibe To Outliers | Assumptions | Training Phase time | Test Phase Timings |
| Support vector Machine | Non Parametric | High | Medium | Large | No | Yes | Flexible | Medium | Low |
| Naïve Bayes | Probabilistic | Medium | Medium | Large | Yes | Yes | Flexible | Medium | Low |
| Random Forest | Rule Based | High | Low | Large | Yes | No | Flexible | High | Medium |

2) SVM works better than Naïve Bayes as it chooses the most optimal models of all the models its generate ie.hyperplane with the maximum margin.

Kernel can be used to increase the dimensionality of the feature set,therefore making separation of data into classes easier than in the lower dimensionality