PROJECT REPORT

Predicting Crop Diseases using Machine learning

Submitted by: Kajal Singla (MT19214), Debashish mohanta (MT19211)

Articulation of the problem

Motivation :

- ► Cash crops such as maize, wheat are prone to diseases caused due to pest and microbes
- ► Pesticides and insecticides are used heavily to inconcure the prevention of crop diseases
- ► Heavy usage of chemicals is harmful for human health
- ► Prediction or detection the crop diseases can prevent the heavy usage of these chemicals
- ► The pigmentations and symptoms can be seen on diseased leaves and stems
- ► Processing the image of leaves can help in disease prediction of crop
- ► Extraction of the important features from image data and it through image processing can help in predicting the disease before its outbreak to rest of the crops

Healthy leaf

Diseased leaf

Objective:

Classification of leaf into healthy and diseased sample on the basis of their visual features extracted from their image and predicting whether a leaf is diseased on not using machine learning technique

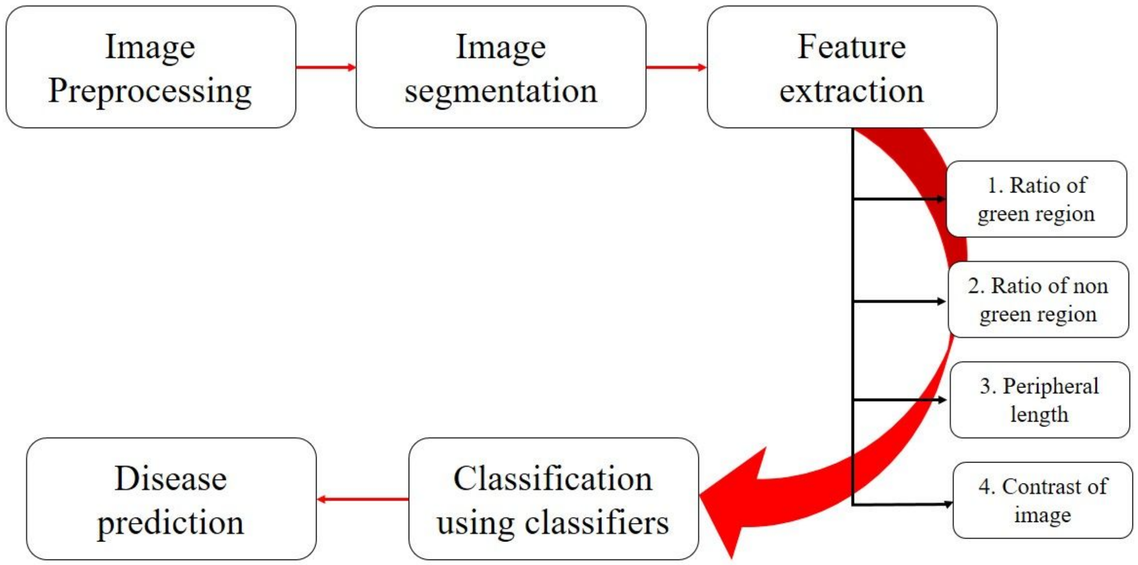
The complexity of data acquisition

Source of Data : https://www.kaggle.com/vipoooool/new-plant-diseases-dataset

The data is a set of augmented data which is a collection of images of diseased and healthy leaves. It consists of 87K rgb images and is categorised into 38 different classes. The total dataset is divided into 80/20 ratio of training and validation data. Contains 33 test images for prediction purpose.

The complexity involved in implementation

Workflow:



Algorithm used:

Support vector machine was used for classification of images data into healthy and diseased crop images samples.

The time complexity of the SVM is of the order : O(n^3)

Pseudo code:

```
Algorithm 1. Local SVM algorithm ($k$SVM)
    input  :
             training dataset $D$
             number of local models $k$
             hyper-parameter of RBF kernel function $\gamma$
             $C$ for tuning margin and errors of SVMs
    output:
             $k$ local support vector machines models
1  begin
2      /*$k$-means performs the data clustering on $D$;*/
3      creating $k$ clusters denoted by $D_1, D_2, \ldots, D_k$ and
4      their corresponding centers $c_1, c_2, \ldots, c_k$
5      #pragma omp parallel for
6      for $i \leftarrow 1$ to $k$ do
7          /*learning a local SVM model from $D_i$;*/
8          $lSVM_i = SVM(D_i, \gamma, C)$
9      end
10     return $kSVM - model = \{(c_1, lSVM_1), (c_2, lSVM_2), \ldots, (c_k, lSVM_k)\}$
11 end
```

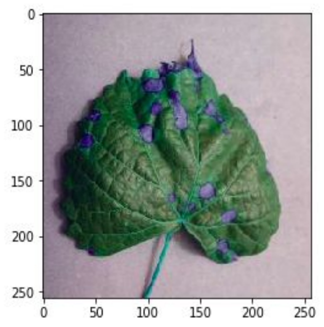
# STEPS FOLLOWED FOR ANALYSIS:

STEP 1: Loading image with the help of opencv module

- Read the image in BGR (Blue, green, red) format

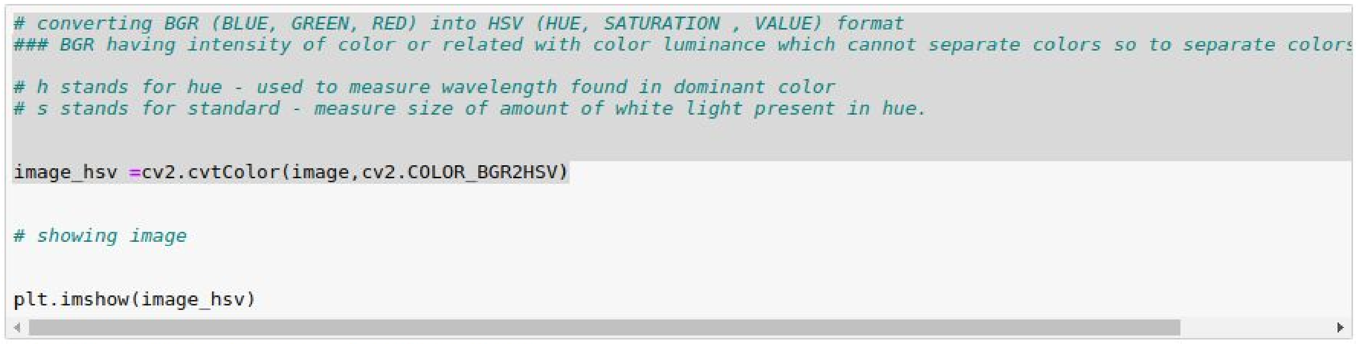
```
# loading images with the help of cv2 module in the form of BGR

image = cv2.imread('/home/kajal/MTECH/ACB/PROJECT/dataset/plantvillage-dataset/color/Grape___Black_rot/c8bfe895-d9a

# showing image

plt.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7fc2eed2ca90>
```



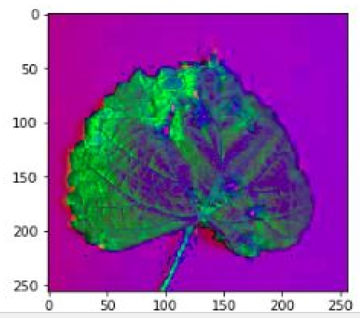
STEP 2: Convert the image from BGR (Blue, Green, Red) to HSV (Hue, Standard, Value)

- HSV helps in extracting features of image which can not be done with BGR



```
# converting BGR (BLUE, GREEN, RED) into HSV (HUE, SATURATION , VALUE) format
### BGR having intensity of color or related with color luminance which cannot separate colors so to separate colors

# h stands for hue - used to measure wavelength found in dominant color
# s stands for standard - measure size of amount of white light present in hue.


image_hsv =cv2.cvtColor(image,cv2.COLOR_BGR2HSV)


# showing image


plt.imshow(image_hsv)
```

```
<matplotlib.image.AxesImage at 0x7fc2eeef4f60>
```



STEP 3: Feature extraction of image

- Design a function which can calculate the 4 features of multiple images together and save the result in csv file
- Features:
  # feature 1: Ratio of green part in an image
  # feature 2: Ratio of non-green part
  # feature 3: Perimeter of the leaf
  # feature 4: Finding spots in image

```python
def feature_extraction (file_path, image_files, mark, output_file):

    image = np.empty(len(image_files), dtype=object)

    for n in range(0,len(image_files)):

        # loading images with the help of cv2 module in the form of BGR

        image[n] = cv2.imread(join(file_path, image_files[n]))

        # converting BGR (BLUE, GREEN, RED) into HSV (HUE, SATURATION , VALUE) format
        ### BGR having intensity of color or related with color luminance which cannot separate colors so to separat

        # h stands for hue - used to measure wavelength found in dominant color
        # s stands for standard - measure size of amount of white light present in hue.


        image_hsv = cv2.cvtColor(image[n],cv2.COLOR_BGR2HSV)


        # separating the green color part in mask variable using cv2.inRange

        #(30,0,0) - lower bound for green region
        #(70,255,255) - upper bound for green color

        ## mask is containing value of green region
        mask = cv2.inRange(image_hsv,(30,0,0) ,(70,255,255))

        # taking out the ratio by dividing mask with whole imagesize

        ratio_of_green = cv2.countNonZero(mask)/(image.size/3)
        feature1 = np.round(ratio_of_green,2)

        # Feature 2 amount of non-green colour in the picture

        feature2 = 1-feature1
```

```python
        ## feature3

        # converting the image into gray color
        image_gray = cv2.cvtColor(image[n],cv2.COLOR_BGR2GRAY)

        # fitting in gaussian curve
        gray = cv2.GaussianBlur(image_gray,(3,3),0)

        # able to find small spots in image with the help of otsu
        otsu, biny = cv2.threshold(gray,0,255,cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV )

        # help in defining the boundary
        contours, temp = cv2.findContours(biny, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        # taking out the perimeter of image
        peri = 0
        for c in contours:
            peri = peri + cv2.arcLength(c,True)
        feature3 = perimeter / 150


        ## feature 4

        # taking out the contrast from the background
        grey_scale = greycomatrix(image_gray,[1,2], [0, np.pi/2, 3*np.pi/4])
        contrast = greycoprops(grey_scale,"contrast")

        feature_4 = contrast[0][0]+contrast[0][1]+contrast[0][2]+contrast[1][0]+contrast[1][1]+contrast[1][2]
        feature_4 = feature_4/20000

        # writing data in csv file
        l = [int(mark),feature1,feature2,feature3,feature_4]
        out = open(output_file, "a")
        for row in l:
            out.write('%f,'%row)
```

Example of feature 3 extraction:

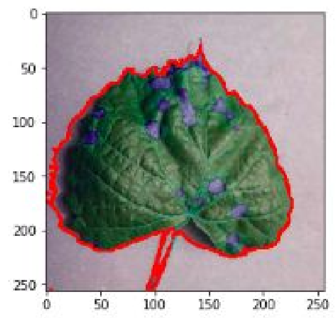
Feature 3 Periphery

converts the pixels of the image into array of numbers and matrix

```
# cv2.split - it convert the hsv image into 3 different matrix sections of blue , green, red respectively.
##Each matrix is to dimensional-array

(matrix_b, matrix_g, matrix_r) =cv2.split(image_hsv)
```

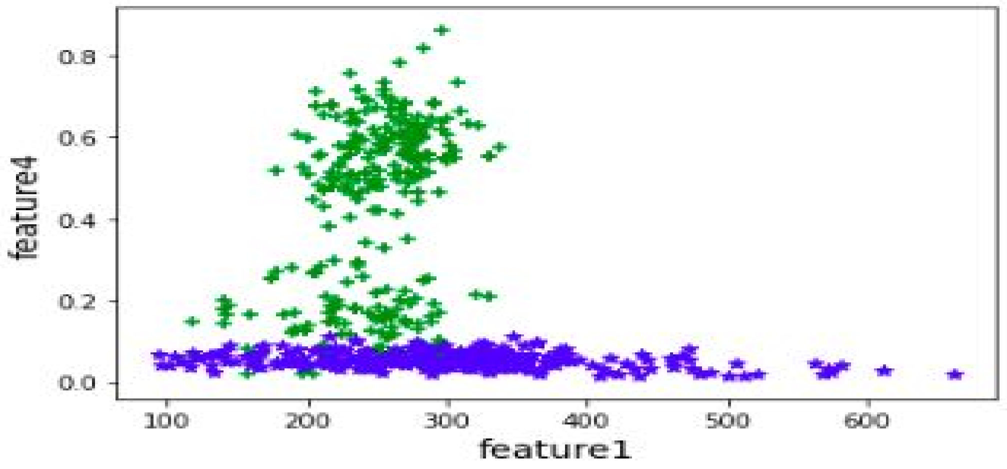
```
image_gray =cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

gray =cv2.GaussianBlur(image_gray,(3,3),0)

binary,otsu = cv2.threshold(gray,0,255,cv2.THRESH_OTSU | cv2.THRESH_BINARY_INV )

contours = cv2.findContours(otsu, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[1]

h= cv2.drawContours(image, contours, -1,(255,0,0),thickness =2)

plt.imshow(h)
```

```
<matplotlib.image.AxesImage at 0x7fc2eebf94e0>
```



#Plot between feature 1 and feature 4

```
Healthy with green and diseased with blue
474 474 119 119
```



STEP 4: Divide the data into train and test with 8: 2 ratio

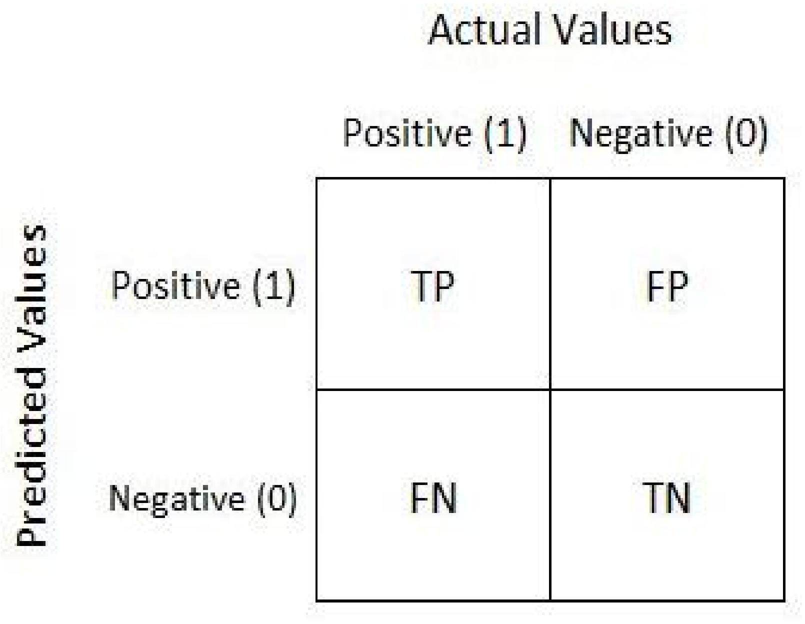
- Done with the help of 'train_test_split' function

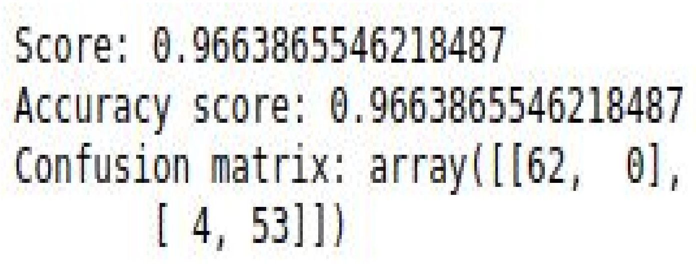
STEP 5: Train data with SVM

Explanation of findings.

Find accuracy and confusion matrix of test data

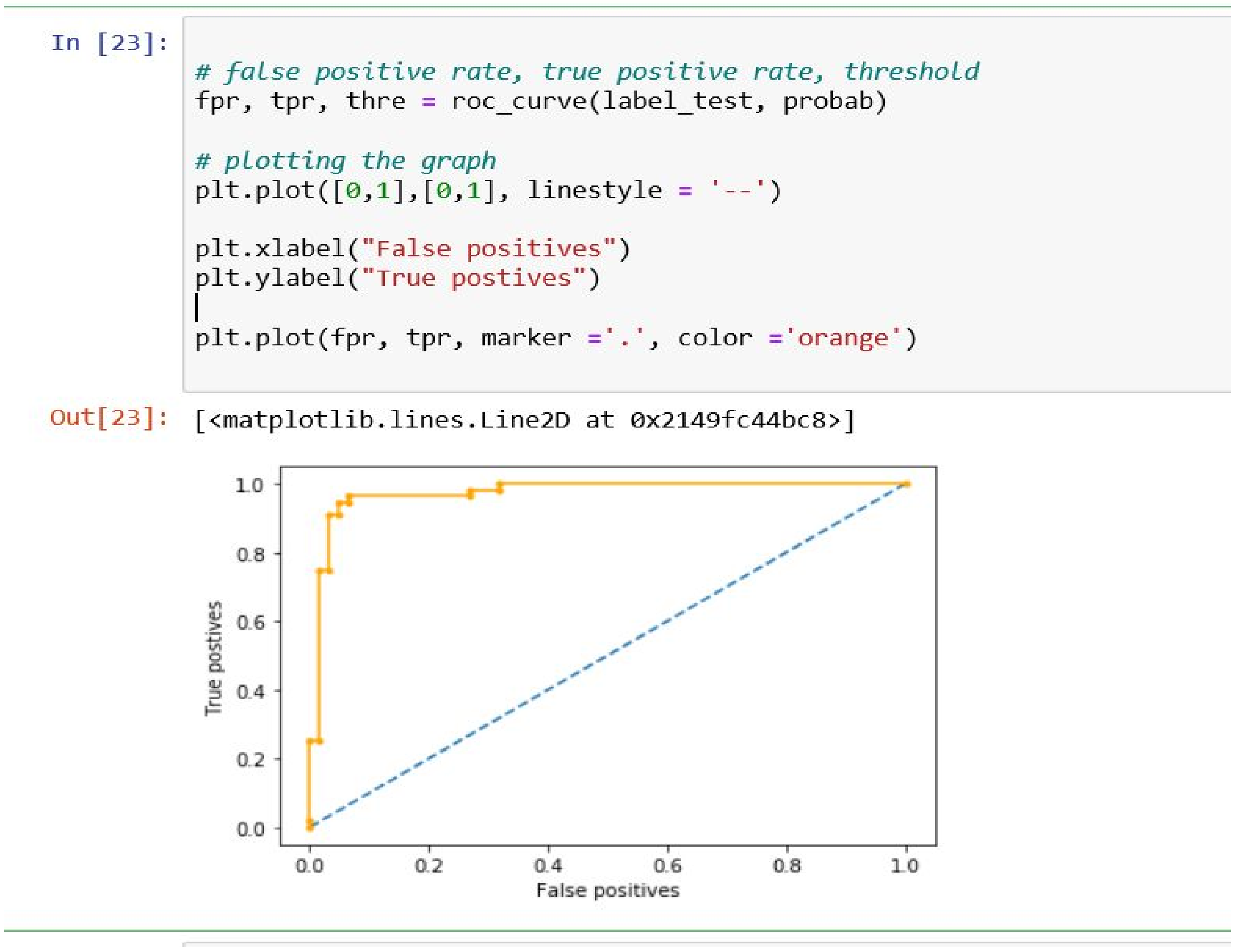
- Getting >0.90 accuracy
- Confusion matrix





```
Score: 0.9663865546218487
Accuracy score: 0.9663865546218487
Confusion matrix: array([[62,  0],
       [ 4, 53]])
```

Roc-Curve:



```
In [23]:
# false positive rate, true positive rate, threshold
fpr, tpr, thre = roc_curve(label_test, probab)

# plotting the graph
plt.plot([0,1],[0,1], linestyle = '--')

plt.xlabel("False positives")
plt.ylabel("True postives")

plt.plot(fpr, tpr, marker ='.', color ='orange')

Out[23]: [<matplotlib.lines.Line2D at 0x2149fc44bc8>]
```

<u>Conclusions:</u>

The accuracy of prediction between healthy and diseased was found to be more than 90%. The results showed that SVM proved to be a very good classification tool between diseased and healthy. These results can be further improved by taking more varied variety of images. Here we have only used four major features for the classification of sample but it can further be furnished using more such significant feature of a leaf image.

<u>Reproducibility</u>

The pipeline can further be extended to find out different subtypes of diseases such as fungal, bacterial, viral etc.