*Machine Learning Based Peanut Maturity Classification from Hyperspectral Image*

Sree Nirmillo Biswash Tushar

Department of Electrical and Computer Engineering, Texas State University,

San Marcos, TX-78666, United States of America

***Abstract-* The maturity of the seed is key information to ensure quality of the crops and better economic returns. But the assessment of the maturity of the peanut requires exocarp removal which suffers from observer’s color assessment skill and experience. Moreover, it requires a great amount of time for a large number of peanut and often ends up in a blasting peanut pod. In order to find an optimal solution, researchers tried to apply digital image processing method. Although this method does not cause pod blasting and does not require personal inspection, the method demands exocarp removal. Recently, a research group has shown why traditional RGB image fails to classify peanut maturity and they have introduced a hyper-spectral unmixing based classifier to solve the problem. In this project, I have worked the same problem with logistic regression, random forest and support vector machine to compare their performance with the previous classifier.**

Keywords ***– Machine Learning, Hyperspectral image, Peanut Maturity***

1. INTRODUCTION

Peanut is an important source of protein which mainly grows in tropical and subtropical region. In order to maximize the peanut production, maturity level of the peanut pod is one of the most important things to know. But, peanut pod maturity level changes over time. If the pod is immature, it will not provide optimum level of performance in terms of seed quality, grade and flavor. On the contrary, over-mature peanut pod can cause aflatoxin contamination [1].

Researchers have developed various methods for evaluating peanut pod maturity. The most recognized method for peanut maturity level classification is MPB (Maturity Profile Board) which maps the color of the mesocarp to the five different maturity categories and the shades within the color into different subcategories [2]. But this method needs exocarp removal and it often causes pod blasting. Since, immature pods are fragile compared to mature pods, most immature pod blows apart. Focusing on not destroying too many immature pods consumes a considerable amount

of time. Moreover, color categorization using MPB requires human visual inspection which is very subjective to human visual condition, lighting condition, observer skill in discriminating color that leads to the possibility of large error. This is very time consuming when one has to classify large number of pods.

In 2017, an objective image processing-based pod maturity classification was proposed by [3] where authors map the median value of the RGB color of the peanut mesocarp to the peanut maturity using Nearest Neighbors classifier. But this method has some serious disadvantages. First, R, B, G color indexes can’t discriminate peanut in terms of maturity level [4]. Second of all, it still requires pod blasting and exocarp removal. The idea of non-destructive peanut maturity classification is initiated by authors of [5], who show a positive correlation between tannin and maturity level. Interestingly, hyperspectral reflectance also varies with different chemical composition. Therefore, HSI image can be used as a peanut pod tannin distinguisher and hence their maturity level. Using all the information, authors in [4] established a linear unmixing model and fully constrained least square algorithm to classify mature and immature pods. They have made the classification based on the multispectral reflectance of the pericarp which does not require exocarp removal. They have also shown that visible part of the spectrum is indistinguishable for mature and immature pod.

The goal of my term paper project is to explore logistic regression, support vector machine and random forest classifier separately to classify peanut pod maturity and analyze the performance of those classifier with a number of evaluation metrics (such as: roc\_auc score, precision score, overall accuracy, balanced accuracy). Lastly, the feasibility of those 3 classifiers will be stated by comparing its performance with the performance of the zare *et. al.* classifier [4].

1. PROCEDURES
2. Data Preprocessing

All the datapreprocessing steps are shown in figure-1 as a part of the methodology. Since each image file contains 15 images total, firstly, the peanut pixels are segmented from the background. Calibration panels and non-uniform background were removed by cropping beyond their location. Some resultant peanut pixel has been assigned as background and therefore, a filter bank has been used to do low pass filtering to solve this problem. Then, each peant’s pixel has been identified and labeled. After that, by averaging the spectrum of each peanut’s pixel, average spectrum for each peanut has been estimated. Lastly, a dataframe has been created for all the peanut spectrum along with the peanut’s maturity label. Then, this dataframe will be processed again with standard machine learning approach and classified with logistic regression, support vector machine and random forest. For HSI data reading, spectral library that has been developed by MIT is used [7]. Along with this library, other standard python library such as numpy, pandas and scikit learn also has been used.

Diagram

Description automatically generated

Fig. 1 Training and Testing Procedures

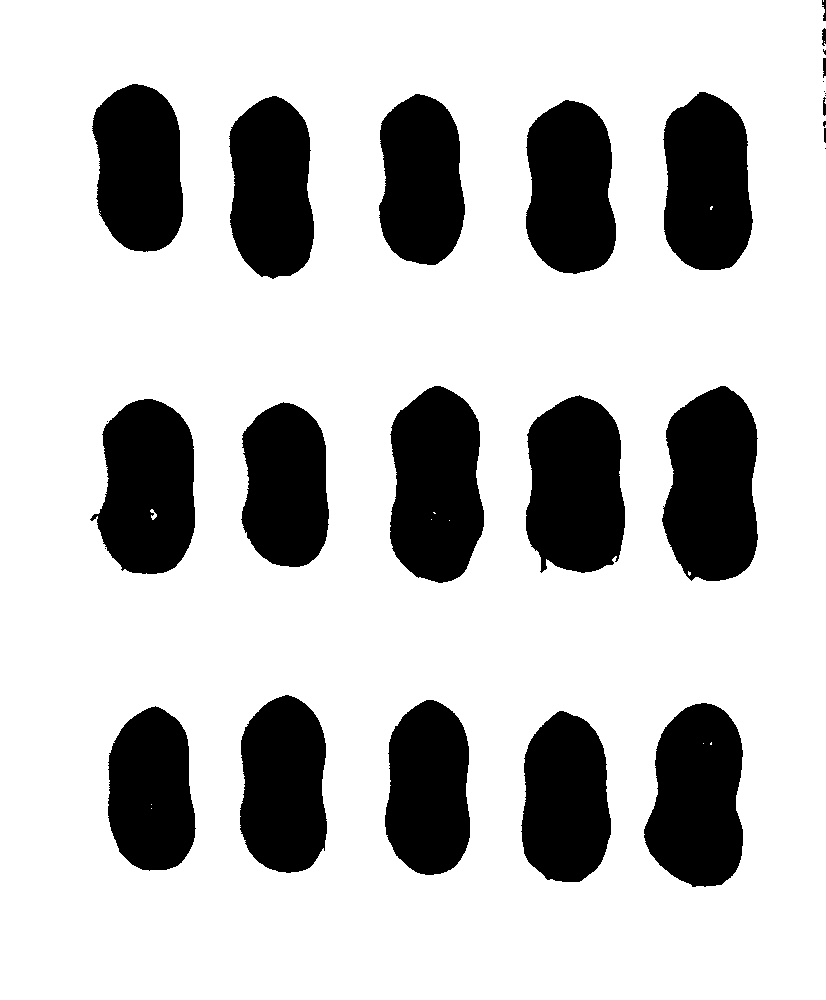
1. *Image segmentation*

First of all, reflectance of Red ,Green and Blue wavelength data was collected and with those information, an RGB image was formed. For red, green and blue channel, 450 nm, 550 nm and 650 nm wavelength were considered. Two methods has been used for segementaion – k-means thresholding [8] and ostu thresholding [9]. They both perform better for those image where peanuts are separated from each other (figure-2). In figure 2, black pixels are peanut whereas the while pixel’s are background and therefore segmented images are binary image. But when two peanuts shadow overlap, it becomes difficult to separate the peanuts from the background as shown in figure-3.



 (a)

Icon

Description automatically generated (b) (c)

1. *Image Filtering*

Fig. 2 (a) RGB image of the peanut from red, green and blue channel wavelength where peanuts can be segmented easily (b) K-means threshold image mask (leaf portion=white (pixel value=1), background=black (pixel value=0)) (c) Ostu threshold image mask

 (a)

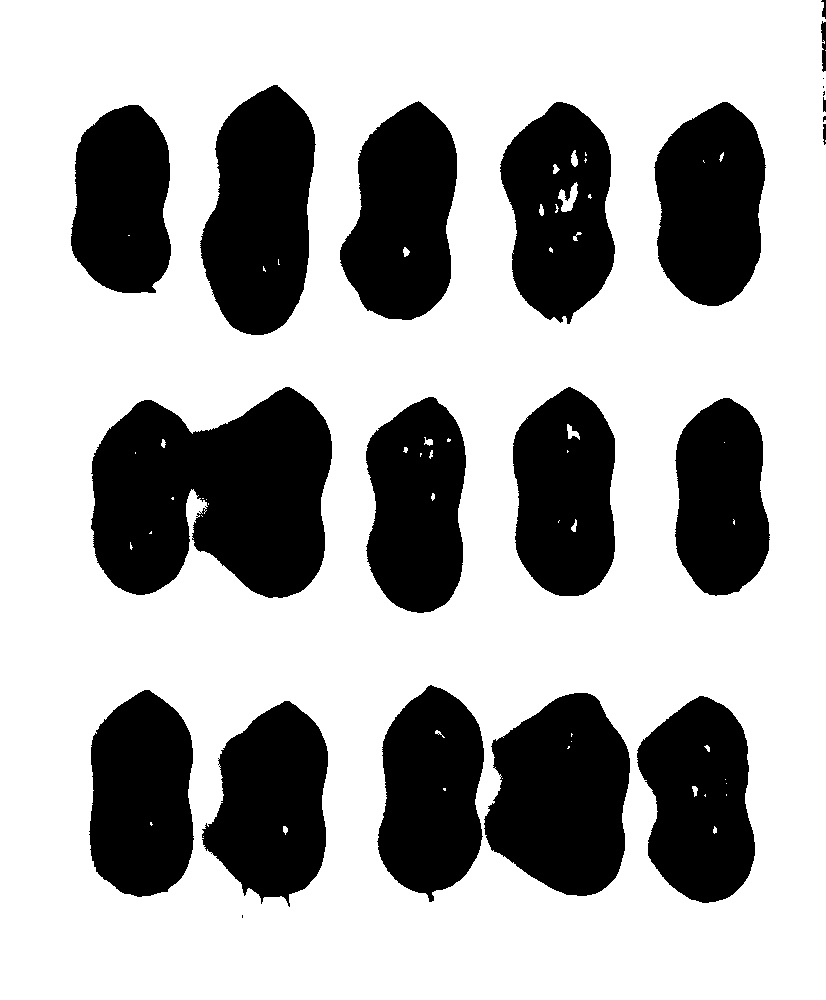
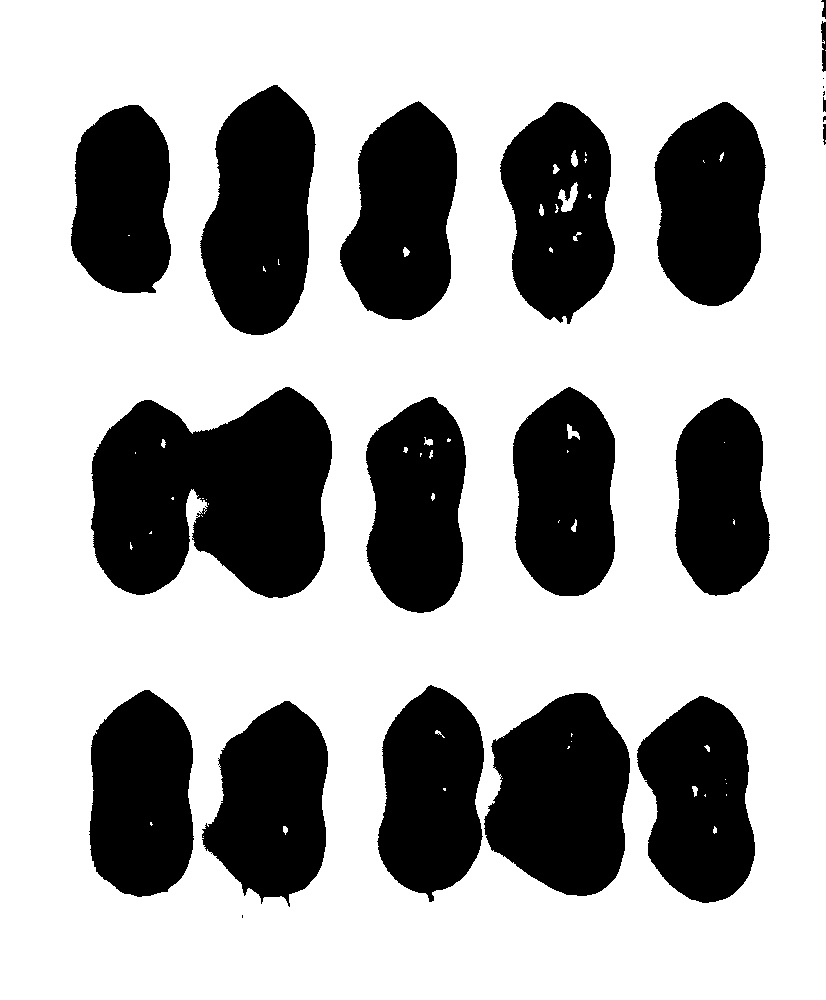
 (b)  (c)

Fig. 3 (a) RGB image of the peanut from red, green and blue channel wavelength where peanuts cannot be segmented easily (b) K-means threshold image mask (leaf portion=white (pixel value=1), background=black (pixel value=0)) (c) Wheat leaf Binary image with spot removal (d) Ostu threshold image mask

From figure 2 and 3, it is evident that some of the segmented peanut’s region has been labeled as background (some while pixel in peanut region). In order to tackle this problem, a random median filter bank was chosen and again, the chioce of the filter is based on experiment. The filter bank consists of 5 median morphological filter with disk size 10,10,10,15 and 20 respectively. Most of the cases it remove those white spots (figure-4) in the peanut pixel but since median filter bank is a low pass filter, it removes the edges and sometimes can make segemention worse (figure-5). Therefore, those cases where filtering does a worse job filtering was not used. For an example, for cases in figure-5, thresholded image is used instead of filtered image.

1. *Indentify each peanut from the image:*

The identifiaction of each peanut from those images is a tricky task. The pixel position of one peanut is far from other peanuts and therefore, k-means clustering was algorithm once gain used to identify each image. The algorithm that has been used is shown in figure-6 (b). Figure 6(a) is showing the identified peanut. The convention of reading peanut number is given in the figure caption.

1. *Calculate the average spectrum of each peanut and create a dataframe with their maturity label:*

In this stage, average spectrum of each peanut is calculated by averaging spectrum from each peanut from section 2. Section 2 only returns position of each peanut in the image. Therefore, in this section, average reflection at each peanut position is first calculated for each wavelength. This gave us an array of shape (Wavelength,Peanut Number, Average Reflection) and from therefore, for each peanut number average Reflection is calculated for all wavelengths (specctrum). The whole procedure is shown in figure -7.

*B. Create a Dataframe:*

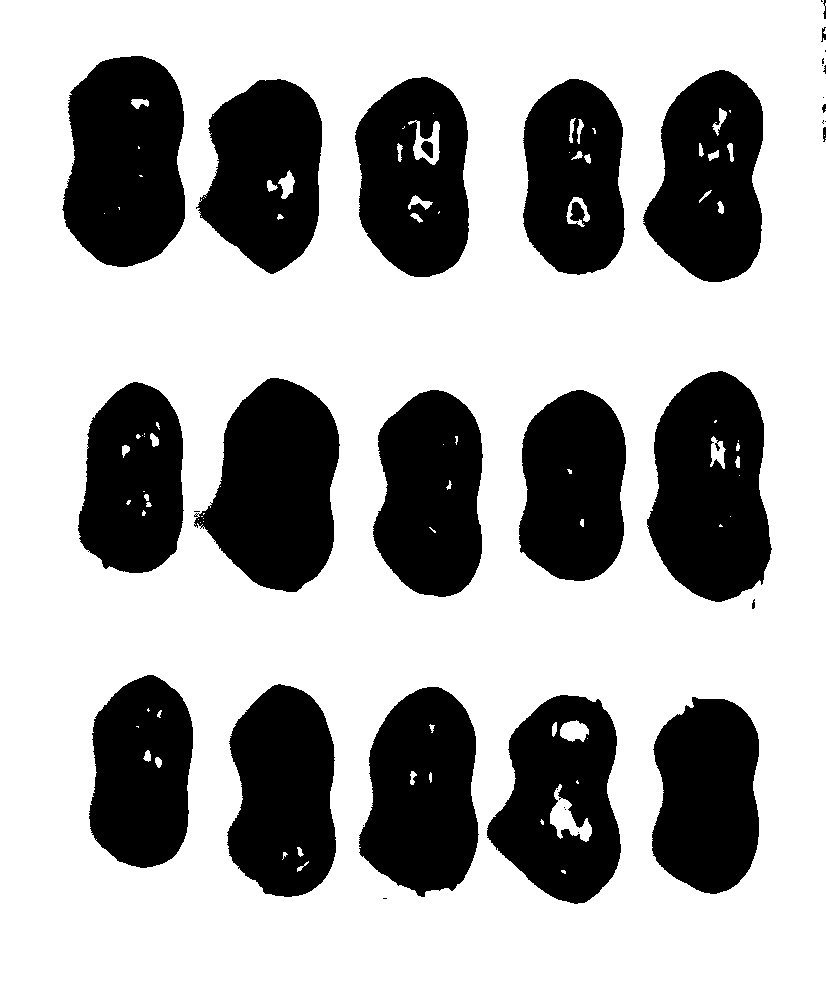
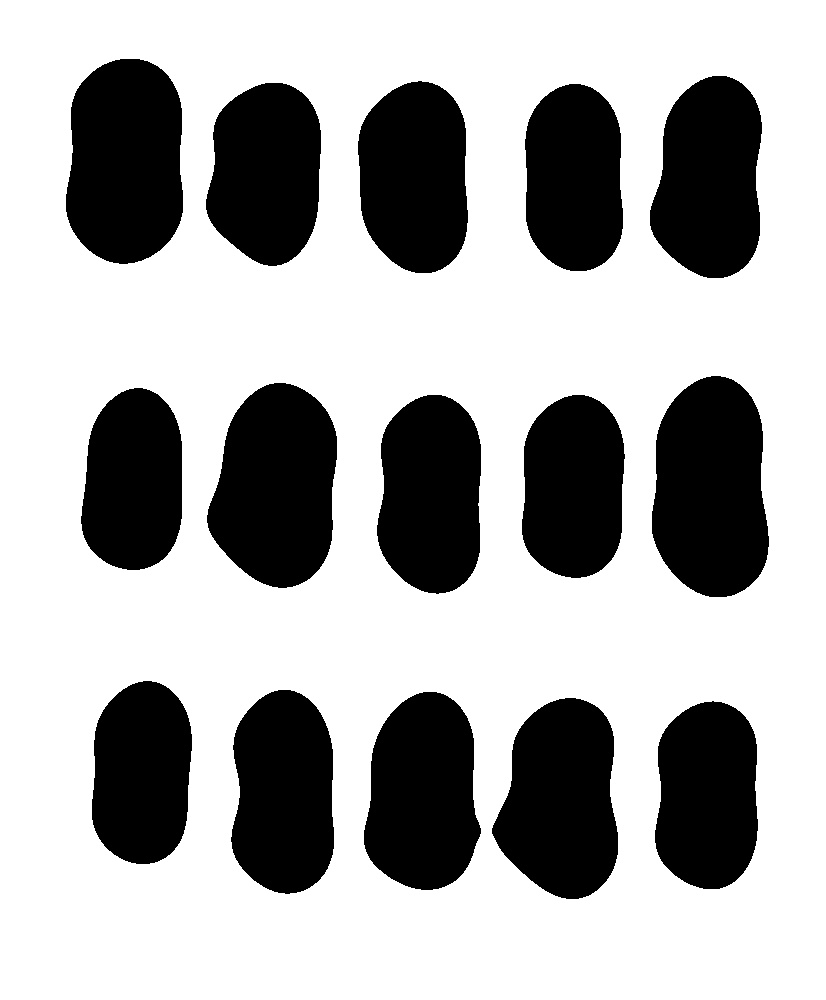
Finally a dataframe has been created where Features are the spectrum of each peanut witheach peanut’s label. The final dataframe is of shape (number peanuts, number of wavelength).

*C. Classifier selection , Classification and Analysis:*

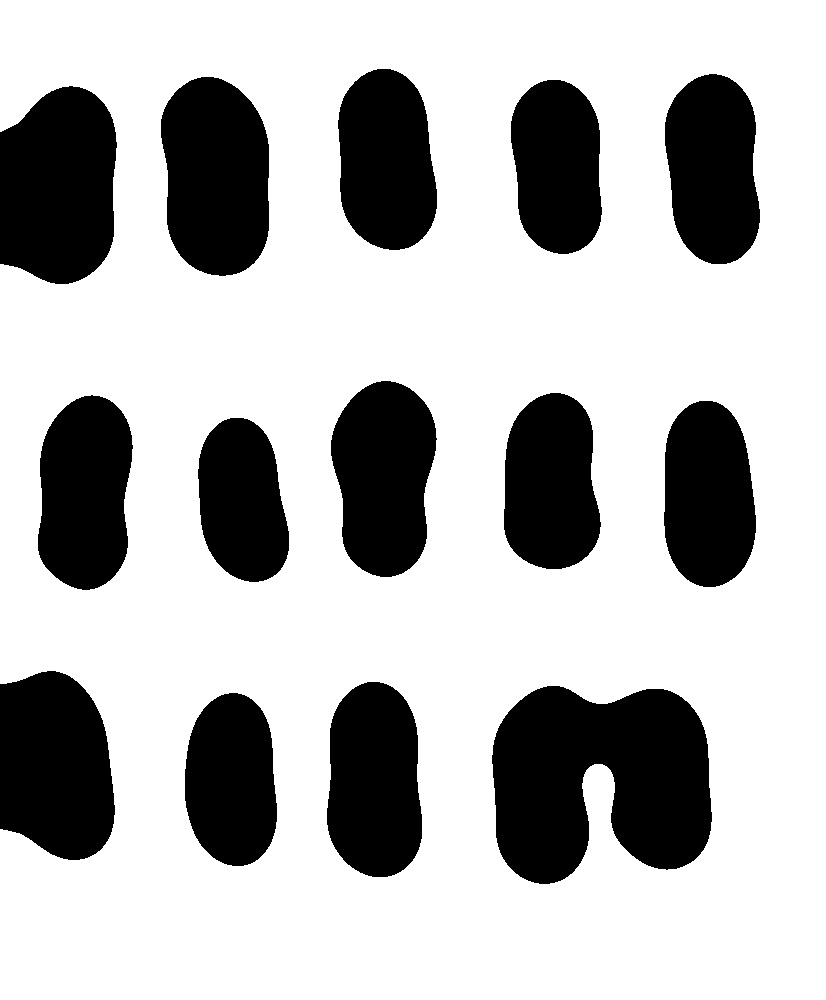
Lastly, a suitable classifier is chosen and hyperparameters are tuned to find out best model. The procedure followed will be discussed in the result and analysis section.

1. DATASET

The dataset used by [4] were collected from the field experiment of the North Florida Research and Education Centre. In 2016 and 2017, total five and seven cultivators, with 3 replicated plots for each cultivator, were used respectively for experiment. In my project, I have just used 2016 dataset due to the fact of large size of data file. The name of the cultivators used in 2016 are TUFRunner 511, FloRun 157, Georgia-06G and TUFRunner 257, FloRun 331. Total 200 samples were randomly selected from 15 plots of 2016 and again 15 samples from each plot were used to capture HSI image. Thereby, the final dataset contained 225 (5×3×15) peanut pod HSI images from 2016 sample collection. In the provided dataset, some data were found corrupted and therefore, I could only read 205 peanut image. The HSI images were taken using pushbroom line-scan hyperspectral imaging system. Then, all images were denoised using 4th order Savitzky eGolay filter and the resultant images had the shape of 1376×467 for the spectral range of 400nm-1000nm with 1.4 nm spectral resolution. The provided dataset has two files for each 15 peanuts (one header file with an extension .hdr and one data file with an extension .hyp). The quantization of each image was 32 bit and therefore every reflectance value returns 4 bytes.

 (a)  (b)

*Fig. 4 (a) Mask of the peanut region segmented by ostu thresholding of* TUFRunner 511 *replication 1 (b) Filtered image of the threshold image. Low pass filtering has smoothen the edge but fills those white spots in peanut region*

 (a)  (b)

*Fig. 5* (a) *Mask of the peanut region segmented by ostu thresholding of* TUFRunner 511 *replication 3 (b*) *Filtered image of the threshold image. Low pass filtering makes it difficult to separate some peanuts.*

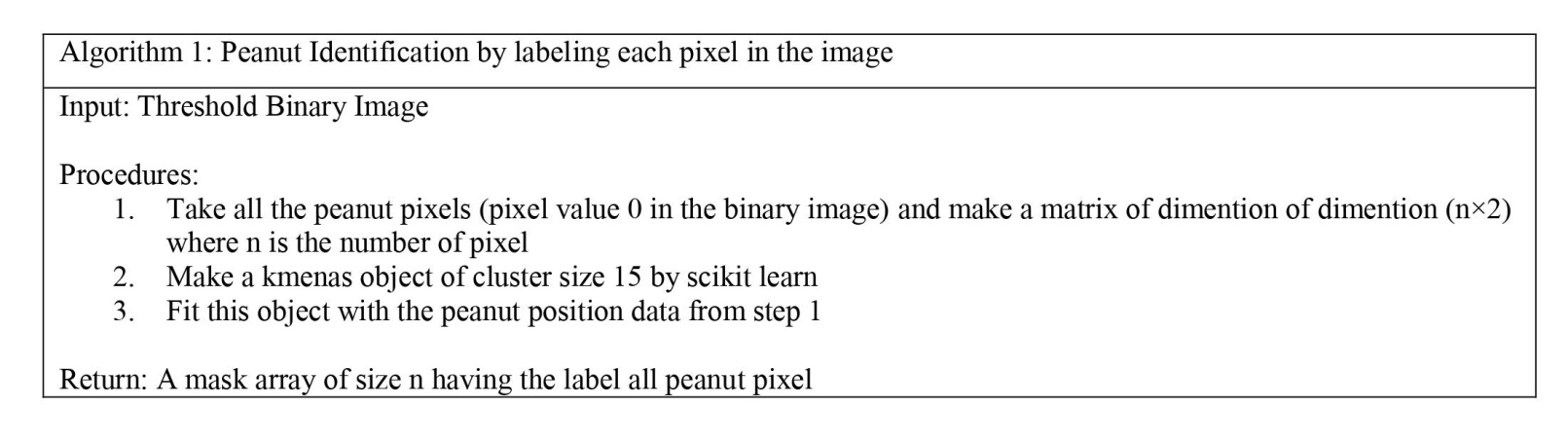
1. RESULT AND ANALYSIS
2. Peanut Data Visualization in spectral domain

According to [4], in the near infrared region (around 600 nm to 900 nm), reflectance from peanut’s mesocarp layer is higher for the immature peanut compared to the mature peanut. At around 900 nm, reflectance value for both classes of peanut cross each other and the mature peanut starts to dominate. But, the curve that I have obtained through rigorous image processing process shows that mature peanut reflects more light in the whole infrared region from 900-970 nm approximately (figure-8). This particular problem can arise either from poor segmentation of the peanut or from the step of replacing the corrupted byte by its closest neighbor. Since, the difference in reflectance is very small, any approximation can flip the curve. This also dictates the importance of higly accurate image processing to tackle this problem.

Chart, shape, bubble chart

Description automatically generated

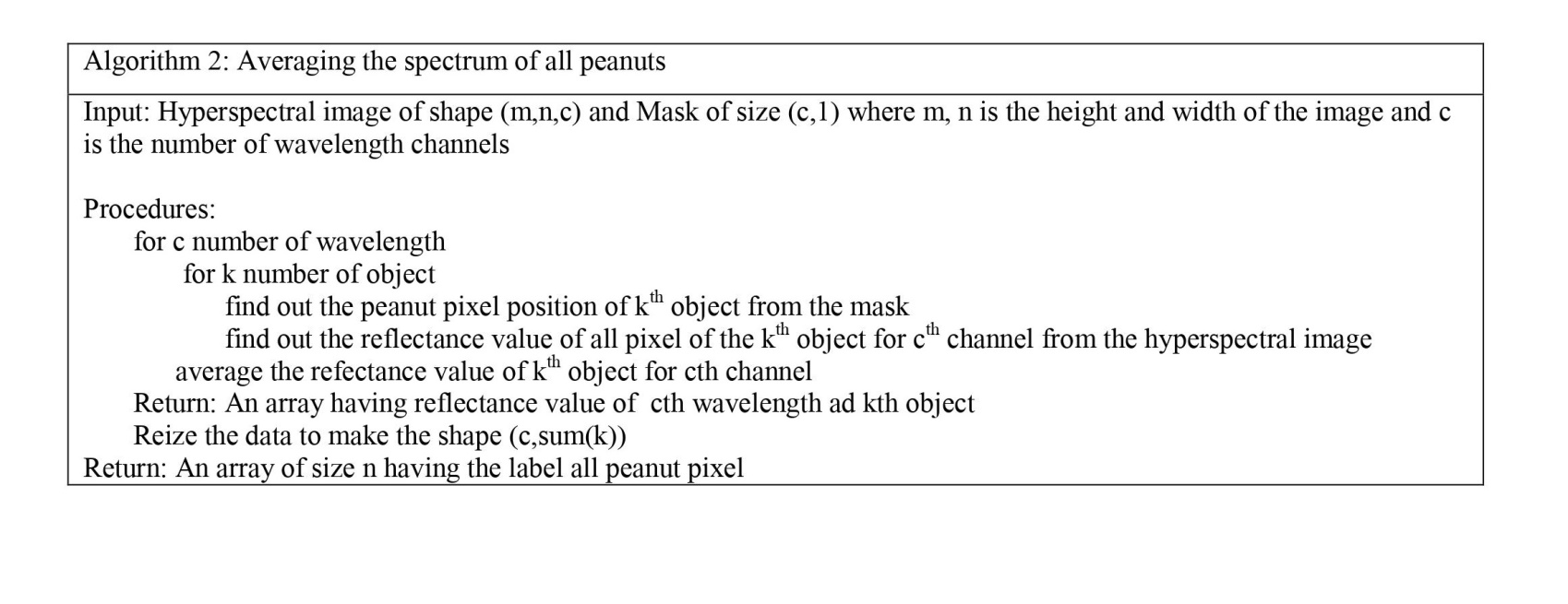
(a)



(b)

*Fig. 6:* (b) *Algorithm for each peanut identification by their position pixel, (a) Figure of all identified image, Red (1-5)*

*Square, Circle, Cross, left arrow and right arrow refines 1,2,3,4,5 respectively. Same Convention for other colors*



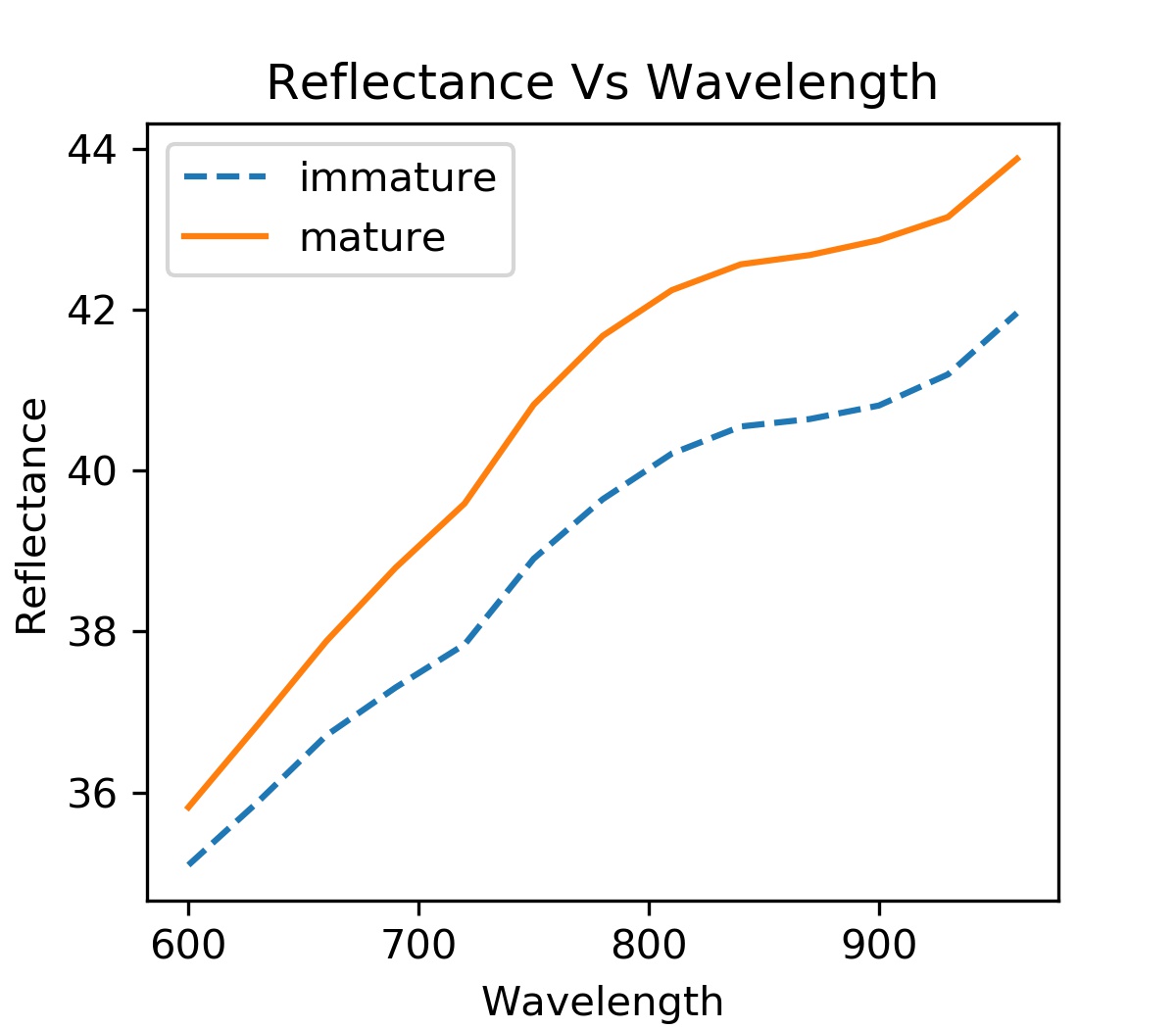
*Fig. 7:* *Algorithm for creating an average spectrum of all peanuts*

This random behaviour becomes more evident when examining individual cultivars’ graph. For some peanut cultivars (FloRun 157, FloRun 331, Georgia-06G), the mature peanut provides more reflection than immature peanut. Wheres for Tufrunner 297 and Tufrunner 511 cultivars, the mesocarp layer of immature peanuts are more reflective. When individual peanut is examined, some unexpected behaviour is obsered. For some peanuts, the reflectance decreases with wavelength (figure-9). The decreament of relectance with wavelength seems ambigious but defraction may play a great role here due to rough surface. Lastly, the reflectance value of the peanut has a very large variance which hinders to fix a threshold for classification. Therefore, it is very difficult to classify peanut without identifying the signature having a clear constrast between two classes.

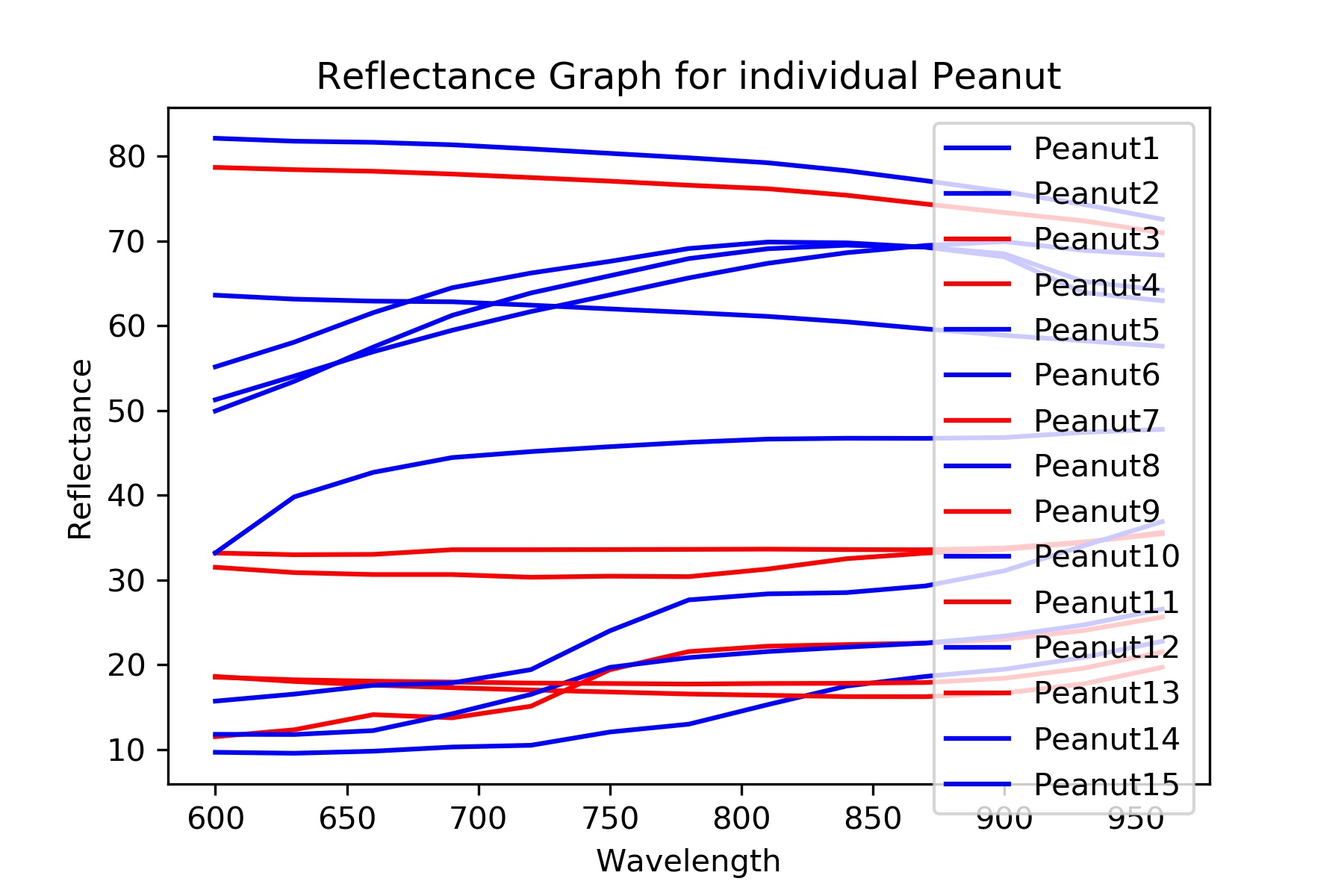
1. *Standardization and Dimensionality Reduction:*

The dataset is standardized using scikit learn class StandardScaler. The standard scaling showed a great difference in the performance. Later, the dimension of the dataset is reduced to 1 by linear discernment analysis (LDA). Since the number of class is 2 here, it is not allowed

to reduce the dimension greater than 1. Principal component analysis is also tried but it gives a relatively worse Performance compared to LDA.



*Fig. 8. Average Reflectamce Vs Wavelength curve for all mature and immature peanuts.*



*Fig. 9. Average Reflectamce Vs Wavelength curve for all peanuts of cultivars TrufRunner 51. Red color refers to immature, Blue color refers to mature*

1. Model selection:

In this project, 3 standard models – random forest, logistic regression and support vector machine – were tested. There is no standard method for finding out the best set of hyperparameters to describe the training dataset. Sometimes, hyperparameter optimization requires complex heuristic optimization method to converge the solution in a very small region and then using grid search to find the optimum model. But, this course covers hyperparameter tuning rather than hyperparameter optimization. Therefore, I have figured out the region of the highly likely best hyperparameters by rigorous random search. Then, I have created a grid of hyperparameters using GridSearchCV function of scikit learn library and evaluate the performance of the model for each set the hyperparameters in order to find the best model with its’ hyperparameters.

Random Forest classifier can be considered as a maximum voting classifier consisted of a number of decision tree classifiers. The number of decision tree is termed as n\_estimator by scikit learn. Random Forest classifier chosen for this classifier since it is effective for smaller training dataset. Other important hyperparameters that are considered to tune are maximum depth of the tree (max\_depth), minimum number of leaves at the end node (min\_sample\_leaf), maximum number of features (max\_features) and minimum number of samples needed to split a node (min\_sample split). The term in the bracket is used by scikit-learn. As described before, first of all, expected effective range of the hyperparameter is evaluated by random search. The range of the hyperparameters is given in the appendix. Lastly, a grid of the hyperparameters are created using GridSearchCV and best set of hyperparameters are found out using 10 fold cross validation as a validation method. The best model was found to provide 65.03% training accuracy and 67.74% testing accuracy. The best random forest model is given below:

RandomForestClassifier(max\_depth=2,max\_features=None, min\_samples\_leaf=3,min\_samples\_split=8,n\_estimators=5)

max\_features=None means all features. Other input arguments are default argument.

Same methodology is applied to the Support Vector machine and Logistic regression model. The best support vector machine model was found to be:

svm.SVC(C=10, gamma=10.0, kernel='rbf’)

Here C is the inverse of the regularization term, kernel was chosen as ‘rbf’ and gamma defines compactness of the decision boundary. Using those parameters, training and testing set accuracy was found to be 67.13% and 58.02%.

On the other hand, the best logistic regression model was found to be:

LogisticRegression(C=0.0001, penalty='l2',solver='liblinear', tol=0.0001)

Training set accuracy: 62.23%, Testing set accuracy: 69.35%

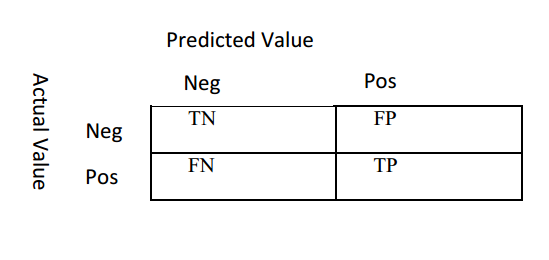
Lastly, a maximum voting approach is taken where all the 3 classifiers were included. It appears to me that the misclassified data were almost same for all 3 classifiers and therefore, in maximum voting did not improve the result (training set accuracy: 65.03% and testing set accuracy: 67.74%). All of those results are shown in table-1:

1. TRAINING AND TESTING DATASET ACCURACIES FOR DIFFERENT CLASSIFIERS

|  |  |  |
| --- | --- | --- |
| Classifier | Training Accuracy | Testing Accuracy |
| Random Forest | 65.03% | 67.74% |
| Support Vector Machine | 67.13% | 58.02% |
| Logistic Regression | 62.23% | 69.35% |
| Maximum Voting | 65.03% | 67.74% |

1. Evaluation Method:

Total 4 evaluation metrics (overall accuracy, precision score, auc score, balanced accuracy) were used to evaluate the performance of the model. All the metrics can be calculated from confusion matrix. According to scikit-learn convention, a confusion matrix is a 2×2 matrix composed of True Positive (TN), False Positive in the first column and False Negative (FN) and True Positive (TP) in the second row.



All validation metrics formula is listed below:

In order to find the optimum threshold and also determining the model’s performance, ROC\_AUC and Precision Vs Recall curve need to be analyzed. Figure 10 illustrates the ROC curve where x axis is the false positive rate and y axis is the true positive rate. For Higher threshold value, false positive rate and true positive rate are very low.

Chart, line chart

Description automatically generated

*Fig. 10. ROC curve for both training and testing dataset*

Chart, line chart

Description automatically generated

*Fig. 11. Precision VS Recall curve for both training and testing dataset*

This infers that for higher threshold, less number of cases has been predicted positive and therefore, both false positive rate and true positive rate both are low. As the threshold increases, true positive rate increases but false positive rate also increases. Therefore, it is important to find an optimum threshold where there is a balance between true positive rate and false positive rate. This is a requirement that should be determined by the user. It looks like 0.51 threshold works better which gives about 79% true positive rate and 38% false positive rate for training set and 76% true positive rate and 53% false positive rate for testing data.

Figure 10 shows the precision vs recall curve. Precision and recall refer to the percentage of true positive in total predicted positive and total actual positive cases respectively. From figure-10, it is evident that for lower threshold, precision is very high and recall is very low since only a few samples are being predicted as positive and we have put a high condition on predicting something positive. As threshold increases, curve moves downward which means precision decreases and recall increases. Again, user should define the optimum threshold. For, threshold=0.51, 72% precision score and 77% recall score for training dataset and 63% precision score and 76% recall score for testing dataset was obtained.

The performance score from each metrics is demonstrated below in table-2:

1. PERFORMANCE OF THE RANDOM FOREST CLASSIFIER

|  |  |  |
| --- | --- | --- |
| Validation Metrics | Training accuracy | Testing accuracy |
| Overall Accuracy | 0.71 | 0.63 |
| AUC\_ROC | 0.77 | 0.68 |
| Precision | 0.79 | 0.76 |
| Balanced Accuracy | 0.78 | 0.69 |

1. CONCLUSION

The maturity classification of peanut needs accurate image segmentation. K-means and Ostu thresholding both performs better when peanuts are separated by a distance and their shadows do not overlap. The raw spectrum does not look like a good feature that can distinguish mature and immature peanut. Random forest shows better performance that other classifiers. It is possible to improve the model if the feature can be extracted in such a way that shows good contrast between both classes. The deep learning approach will be used in my future work.

ACKNOWLEDGEMENT

I would like to thank Dr. Damien Valles for the course materials that was helpful for the project.

REFERENCES

[1] *florida.growingamerica.com*. [Online]. Available: https://florida.growingamerica.com/news/2015/08/optimizing-peanut-maturity-review-methods-and-techniques. [Accessed: 08-Dec-2020].

[2] Williams, E. J., & Drexler, J. S. (1981), “A non-destructive method for determining peanut pod maturity,” Peanut Science, 8(2), 134 e141.

[3] Bindlish, E., Abbott, A. L., & Balota, M. (2017), “Assessment of peanut pod maturity,” In Applications of computer vision (WACV), 2017 IEEE winter conference on (pp. 688 e696), IEEE.

[4] Zou, Sheng, et al., "Peanut maturity classification using hyperspectral imagery," Biosystems Engineering 188 (2019): 165-177.

[5] Sanders, Timothy H., "Changes in tannin-like compounds of peanut fruit parts during maturation," Peanut Science 4.2 (1977): 51-53.

[6] Pan, Bin, Zhenwei Shi, and Xia Xu. "MugNet: Deep learning for hyperspectral image classification using limited samples." ISPRS Journal of Photogrammetry and Remote Sensing 145 (2018): 108-119.

[7] *Spectral Python 0.21 documentation*. [Online]. Available: https://www.spectralpython.net/fileio.html. [Accessed: 08-Dec-2020].

[8] S. Lloyd, "Least squares quantization in PCM," in IEEE Transactions on Information Theory, vol. 28, no. 2, pp. 129-137, March 1982, doi: 10.1109/TIT.1982.1056489.

[9] N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," in IEEE Transactions on Systems, Man, and Cybernetics, vol. 9, no. 1, pp. 62-66, Jan. 1979, doi: 10.1109/TSMC.1979.4310076

Code: All necessary functions:

def BGR\_imread(Wavelenth,Image):

Image\_bgr=[];

Shape=Image.shape;

for i in range(Shape[0]):#1000

for j in range(Shape[1]):#1376

try:

Image\_bgr.append(np.array(Image[i,j,Wavelenth],dtype=np.uint8))

except:

Image\_bgr.append(Image\_bgr[i\*Shape[1]+j-1])

return np.array(Image\_bgr).reshape(Shape[0],Shape[1],3)

def k\_means\_thresh(Image,max\_iter=10,eps=1.0,attempts=10,Condition=3,K=2):

criteria = (Condition, max\_iter, eps) #Setting Criteria as cv2.TERM\_CRITERIA\_EPS + cv2.TERM\_CRITERIA\_MAX\_ITER

Image\_gray = cv2.cvtColor(Image, cv2.COLOR\_BGR2GRAY) #(1000,1376) Row indexing top to bottom, Collumn indexing left to right

Z = Image\_gray.reshape((-1,1)) # Data Shape(1000\*1376,3) m\*n number of pixels

Z = np.float32(Z) #Converting to 32 bit float

ret,label,center=cv2.kmeans(Z,K,None,criteria,attempts,cv2.KMEANS\_RANDOM\_CENTERS)

#ret=total distance from corresponding center, label of each pixel (1000\*1376)

center = np.uint8(center) # (2,1)

res = label.flatten() # label.flatten() is the flatten list of the pixel (1376\*1000) either 1/0 #center[label.flatten()] is the list of center [0] if class 0, center[1] value if class 1

#res2 = res.reshape((Image\_gray.shape))# (1000,1376)

res2=res.reshape((Image\_gray.shape))

return np.array((res2[:,int(res2.shape[1]\*0.4):])\*255,dtype=np.uint8)

def filter\_bank(X):

median1 = filters.median(X, disk(10));

median2 = filters.median(median1, disk(10));

median3 = filters.median(median2, disk(10));

median4 = filters.median(median3, disk(15));

return filters.median(median4, disk(20));

def identify\_peanut\_by\_label(pixel\_position,kmeans):

Colors=('red','green','blue')

markers = ('s', 'x', 'o', '^', 'v')

row=len(np.unique(kmeans.labels\_))//5;

plt.scatter(pixel\_position[:,0],pixel\_position[:,1], c=kmeans.labels\_, cmap='plasma')

for i in range(len(markers)):

plt.scatter(kmeans.cluster\_centers\_[i\*row:(i+1)\*row,0] ,kmeans.cluster\_centers\_[i\*row:(i+1)\*row,1], color=Colors[:row], marker=markers[i])

plt.savefig('peanut.png')

def peanut\_pixel\_position(median5):

Pixel\_Position=[];

for i in range(median5.shape[0]): #Height

for j in range(median5.shape[1]): #Width

if median5[i][j]==0:

Pixel\_Position.append([i,j])

return np.array(Pixel\_Position)

def wavelength\_channel\_conversion(X,option=1):

lemda1=400.450400;

lemda2=401.673000;

if option==1:

return np.int16((X-lemda1)/(lemda2-lemda1))+1;

elif option==0:

return (lemda2-lemda1)\*X+lemda1;

else:

raise TypeError("Only 0 or 1 is allowed")

def Wavelength\_Peanut\_pixel(hyp\_image,Wavelength,Final\_pixel):

Feature\_Mat=[];

Shape=hyp\_image.shape;

for j in range(len(Wavelength)):

Image\_hyp=[];

for m in range(Shape[0]):

for n in range(Shape[1]):

try:

Image\_hyp.append(hyp\_image[m,n,[Wavelength[j]]])

except:

Image\_hyp.append(Image\_hyp[np.int64(m\*Shape[1]+n-1)])

Image\_hyp=np.array(Image\_hyp).reshape(Shape[0],Shape[1])

Feature\_label=[];

for i in range(len(Final\_pixel)):

Feature=[];

for k in range(len(Final\_pixel[i])):

Feature.append(Image\_hyp[Final\_pixel[i][k][0],Final\_pixel[i][k][1]])

Feature\_label.append(np.array(Feature))

Feature\_Mat.append(Feature\_label)

return Feature\_Mat

#Image Processing Code ##1\_import all library

import spectral.io.envi as envi #To read spectral data

import os

import numpy as np

import cv2

from skimage import filters

from skimage.morphology import disk

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

import pandas as pd

from all\_function import \* #My all function

from sklearn.preprocessing import StandardScaler

from sklearn import svm

##2\_Image Segementaion

#1\_File read

file="1A\_Side1";

path="E:\\Spring\\EE 354\\HW12";

hdr\_file\_name="1A\_1-15\_Side1\_20160914\_0001\_cnbh (1376x1000x467).hdr";#It contains the information of which bands are used,

#height and width of the image,

hyp\_file\_name="1A\_1-15\_Side1\_20160914\_0001\_cnbh (1376x1000x467).hyp";#It containts data of all bands

hyp\_image = envi.open(os.path.join(path,hdr\_file\_name), os.path.join(path,hyp\_file\_name))

#hyp\_image #Data description #BIL=Band Interleaved Image

#Image Show

Number=200;

S\_image=np.squeeze(hyp\_image[:,:,Number]);

cv2.imshow("image",S\_image/255)

cv2.waitKey(0)

file1="Single\_Wavelength\_"+str(file)+".jpg"

cv2.imwrite(file1, S\_image)

#2\_Finding out the R,G,B channel number and create an RGB image

wave\_length=np.array([450,550,650]); #R,G,B Wavelength

BGR=wavelength\_channel\_conversion(wave\_length,option=1) #Convert wavelength to the index of the wavelength data

Image=BGR\_imread(list(BGR),hyp\_image) #Read RGB image, 8 bit, shape(1376,1000,3), Replace nth pixel by n-1 th pixel

#when can not read

file1="Raw\_Image\_"+str(file)+".jpg"

cv2.imwrite(file1, Image)

#Image Show

cv2.imshow("image",Image)

cv2.waitKey(0)

#3\_K\_means and ostu thresholding method peanut\_black

Image\_kmeans=255-k\_means\_thresh(Image,max\_iter=10,eps=1.0,attempts=10,Condition=3,K=2) #k\_means\_thresh is a function

#does k\_means thresholding using opencv2 kmeans

file2="k\_means\_Threshold\_"+str(file)+".jpg";

cv2.imwrite(file2, Image\_kmeans)

Image\_gray = cv2.cvtColor(Image, cv2.COLOR\_BGR2GRAY) #For ostu thresholding image has been converted to grayscale first

thresh, im\_bw = cv2.threshold(Image\_gray, 0, 255, cv2.THRESH\_OTSU) #Binary Ostu threshold image

Image\_ostu=np.array(im\_bw[:,int(im\_bw.shape[1]\*0.4):]) #Croped calibration panel

file3="Ostu\_Threshold\_"+str(file)+".jpg";

cv2.imwrite(file3, Image\_ostu)

#cv2.imshow("image\_kmeans",Image\_kmeans)

cv2.imshow("image\_ostu",Image\_ostu)

cv2.waitKey(0)

#4\_Applying Filter

Filtered\_image\_kmeans=filter\_bank(Image\_kmeans)

file4="Image\_Fil\_kmeans\_"+str(file)+".jpg";

cv2.imwrite(file4, Filtered\_image\_kmeans)

file5="Image\_Fil\_ostu\_"+str(file)+".jpg";

Filtered\_image\_ostu=filter\_bank(Image\_ostu)

cv2.imwrite(file5, Filtered\_image\_ostu)

#5 Separating Peanuts

Peanut=15;

pixel\_position=peanut\_pixel\_position(Filtered\_image\_ostu); #number of peanut pixel 2

kmeans = KMeans(n\_clusters=Peanut, random\_state=0).fit(pixel\_position);

#identify peanut by sign and color

# Red (1,5) Green (6,10) Blue (11,15) 's', 'x', 'o', '^', 'v'

#So, Red square is 1, Green Square is 6, Blue Square is 11

#So, Red X is 2, Green X is 7, Blue X is 12

identify\_peanut\_by\_label(pixel\_position,kmeans)

Final\_pixel=[pixel\_position[np.where(kmeans.labels\_==i)] for i in range(Peanut)];#Arrange peanut pixel according to their label

#such as peanut 0's pixel first,peanut 1's pixel second

##3\_Wavelength\_Peanut\_pixel

Wavelength=wavelength\_channel\_conversion(np.arange(600,970,30),option=1); #Wavelength to index

WPP\_Mat=Wavelength\_Peanut\_pixel(hyp\_image,Wavelength,Final\_pixel) #(#Wavelength,#Peanut,#pixel)

#For wavelength 600 nm, 1st peanut pixel

#WPP\_Mat[0][0]

##4\_DataFrame\_Creation

#1\_Class

DF=pd.read\_csv('Peanut\_Maturity.csv')

Num=np.array([9,6,8,4,15,2,14,12,7,3,13,5,1,10,11]) #peanut number arranged from 1 10 15

assert(len(np.unique(Num))==Peanut)

Group\_new=[];

for i in range (DF.shape[0]//15):

Group\_new.append([DF['Group'][int(i\*15)]]\*15)

DF['Group\_new']=np.array(Group\_new).reshape(-1)

Y=DF[DF['Group\_new']=='1A'].iloc[Num-1,2].replace('M',1).replace('IM',0).values

#2\_Feature

X=[];

for i in range(Peanut):

HI=np.array(WPP\_Mat)[:,i];

HP=np.stack(HI,axis=0); # HI data of size (D,N)

X.append(np.mean(HP,axis=1))

New\_dataFrame=pd.DataFrame(X)

#3\_Standardization

#stdc=StandardScaler();

#New\_dataFrame\_std=stdc.fit\_transform(New\_dataFrame)

#4\_Model\_fitting

#sv=svm.SVC()

#sv.fit(New\_dataFrame,Y)

#print(sv.score(New\_dataFrame,Y))

#Feature\_Matrix\_Creation

Last\_DF=pd.concat([New\_dataFrame,pd.DataFrame(Y)],axis=1);

file\_name="5C\_side1.csv"

Last\_DF.to\_csv(file\_name);

Last\_DF.head()

#cv2.imwrite("Inverted\_image.jpg",cv2.bitwise\_not(cv2.imread('Test\_image.jpg')));

#Machine Learning Portion Code

import pandas as pd

import numpy as np

import itertools

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split, cross\_val\_score,GridSearchCV

from sklearn.preprocessing import StandardScaler

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA

from sklearn.ensemble import RandomForestClassifier, VotingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn import svm

from sklearn import metrics

from sklearn.metrics import roc\_auc\_score,roc\_curve,precision\_recall\_curve,confusion\_matrix,average\_precision\_score,balanced\_accuracy\_score

#from sklearn.preprocessing import MinMaxScaler

#from sklearn.decomposition import PCA

#from sklearn.pipeline import make\_pipeline

#from sklearn.decomposition import KernelPCA

#from sklearn.metrics import roc\_curve,precision\_recall\_curve

#from sklearn.metrics import confusion\_matrix

#from sklearn.model\_selection import cross\_val\_score

#from sklearn.metrics import average\_precision\_score

#from sklearn.model\_selection import GridSearchCV

#from sklearn.metrics import balanced\_accuracy\_score

#Crate a dataframe from Peanut\_Maturity.csv and make a column from each peanut's origin (such as 1A)

#Read Peanut Maturity csv file

DF=pd.read\_csv('Peanut\_Maturity.csv')

#New\_peanut grouping\_indexing

Group\_new=[];#New\_peanut grouping\_index

for i in range (DF.shape[0]//15):

Group\_new.append([DF['Group'][int(i\*15)]]\*15) #DF['Group'] [0] has value '1A' (taking every 15th value)

#by DF['Group'][int(i\*15)] and make 15 copy of them

DF['Group\_new']=np.array(Group\_new).reshape(-1,1)#Create a new column for Group\_new

#Arrange the peanut according to the arrangement of the kmeans peanut's classification label

#Index of all peanut in the indentified peanut image, for an example, in 1A image, there are 15 peanuts, first peanut identified

#has been labeled as 1, second as 15,...

#Num\_1 has the label of 1A,1B,1C

Num\_1=np.array([[1,15,14,6,3,7,9,13,11,10,4,5,2,8,12],[4,9,10,8,3,7,2,6,5,1],[3,1,10,9,6,14,5,11,2,13,12,7,4,15,8]]);

Num\_2=np.array([[4,9,7,1,10,13,2,11,3,14,6,15,8,5,12],[3,13,2,1,4,7,11,15,8,10,9,5,6,12,14],[4,13,10,12,7,8,14,3,5,11,1,6,15,2,9]])

Num\_3=np.array([[7,2,4,1,6,9,13,11,8,3,5,15,14,10,12],[1,11,5,2,14,9,3,13,4,6,15,7,8,12,10],[5,10,8,1,9,6,2,3,4,7]])

Num\_4=np.array([[13,10,15,9,4,2,6,1,11,12,8,5,7,14,3],[2,4,5,13,9,12,15,8,1,6,10,7,3,14,11],[2,9,4,1,6,13,11,10,8,12,5,15,7,3,14]])

Num\_5=np.array([[13,7,11,1,5,2,15,6,8,10,9,4,3,14,12],[4,9,1,5,3,6,2,7,10,8],[1,5,4,3,7,8,6,10,9,2]])

#Assert will give an error if 2 peanut has been labeled same

assert(set([len(np.unique(m)) for m in Num\_1]) == {15} or {10} or {10,15})

assert(set([len(np.unique(m)) for m in Num\_2]) == {15} or {10} or {10,15})

assert(set([len(np.unique(m)) for m in Num\_3]) == {15} or {10} or {10,15})

assert(set([len(np.unique(m)) for m in Num\_4]) == {15} or {10} or {10,15})

assert(set([len(np.unique(m)) for m in Num\_5]) == {15} or {10} or {10,15})

#Num is a list of all Peanut label

Num=[Num\_1,Num\_2,Num\_3,Num\_4,Num\_5];

#Creating Maturity label and Color label based on Peanut labeling in the indentification image

M\_label=[]; #Maturity label

Color\_label=[];#Color label

label=[];

Group=['A','B','C'];

for i in range(len(Num)):

for j in range(len(Group)):

string=str(i+1)+str(Group[j])#Creating all the string of the peanut image name like 1A,1B,....

label.append(string)#Creating all the string of the peanut image name like 1A,1B,...

All=DF[DF['Group\_new']==string];

M\_label.append(All.iloc[np.array(Num[i][j])-1,2].values)#Separting Peanut maturity based on peanut image name 1A

#and peanut label, like peanuty maturity label of 1A image's 0th peanut

Color\_label.append(All.iloc[np.array(Num[i][j])-1,3].values)#Color label

M\_label\_flatten = list(itertools.chain.from\_iterable(M\_label))#Flatten label list

Color\_label\_flatten = list(itertools.chain.from\_iterable(Color\_label))#Flatten color list

#Make the dataframe having right column name (wavelength), Maturity label and Color label

Wavelength=list(range(600,970,30));#All wavelength

df=pd.read\_csv("1A\_side1.csv") #Reflectance of the 1A peanut image

#Reading all excel file

for i in range(len(label)-1):

new\_file=str(label[i+1])+"\_side1.csv";

df1=pd.read\_csv(new\_file);

df=pd.concat([df,df1],ignore\_index=True)

#Create two columns name Maturity\_label and Peanut\_Color

df['Maturity\_label']=M\_label\_flatten

df['Peanut\_Color']=Color\_label\_flatten

#Replace column name with wavelength

Columns\_name=[(str(y),str(x)+"nm") for (x,y) in zip(Wavelength,df.columns[1:len(Wavelength)+1])] #Create a tuple to

#replace inde 0,1,2 with wavelngth 600nm, .....

df\_final=df.rename(columns=dict(Columns\_name))

Df\_Final=df\_final.drop(columns=[df\_final.columns[0],df\_final.columns[len(Wavelength)+1]])#drop redundant index column and another label column (has some error)

Df\_Final['Maturity\_label']=Df\_Final['Maturity\_label'].replace('M',1).replace('IM',0).values

#df\_final.head()

Df\_Final.head()

#Plot of peanut mature vs immature

plt.figure("General",figsize=(4, 3.5),dpi=300)

plt.plot(Wavelength,np.mean(Df\_Final.iloc[:,:-2][Df\_Final['Maturity\_label']==0].values,axis=0),'--',label='immature')

plt.plot(Wavelength,np.mean(Df\_Final.iloc[:,:-2][Df\_Final['Maturity\_label']==1].values,axis=0),label='mature')

plt.xlabel('Wavelength')

plt.ylabel('Reflectance')

plt.title('Reflectance Vs Wavelength')

plt.legend(loc='upper left')

plt.savefig("All\_peanut.jpg",dpi=300)

#Reflection graph of all species (M vs IM), all replications (for all species) M vs IM graph

Species\_name=['TrufRunner 511','FloRun 157','Georgia-06G','TrufRunner 297','FloRun 331'];

for k in range(len(Species\_name)):

title1='Reflectance Graph for 3 replications of species '+str(Species\_name[k]);

title2='Reflectance Graph for species '+str(Species\_name[k]);

Len=[];

for i in range(5):

for j in range(3):

Len.append(len(Num[i][j]))

#To Get the index of all the peanuts of kth specifies and all 3 replication

Peanut\_Len=np.array(Len); #number of peanut in all images

List=list(range(k\*3,(k+1)\*3)); #For 'TrufRunner 511' List will 0,1,2

xs,ys=np.sum(Peanut\_Len[:List[0]]),np.sum(Peanut\_Len[:List[2]+1]); #Taking the number of peanuts for species

#'TrufRunner 511' for an instance

#Peanut number for every replication

Rep\_xy=list(Peanut\_Len[range(k\*3,(k+1)\*3)]);

Rep\_xy.insert(0,0);

X\_point=xs;

plt.figure(k,figsize=(12, 6),dpi=300)

plt.subplot(1,2,1)

for i in range(3):

X\_point=X\_point+Rep\_xy[i]; #X coordinate of ith replication

Y\_point=X\_point+Rep\_xy[i+1]; #Y coordinate of ith replication

#Plot mature vs immature

plt.plot(Wavelength,np.mean(Df\_Final.iloc[X\_point:Y\_point,:-2][Df\_Final['Maturity\_label']==0].values,axis=0),'--',label='immature\_Replication\_'+str(i+1))

plt.plot(Wavelength,np.mean(Df\_Final.iloc[X\_point:Y\_point,:-2][Df\_Final['Maturity\_label']==1].values,axis=0),label='mature\_Replication\_'+str(i+1))

plt.xlabel('Wavelength')

plt.ylabel('Reflectance')

plt.title(title1)

plt.legend(loc='upper right')

plt.subplot(1,2,2)

#M Vs IM curve for all peanut of a single species

plt.plot(Wavelength,np.mean(Df\_Final.iloc[xs:ys,:-2][Df\_Final['Maturity\_label']==0].values,axis=0),'--',label='immature')

plt.plot(Wavelength,np.mean(Df\_Final.iloc[xs:ys,:-2][Df\_Final['Maturity\_label']==1].values,axis=0),label='mature')

plt.xlabel('Wavelength')

plt.ylabel('Reflectance')

plt.title(title2)

plt.legend(loc='upper right')

plt.savefig(str(Species\_name[k])+".jpg",dpi=300)

#Plot of all individual peanuts of 'TrufRunner 511' replication 1

plt.figure("Individual Peanut",figsize=(6,4),dpi=300)

#Taking first 15 peanuts

for i in range(15):

if Df\_Final.iloc[i,-2]==0:

color='red';

else:

color='blue';

plt.plot(Wavelength,Df\_Final.iloc[i,:-2].values,color=color,label="Peanut"+str(i+1))

plt.xlabel('Wavelength')

plt.ylabel('Reflectance')

plt.title("Reflectance Graph for individual Peanut")

plt.legend(loc='upper right')

plt.savefig("1A.jpg",dpi=300)

#Data splitting

X=Df\_Final.iloc[:,:-2].values;

Y=Df\_Final.iloc[:,-2].values;

Test\_size=0.30;

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size=Test\_size, random\_state=1, stratify=Y);

#Standardization

stdc=StandardScaler();

X\_train\_std=stdc.fit\_transform(X\_train);

X\_test\_std=stdc.fit\_transform(X\_test);

#Dimentionality reduction

lda=LDA(n\_components=1);

X\_train\_lda=lda.fit\_transform(X\_train\_std,y\_train);

X\_test\_lda=lda.fit\_transform(X\_test\_std,y\_test);

#Fitting Model

# Random Forest

param\_grid\_rf = {'bootstrap': [True],'max\_depth': [2, 3, 4],'min\_samples\_leaf': [3, 4, 5],'min\_samples\_split': [2, 4, 8],'n\_estimators': [5, 10, 20, 50],'max\_features':[None]}

rf = RandomForestClassifier()

grid\_search\_rf = GridSearchCV(estimator = rf, param\_grid = param\_grid\_rf, cv = 10, n\_jobs = 2, verbose = 0)

grid\_search\_rf.fit(X\_train\_lda, y\_train)

best\_grid\_rf = grid\_search\_rf.best\_estimator\_

print(best\_grid\_rf.score(X\_train\_lda,y\_train))

print(best\_grid\_rf.score(X\_test\_lda,y\_test))

#Fitting Model

# SVM

param\_grid = {'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],

'kernel': ['rbf'],'gamma': [1e-3,1e-2, 1e-1, 1e0, 1e1, 1e2,1e3]}

sv = svm.SVC()

grid\_search\_sv = GridSearchCV(estimator = sv, param\_grid = param\_grid, n\_jobs=3, cv = 10, verbose = 0)

grid\_search\_sv.fit(X\_train\_lda, y\_train)

best\_grid\_sv = grid\_search\_sv.best\_estimator\_

print(best\_grid\_sv.score(X\_train\_lda,y\_train))

print(best\_grid\_sv.score(X\_test\_lda,y\_test))

#Fitting Model

# Logistic Regression

param\_grid = {'penalty' : ['l1', 'l2'],'C' : np.logspace(-4, 4, 20),'solver' : ['liblinear'],'tol': np.logspace(-10, -2, 5)}

lr = LogisticRegression()

grid\_search\_lr = GridSearchCV(estimator = lr, param\_grid = param\_grid, cv = 10, n\_jobs = -1, verbose = 0)

grid\_search\_lr.fit(X\_train\_lda, y\_train)

best\_grid\_lr = grid\_search\_lr.best\_estimator\_

print(best\_grid\_lr.score(X\_train\_lda,y\_train))

print(best\_grid\_lr.score(X\_test\_lda,y\_test))

#maximum voting classifier

eclf1 = VotingClassifier(estimators=[('lr', best\_grid\_rf), ('rf', best\_grid\_sv), ('gnb', best\_grid\_lr)], voting='hard')

eclf1 = eclf1.fit(X\_train\_lda, y\_train)

print(eclf1.score(X\_train\_lda,y\_train))

print(eclf1.score(X\_test\_lda,y\_test))

#Validation method for random forest

y\_train\_pred=best\_grid\_rf.predict(X\_train\_lda) #Predicted Training Score

y\_test\_pred=best\_grid\_rf.predict(X\_test\_lda) #Predicted Testing Score

y\_score\_train = best\_grid\_rf.predict\_proba(X\_train\_lda) #Predicted Training Probability Score (both for 0 and 1)

y\_score\_test = best\_grid\_rf.predict\_proba(X\_test\_lda) #Predicted Testing Probability Score (both for 0 and 1)

fpr\_train, tpr\_train, thresholds\_train =roc\_curve(y\_train, y\_score\_train[:,1]) #y\_score\_train[:,1] means prediction for class 1

#y\_score\_train[:,0] means prediction for class 0 (redundant here)

fpr\_test, tpr\_test, thresholds\_test =roc\_curve(y\_test, y\_score\_test[:,1])#false positive test, true positive test and threshold

#Confusion Matrix for Training and Testing

print("Confusion Matrix of the training data:\n",confusion\_matrix(y\_train\_pred,y\_train))

print("Confusion Matrix of the testing data:\n",confusion\_matrix(y\_test\_pred,y\_test))

#Overall Accuracy for training and testing

print("The overall accuracy score of training data=",np.round(best\_grid\_rf.score(X\_train\_lda,y\_train),2))

print("The overall accuracy score of testing data=",np.round(best\_grid\_rf.score(X\_test\_lda,y\_test),2))

#Average AUC score for training and testing

print("Average AUC score for training data:",metrics.auc(fpr\_train, tpr\_train)) #Average AUC score for training

print("Average AUC score for testing data:",metrics.auc(fpr\_test, tpr\_test)) #Average AUC score for testing

#Average Precision score for training and testing

print("Average precision score for training data:",average\_precision\_score(y\_train, y\_score\_train[:,1]))

# Average precision score for training

print("Average precision\_score for testing data:",average\_precision\_score(y\_test, y\_score\_test[:,1]))

# Average precision score for testing

BA\_train=(average\_precision\_score(y\_train, y\_score\_train[:,1])+balanced\_accuracy\_score(y\_train,y\_train\_pred))/2;

BA\_test=(average\_precision\_score(y\_test, y\_score\_test[:,1])+balanced\_accuracy\_score(y\_test,y\_test\_pred))/2;

print("Balanced accuracy of the training data:\n",BA\_train)

# Average balanced accuracy score for training

print("Balanced accuracy of the testing data:\n",BA\_test)

#Average balanced accuracy score for testing

#tpr\_train[np.where(np.round(thresholds\_train,2)==0.51)],fpr\_train[np.where(np.round(thresholds\_train,2)==0.51)]

#tpr\_test[np.where(np.round(thresholds\_test,2)==0.51)],fpr\_test[np.where(np.round(thresholds\_test,2)==0.51)]

#precision\_train[np.where(np.round(thresholds\_train,2)==0.51)],recall\_train[np.where(np.round(thresholds\_train,2)==0.51)]

#precision\_test[np.where(np.round(thresholds\_test,2)==0.51)],recall\_test[np.where(np.round(thresholds\_test,2)==0.51)]

#Graph

#ROC curve for training and testing

plt.figure(1,figsize=(6,4),dpi=300)

plt.plot(fpr\_train,tpr\_train,'--',label='train')

plt.plot(fpr\_test,tpr\_test,label='test')

plt.legend(loc='upper left')

plt.title('ROC Curve')

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.savefig('ROC.jpg',dpi=300)

#Precision curve for training and testing

plt.figure(2,figsize=(6,4),dpi=300)

precision\_train, recall\_train, thresholds\_train =precision\_recall\_curve(y\_train, y\_score\_train[:,1])

precision\_test, recall\_test, thresholds\_test =precision\_recall\_curve(y\_test, y\_score\_test[:,1])

plt.plot(recall\_train,precision\_train,'--',label='train')

plt.plot(recall\_test,precision\_test,label='test')

plt.legend(loc='upper left')

plt.title('Precision Recall Curve')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.savefig('PreVsRecall.jpg',dpi=300)