

Fruit Feature Classification

1st John Iacoucci

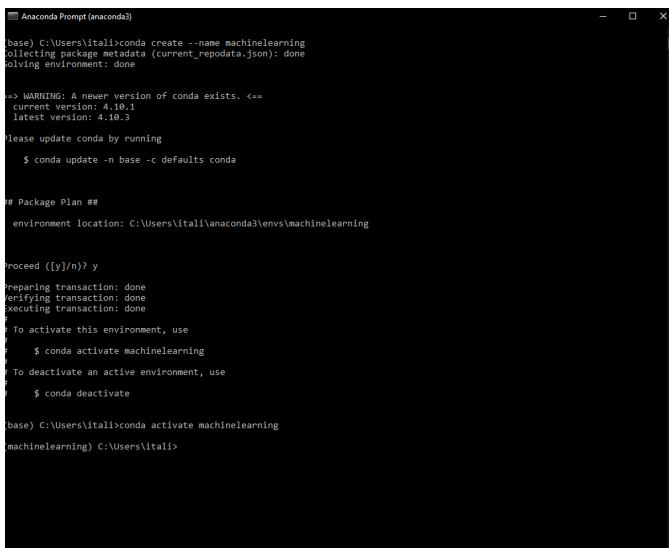
*CSC410 Big Data and Machine Learning
University of North Carolina at Greensboro*

Abstract—This paper will explore the process of machine learning and big data through the eyes of a current undergraduate student. Using modern technologies and libraries to extract, process, and analyze data obtained through feature extraction of fruit images.

I. ENVIRONMENT SETUP

The environment used was Anaconda with Python as the language, this environment is good for data science applications since it is versatile allowing the ease of creating new environment and installing the necessary packages. I used the following commands to create my environment along with the installation of libraries:

Environment creation: `conda create --name machinelearning`
Environment activation: `conda activate machinelearning`
OpenCV install: `conda install -c conda-forge opencv`
Jupyter Lab install: `conda install -c conda-forge jupyterlab`
Matplotlib install: `conda install -c conda-forge matplotlib`
Pandas install: `conda install -c anaconda pandas`
Numpy install: `conda install -c anaconda numpy`



```
base) C:\Users\itali>conda create --name machinelearning
Collecting package metadata (current_repodata.json): done
Solving environment: done

--> WARNING: A newer version of conda exists. <-->
  current version: 4.10.1
  latest version: 4.10.3
Please update conda by running
  $ conda update -n base -c defaults conda

# Package Plan ##
  environment location: C:\Users\itali\anaconda3\envs\machinelearning

proceed ([y]/n)? y
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

To activate this environment, use
  $ conda activate machinelearning
To deactivate an active environment, use
  $ conda deactivate

base) C:\Users\itali>conda activate machinelearning
(machinelearning) C:\Users\itali>
```

Fig. 1. Setting up machine learning environment

These libraries are used all throughout the project as we will see later. The projects code was written within Jupyter Lab to keep all the code split into cell and have everything nice and organized.

II. DATASET SELECTION

The dataset used is the FIDS30 fruit images, I choose to select Peaches, Plums, and Mangos. These fruits have distinct physical characteristics such as color and shape that I felt would give better classification in the long run. I choose image 0 from peaches, 7 from plums, and 2 from mangos. The file directory can be seen in Figure 1.

Name	Date modified	Type
acerolas	9/17/2021 12:42 AM	File folder
apples	9/17/2021 12:42 AM	File folder
apricots	9/17/2021 12:42 AM	File folder
avocados	9/17/2021 12:42 AM	File folder
bananas	9/17/2021 12:42 AM	File folder
blackberries	9/17/2021 12:42 AM	File folder
blueberries	9/17/2021 12:42 AM	File folder
cantaloupes	9/17/2021 12:42 AM	File folder
cherries	9/17/2021 12:42 AM	File folder
coconuts	9/17/2021 12:42 AM	File folder
figs	9/17/2021 12:42 AM	File folder
grapefruits	9/17/2021 12:42 AM	File folder
grapes	9/17/2021 12:42 AM	File folder
guava	9/17/2021 12:42 AM	File folder
kiwifruit	9/17/2021 12:42 AM	File folder
lemons	9/17/2021 12:42 AM	File folder
limes	9/17/2021 12:42 AM	File folder
mangos	9/17/2021 12:43 AM	File folder
olives	9/17/2021 12:43 AM	File folder
oranges	9/17/2021 12:43 AM	File folder
passionfruit	9/17/2021 12:43 AM	File folder
peaches	9/17/2021 12:43 AM	File folder
pears	9/17/2021 12:43 AM	File folder
pineapples	9/17/2021 12:43 AM	File folder
plums	9/17/2021 12:43 AM	File folder
pomegranates	9/17/2021 12:43 AM	File folder
raspberries	9/17/2021 12:43 AM	File folder
strawberries	9/17/2021 12:43 AM	File folder
tomatoes	9/17/2021 12:43 AM	File folder
watermelons	9/17/2021 12:43 AM	File folder

Fig. 2. Image of directory containing the fruit images

III. IMAGE IMPORTATION AND CONVERSION

This is where the images that are to be processes are imported using open cv along with their conversion to gray scale.

A. Image Importation Code

In this code (Fig.2) the images are imported in one of two ways. They are imported by a direct file path to the image or the user gives a file path with a folder of images. The images are set to a variable that is to be used later within the code.

```
# Read Images (.jpg, .png, and .tif)
peach_color = cv2.imread("C:/Users/itali/Assignment1_410/FIDS30/peaches/0.jpg")
plum_color = cv2.imread("C:/Users/itali/Assignment1_410/FIDS30/plums/7.jpg")
mango_color = cv2.imread("C:/Users/itali/Assignment1_410/FIDS30/mangos/2.jpg")

# Goes to directory and extracts the files out
path = 'C:/Users/itali/Assignment1_410/FIDS30/peaches'
files = [f for f in listdir(path) if isfile(join(path, f))]
# creates a image array based on the number of images in the directory
images_color = np.empty(len(files), dtype=object)
# adds images to the array
for n in range(0, len(files)):
    images_color[n] = cv2.imread(join(path, files[n]) |
```

Fig. 3. Importation of images code

B. Gray Scale Conversion

This code takes the image that was previously obtained and converts it to gray scale using the opencv library. It then prints it shape for the user to see.

```
# Converts images to gray scale
images_gray = np.empty(len(images_color), dtype=object)
height = np.empty(len(images_color), dtype=object)
width = np.empty(len(images_color), dtype=object)
for n in range(0, len(images_color)):
    images_gray[n] = cv2.cvtColor(images_color[n], cv2.COLOR_BGR2GRAY)
    height[n], width[n] = images_gray[n].shape
    print(images_gray[n].shape)

# Convert images to grayscale.
peachG = cv2.cvtColor(peach_color, cv2.COLOR_BGR2GRAY)
heightPeG, widthPeG = peachG.shape
print(peachG.shape)

plumG = cv2.cvtColor(plum_color, cv2.COLOR_BGR2GRAY)
heightPlG, widthPlG = plumG.shape
print(plumG.shape)

mangoG = cv2.cvtColor(mango_color, cv2.COLOR_BGR2GRAY)
heightmG, widthmG = mangoG.shape
print(mangoG.shape)
```

Fig. 4. Image Gray Scale Conversion

IV. RESIZING IMAGES

This section takes the grayscale images and reduces their size such that they are still divisible by 8. This is so that when we extract the features later we can use our 8x8 filters.

The code under the comment "Resize calculation" in Fig.4 take the images from a array that were obtained from the folder given such as peaches, it then cycles through and obtains the image size. It calculates the aspect ratio, then the new width of the image based on a 256 height. Now it converts the images to their new size and saves them within a array. The bottom portion of the code is similar but only does one image at a time which is the original images selected.

A. Saving the New Images

The new images are then saved into a new folder shown in Fig.5 and Fig.6

```
# Resize calculation by leveraging aspect ratio
images = np.empty(len(images_color), dtype=object)
for n in range(0, len(images_color)):
    tmp_height = height[n]
    tmp_width = width[n]
    aspect_ratio = tmp_width / tmp_height
    new_width = int(256 * aspect_ratio)
    while((new_width % 8) != 0):
        new_width = new_width + 1
    images[n] = cv2.resize(images_gray[n], dsize=(new_width, 256), interpolation=cv2.INTER_CUBIC)
    height[n], width[n] = images[n].shape
    print(images[n].shape)

# Raw Data Resizing
peach = cv2.resize(peachG, dsize=(320, 256), interpolation=cv2.INTER_CUBIC)
plum = cv2.resize(plumG, dsize=(256, 256), interpolation=cv2.INTER_CUBIC)
mango = cv2.resize(mangoG, dsize=(344, 256), interpolation=cv2.INTER_CUBIC)

peach = cv2.normalize(peach.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)*255
plum = cv2.normalize(plum.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)*255
mango = cv2.normalize(mango.astype('float'), None, 0.0, 1.0, cv2.NORM_MINMAX)*255

heightPe, widthPe = peach.shape
heightPl, widthPl = plum.shape
heightM, widthM = mango.shape

print(peach.shape)
print(plum.shape)
print(mango.shape)
```

Fig. 5. Image Resizing

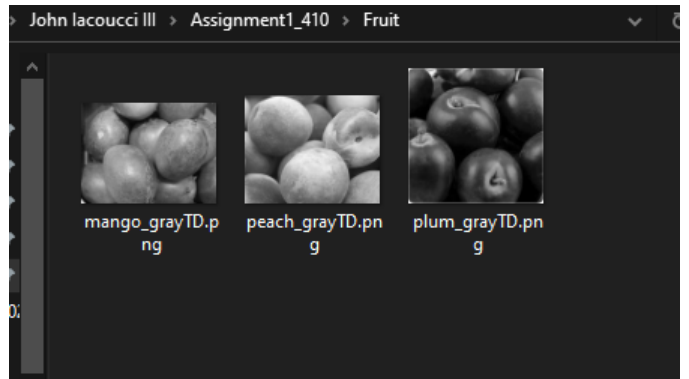


Fig. 6. Resized Gray Scale Images

V. BLOCK FEATURE VECTOR

At this point we will now take the newly created images and split them into 8x8 blocks which will extract the features from the images based on their values. They are then exported into a csv after creating a data frame using pandas. We first binarize the images as shown in the code in Fig.8

This image shows both the 3 original image selections and the folder of images being binarized. We then can take our 8x8 block out of the code which is shown in Fig.9.

This code take the images and cycles through the values of the image to extract out the features which is then put into a data frame and exported to a csv.

Fig.10 displays the mangos image data along with its classifier in column 64 which is 2.

VI. OVERLAPPING 8X8 BLOCK FEATURE VECTORS

This section is very similar to the previous section as it uses the majority of the same concept and code with a few changes.

In Fig.11 the code shows that the image is being looked over by a 8x8 overlapping block so that it is only moving

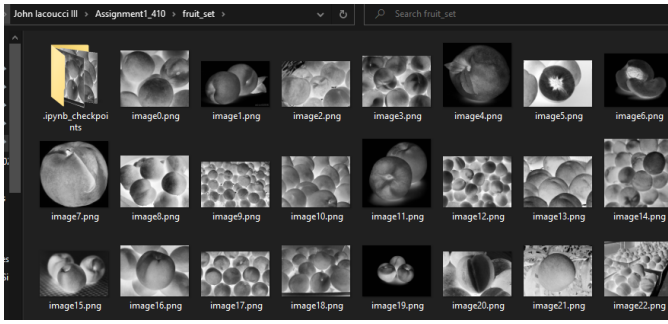


Fig. 7. Resized from user given a folder

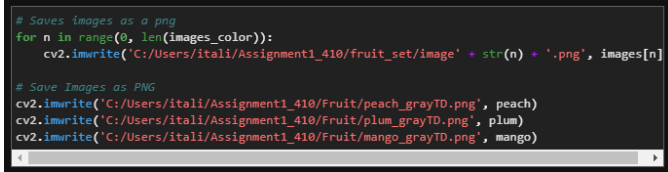


Fig. 8. Saving resized image

over one by one across the image. This allows more data to be captured that can be used in better classification later on.

VII. STATISTICAL DATA OF THE IMAGES

This data can be looked at multiple ways in which we can derive different meaning. The main things we are going to look at is if the data is balanced, accurate, and complete. The data seems to be mostly balance with the lowest observations in Fig.14 being the Plum, but it is only around 400 observations lower which should have little effect. The data is accurate since all the labels on the data is correct as we ourselves have manually assigned the labels. The data also seems complete as we can see in Fig.15. The data does not have a scalability problem as the data only has 64 features per vector which can be scaled if needed. The data is trivial as we can easily process this data with the current technology we are using now in this project. The data is also not highly dimensional since the features is not greater than the observations we have.

VIII. CONSTRUCT A FEATURE SPACE

This is where everything all comes together. In the code in Fig.16 it shows the feature spaces being merged into one feature space, one is from the peaches and plums and the other is Peaches, Plums, and Mangos.

We then also randomize the feature spaces so that when we train our model is has a better learning experience.

IX. SUBSPACE DISPLAY

We can see the first subspace in Fig.15 which we have features 1 and 54 from the peach and plum images. The data is balanced and accurate as the data is spread out. This is taken from the data frame of the non overlapping 8x8 blocks.

This is the 3d subspace where we get a better idea of the positioning of the data and how it orients itself. We can change the features around such that we could get different angles



Fig. 9. Binarizing image code

to view the data to see how the data could be classified. Compared to Fig.15 we get a slightly better idea of how to classify the data but the data is still very much intertwined which could make it hard to classify for a model. We would need to apply some changes to the data sets to get better result before being able to give this to a model. This is the code used to create said graph.

X. ADDITIONAL CODE

The additional code which makes the program more modular is given throughout the figures previously discussed. This code allows a folder to be given and turned into a feature space within a csv file. This code is using the peaches folder so all images are similar and the classifier will be the same this can be helpful when large amounts of similar data needs to be obtained.

XI. EFFECTS OF BLOCK SIZE

The block size chosen is 8x8 which would give is a dimensionality of 64 and or 64 features. This is a relatively low number of features for data that is going to be used to train models. Also depending on if the data is overlapping or not the blocks limit us to a certain number of vectors. If we were to reduce the filter size it would allow us to capture more local information that would allow us to better classify the images.

```

# Non-Overlapping 8x8
# Peach
pe = round(((heightPe)*(widthPe)/64)
flat_pe = np.zeros((pe, 65), np.uint8)
k = 0
for i in range(0,heightPe,8):
    for j in range(0,widthPe,8):
        crop_tmp3 = peach[i:i+8, j:j+8]
        flat_pe[k,0:64] = crop_tmp3.flatten()
        k = k + 1

# Creates Panda obj
fespace_pe = pd.DataFrame(flat_pe)
fespace_pe.to_csv('C:/Users/itali/Assignment1_410/DataFrames/fespace_Pe.csv', index=False)

# Plum
pl = round(((heightPl)*(widthPl)/64)
flat_pl = np.ones((pl, 65), np.uint8)
k = 0
for i in range(0,heightPl,8):
    for j in range(0,widthPl,8):
        crop_tmp4 = plum[i:i+8, j:j+8]
        flat_pl[k,0:64] = crop_tmp4.flatten()
        k = k + 1

# Creates Panda obj
fespace_pl = pd.DataFrame(flat_pl)
fespace_pl.to_csv('C:/Users/itali/Assignment1_410/DataFrames/fespace_Pl.csv', index=False)

# Mango
m = round(((heightM)*(widthM)/64)
flat_m = (np.ones((m, 65), np.uint8)+1)
k = 0
for i in range(0,heightM,8):
    for j in range(0,widthM,8):
        crop_tmp5 = mango[i:i+8, j:j+8]
        flat_m[k,0:64] = crop_tmp5.flatten()
        k = k + 1

# Creates Panda obj
fespace_m = pd.DataFrame(flat_m)
fespace_m.to_csv('C:/Users/itali/Assignment1_410/DataFrames/fespace_m.csv', index=False)

```

Fig. 10. Non-Overlapping 8x8 Blocks

60	61	62	63	64
31	31	29	25	2
31	30	32	31	2
32	33	33	34	2
38	44	49	51	2
55	56	57	59	2
54	54	54	53	2
75	78	77	80	2
69	67	65	63	2
75	76	77	77	2
76	66	68	88	2
86	90	81	90	2
84	88	95	103	2
120	118	120	121	2
135	141	146	150	2
161	160	157	161	2
173	179	177	178	2
185	180	179	178	2

Fig. 11. CSV of the Mango Data Frame

With the current data from the images using this 8x8 block it is going to be difficult to classify the images based on the domain of data.

XII. DIVIDING THE DOMAIN

To start the data collected from the previous assignment was used to train the models that will be discussed later-on. In order to get the data sets to train and test these models I wrote a function that takes a file name specifically a csv and splits the data into a 80:20 split 80 percent will go to training and the other 20 percent will go to testing. Furthermore, I separated the training and testing into the x and y portions of each data set. Then taking each x and y for testing and

```

# Overlapping 8x8 for all images within the given folder.
im = np.zeros(len(images_color), dtype=object)
flat_im = np.zeros(len(images_color), dtype=object)
fespace = np.zeros(len(images_color), dtype=object)

for n in range(len(images_color)):
    im[n] = round(((height[n]-7)*(width[n]-7)))
    flat_im[n] = np.zeros((im[n], 65), np.uint8)
    k = 0
    for i in range(height[n]-7):
        for j in range(width[n]-7):
            crop = images[n][i:i+8, j:j+8]
            flat_im[n][k,0:64] = crop.flatten()
            k = k + 1

    fespace[n] = pd.DataFrame(flat_im[n])
    #fespace[n].to_csv('C:/Users/itali/Assignment1_410/DataFrames/fespace' + str(n) + '.csv', index=False)

# Overlapping 8x8
# Peach
pe = round(((heightPe-7)*(widthPe-7)))
flatPe = np.zeros((pe, 65), np.uint8)
k = 0
for i in range(heightPe-7):
    for j in range(widthPe-7):
        crop_tmp1 = peach[i:i+8, j:j+8]
        flatPe[k,0:64] = crop_tmp1.flatten()
        k = k + 1

# Creates Panda object
fespacePe = pd.DataFrame(flatPe)
fespacePe.to_csv('C:/Users/itali/Assignment1_410/DataFrames/fespacePe.csv', index=False)

# Plum
pl = round(((heightPl-7)*(widthPl-7)))
flatPl = np.ones((pl, 65), np.uint8)
k = 0
for i in range(heightPl-7):
    for j in range(widthPl-7):
        crop_tmp = plum[i:i+8, j:j+8]
        flatPl[k,0:64] = crop_tmp.flatten()
        k = k + 1

# Creates Panda object
fespacePl = pd.DataFrame(flatPl)
fespacePl.to_csv('C:/Users/itali/Assignment1_410/DataFrames/fespacePl.csv', index=False)

# Mango
m = round(((heightM-7)*(widthM-7)))
flatM = (np.ones((m, 65), np.uint8)+1)
k = 0
for i in range(heightM-7):
    for j in range(widthM-7):
        crop_tmp2 = mango[i:i+8, j:j+8]
        flatM[k,0:64] = crop_tmp2.flatten()
        k = k + 1

# Creates Panda object
fespaceM = pd.DataFrame(flatM)
fespaceM.to_csv('C:/Users/itali/Assignment1_410/DataFrames/fespaceM.csv', index=False)

```

Fig. 12. 8x8 Overlapping Block Code

training and exporting them as their own csv files to be used for training and testing later.

The splitting was done using the sklearn library using the split function which takes the output variables as arguments along with the data set to be split, the function also can be customized and be told what percent of data you want to use for testing. In this case the parameter was set to 0.2 for 20 percent testing data.

XIII. RELATION BETWEEN TRAINING AND TESTING DATA SETS

We can look at Fig.20 and 22 to see that both follow the same distribution but with the training data being much more dense. In Fig.21 and 23 we can also see that they follow the same distributions as they are both skewed to the right.

XIV. ELASTIC-NET REGRESSION MODEL

Now we move onto developing the model, in this case we are using a regression model which is the elastic-net model. In

```

# Statistical Info and Graphs
# Mean of the features
mango_mean = fspace_m.iloc[:,0:63].mean()
peach_mean = fspace_pe.iloc[:,0:63].mean()
plum_mean = fspace_pl.iloc[:,0:63].mean()

fig = plt.figure(figsize = (10, 5))
plt.plot(mango_mean, label = 'Mango', color = 'red')
plt.plot(plum_mean, label = 'Plum', color = 'green')
plt.plot(peach_mean, label = 'Peach')
plt.xlabel('Features')
plt.ylabel('Mean')
plt.title('Means of Mangos, Peaches, and Plums')
plt.grid(True)
plt.legend()
plt.show()

# Number of observations
data = {'Mango':len(fspace_m), 'Plum':len(fspace_pl), 'Peach':len(fspace_pe)}
fruit = list(data.keys())
obs = list(data.values())

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(fruit, obs, color = 'b',
        width = 0.4)

plt.xlabel("Fruits")
plt.ylabel("Number of Observations")
plt.title("Observations per fruit")
plt.show()

```

Fig. 13. Statistical Graph Code

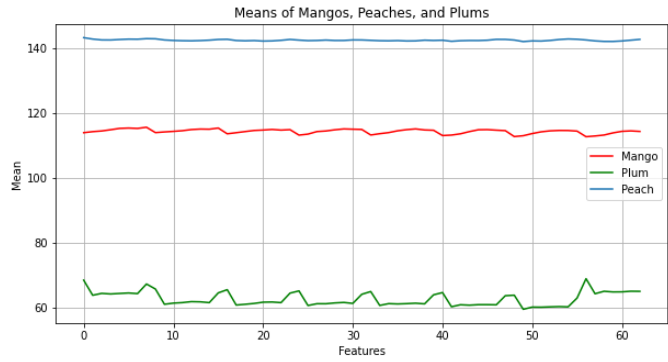


Fig. 14. Means of the Features per Fruit

this model we are taking the training and testing data sets we created earlier and importing them in. We assign the training values x and y to their respective variables and the testing x and y as well. The regression model is done using the sklearn library and takes the training values as parameters. We then create a new variable to hold the predictions and assign the x testing values to to make the predictions. This is then entered into a confusion matrix which takes the y testing values and the predicted values to determine which data points were classified correctly. In the confusion matrix found below you can see that we had 12651 classified as label 0 correctly and 9457 classified as label 1 correctly. This gave an accuracy of 0.7899 and a precision of 0.7643. These numbers are ok for a regression model and could be possibly improved with optimizing our parameters.

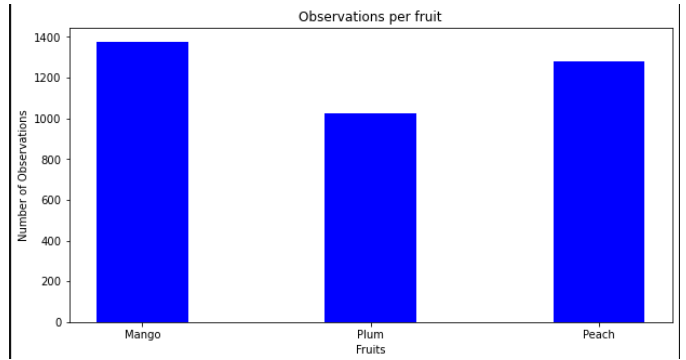


Fig. 15. Bar Graph of Observation per Fruit

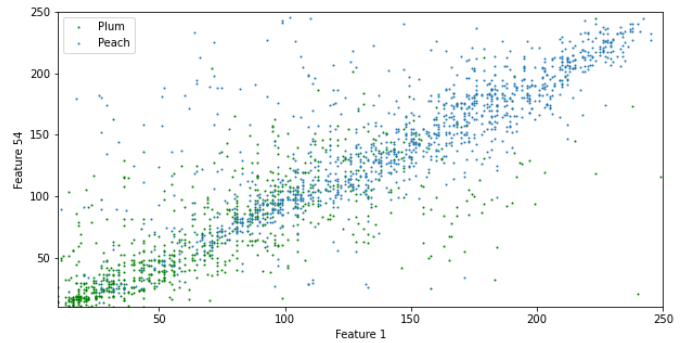


Fig. 16. Feature 1 and 54 from Peach and Plum Image

XV. RANDOM FOREST

The random forest model is a bit more complex by using multiple decision trees to reach a decision or result. It is more accurate as we will see from the results obtained. The data is once again imported in using the pandas library into a data frame and split between the x and y training and testing variables. The training variables are then put into the model to be trained. This training is slightly different because it uses ensembles as shown in the figures below to take the most popular result which uses bootstrap aggregation. It then takes the x testing values and tests the model. Again using a confusion matrix it gives 15236 points classified as label 0 correctly and 11761 points classified as label 1 correctly as seen in the figure below. This gives an accuracy of 0.9646 and a precision of 0.9726 both are much higher than the previous model.

XVI. COMPARING THE MODELS

The models can be compared using a function of the sklearn library which is the metrics accuracy and sensitivity functions. The builtin function measured the elastic-nets accuracy and sensitivity at 0.7898 and 0.7898. The random forest builtin accuracy and sensitivity measured at 0.9646 and 0.9646. This shows that the random forest model is much more accurate at predicting the class and with a higher sensitivity it also has a greater chance of identifying the correct labels.

The differences with the confusion matrix compared to the regular quantitative measure is largely in part from its ability


```
# Merges all data frames and converts to a csv
merged_set = pd.concat(fspace)
merged_set.to_csv('C:/Users/itali/Assignment1_410/DataFrames/merged_set.csv')

# Merge features from classes
frames01 = [fspacePe, fspacePl]
merged01 = pd.concat(frames01)

indx01 = np.arange(len(merged01))
rndmerged01 = np.random.permutation(indx01)

rndmerged01 = merged01.sample(frac=1).reset_index(drop=True)

rndmerged01.to_csv('C:/Users/itali/Assignment1_410/DataFrames/merged01.csv')

# Creates a csv from the 3 dataframes.
frames012 = [fspacePe, fspacePl, fspaceM]
merged012 = pd.concat(frames012)
merged012.to_csv('C:/Users/itali/Assignment1_410/DataFrames/merged012.csv')
```

Fig. 17. Feature Spaces Merged

```
#2D plot of features 1 and 54
fig = plt.figure(figsize = (10, 5))
ax = fig.add_subplot(111)
ax.scatter(fspace_pl.loc[:,1], fspace_pl.loc[:,54], color = 'green', s = 1, label= 'Plum')
ax.scatter(fspace_pe.loc[:,1], fspace_pe.loc[:,54], s = 1, label= 'Peach')
ax.scatter(fspace_m.loc[:,1], fspace_m.loc[:,54], color = 'red', s=1)
plt.xlim(10, 250)
plt.ylim(10, 250)
ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 54')
ax.legend()
plt.show()

# 3D plot of features 1, 22, and 54
fig = plt.figure(figsize = (10, 5))
ax = fig.add_subplot(1,1,1, projection= '3d')
ax.scatter(fspace_pl.loc[:,1], fspace_pl.loc[:,22], fspace_pl.loc[:,54], color = 'green', s=1, label = 'Plum')
ax.scatter(fspace_pe.loc[:,1], fspace_pe.loc[:,22], fspace_pe.loc[:,54], s=1, label = 'Peach')
ax.scatter(fspace_m.loc[:,1], fspace_m.loc[:,22], fspace_m.loc[:,54], color = 'red', s=1, label = 'Mango')
ax.set_xlabel('Feature 22')
ax.set_ylabel('Feature 1')
ax.set_zlabel('Feature 54')
ax.legend()
plt.show()
```

Fig. 19. 2D, 3D Graph Code

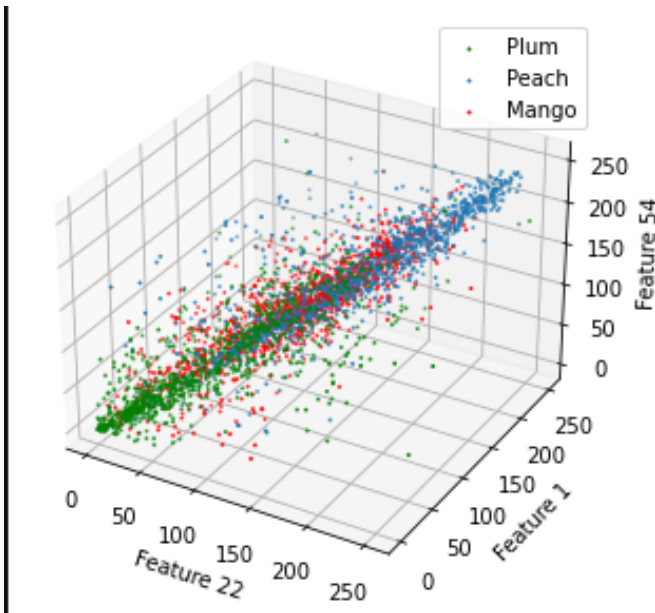


Fig. 18. 3d Subspace showing features 1,22,54 from the 3 fruit images

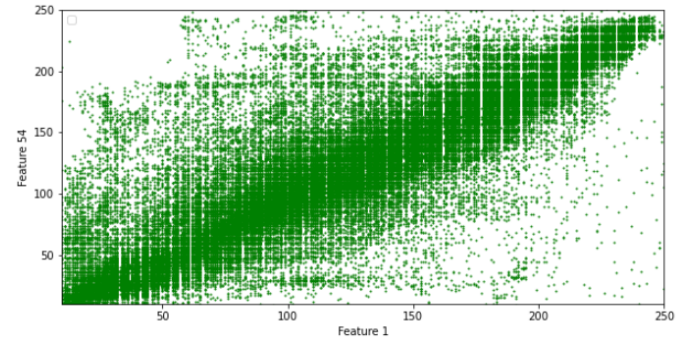


Fig. 20. Scatter Plot of Training Data

XVII. BIG DATA SYSTEM

The big data system is used to process the large amounts of data from the fruit images and more efficiently train machine learning models like we will see later on.

A. Getting Familiar with Databricks

The big data system I used was Databricks, to start I signed up for a community edition account and started the quick-start tutorial along with watching Dr.Suthaharan video. The quick start tutorial showed how to create a new cluster which was done by going to the compute tab and clicking the create a cluster button. I then selected the type of environment to run the notebook in and created the cluster which took roughly 10 minutes to create. The tutorial then showed me how to detach and attach the notebook to the newly created cluster and run everything. A important part of the tutorial is in the code provided within the notebook, the query to the data table in order to convert the table into a data frame using spark is a important step as we will see later on. The query is identical to a SQL query so even though I just queried the entire table I can see how being able to query the data in anyway can be extremely useful.

B. Setting up the Environment

The environment setup was exactly the same as in the tutorial, I created a new cluster called FruitCluster with the 7.3 LTS ML version that runs Spark 3.0.1 and Scala 2.12. Then I began to import the data sets into databricks, to do

to understand the data better while giving feed back to the engineer. The confusion matrix has the ability to determine the positives and false positives of the models predictions. This will allow the engineer to have insight to what the model is predicting and be able to better fix the issues with the model. Compared to the regular measures that will only provide feedback on as specific thing like accuracy. Knowing whether the accuracy is high or low is useful but being able to tell that label 1 has much higher false positives will give the extra advantage in fixing or optimizing the model.

The random forest model is by far the better model than the elastic-net. The model not only had higher accuracy in both the builtin and confusion matrix but also with the precision of the model compared 0.97 to 0.76. This model also has less false positives compared to the elastic-net with a total of 991 to 5880, this model can also be used for multiple classification situations compared to the regression only being optimal for two class scenarios.

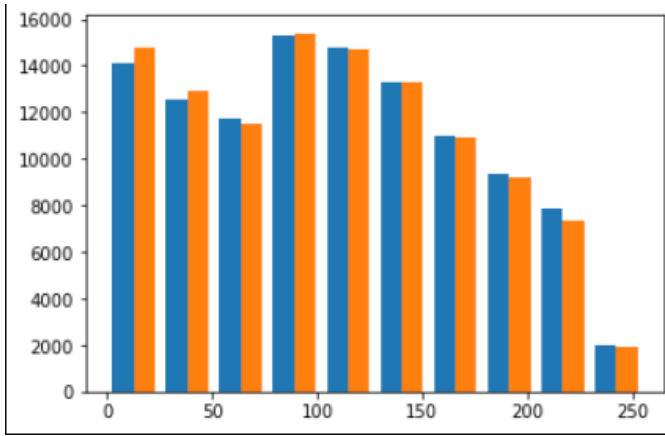


Fig. 21. Histogram of Training Data

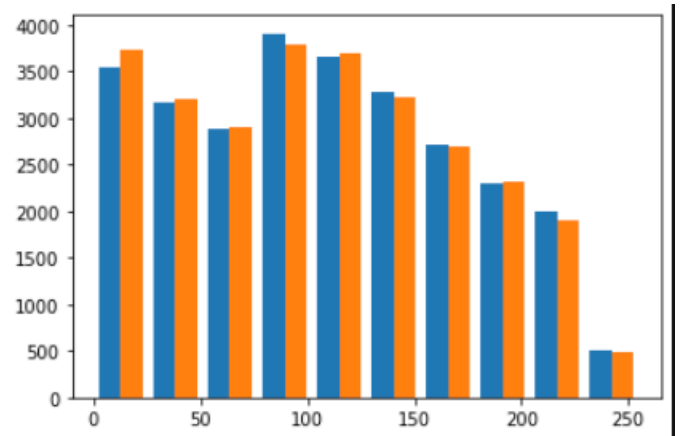


Fig. 23. Histogram of Testing Data

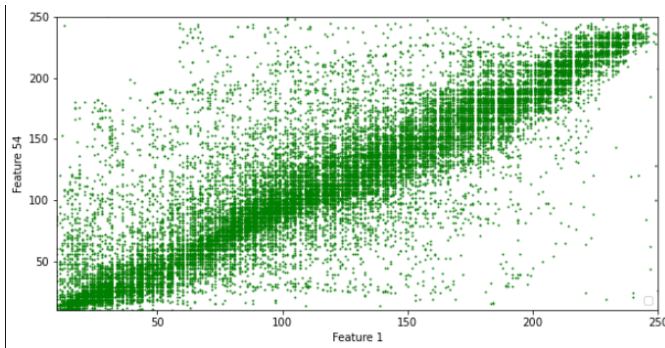


Fig. 22. Scatter Plot of Testing Data

	0	1
0	12651	2916
1	2964	9457

Fig. 24. Confusion Matrix for Elastic-Net Regression Model

so I went to the data tab on the left side and clicked create table. From there I loaded the csv file from the data frame into databricks and clicked create table with UI with the cluster I created before. Then after changing the settings to first row is header, infer schema to on I created the table. This turned the data into a database table which can now be used to query.

XVIII. TRAINING RANDOM FOREST WITH DATABRICKS

The training of the random forest model was fairly straight forward as I followed the tutorial given within the assignment. I started with importing the previously created database and converted it to a dataframe using pandas. Next I checked for any NaN values and split the dataframe 80:20 for the training and testing using the sklearn train test split function. Then I created the random forest model by using mlflow which will allow for each iteration of the model to become an experiment that can be analyzed and used to save the model for later use. This is also where the auc score is calculated which is also logged by mlflow. After the model was trained and tested within the last function I calculated displayed the important features then displayed the auc score for that experiment.

XIX. RESULTS

In this section I will talk about the results obtained from the new databricks model. Also I will compare the databricks model to the previous model computed on the local machine.

A. Results from Databricks

The Databricks experiment showed the following, the auc was 0.987 which means that the model predicted the data accurately and with a time of only 9 seconds. This is lighting fast for a random forest algorithm as we will see later. I also saw from computing the important features that feature 49 had the most impact among the other features.

B. Comparing Databricks to Local Machine

The accuracy of the models were fairly close with the databricks model being slightly more accurate with .98 compared to the local machines .96. The thing that shows how effective the big data system was though was in the computational time the local machine took 2 minutes to complete the training of the random forest model compared to the big data systems 9 seconds. This difference is huge although 2 minutes isn't a lot of time, if the data set was even larger than this one then the local machine would not even be able to process it while the big data system could still process and train the model within a reasonable time frame.

REFERENCES

- [1] Automatic fruit recognition using computer vision, Škrjanec Marko, Bsc Thesis, (Mentor: Matej Kristan), Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2013

```

# Training with 80% data
#X1_train = X1[0:TR-1,:]
#Y1_train = Y1[0:TR-1]

# Import Data
xtr = pd.read_csv('C:/Users/itali/Assignment1_410/Training/xTraining01.csv', header=
ytr = pd.read_csv('C:/Users/itali/Assignment1_410/Training/yTraining01.csv', header=
xtt = pd.read_csv('C:/Users/itali/Assignment1_410/Testing/xTesting01.csv', header=
ytt = pd.read_csv('C:/Users/itali/Assignment1_410/Testing/yTesting01.csv', header=

# Replace NaN values with the mean
xtr.fillna((xtr.mean()),inplace=True)
ytr.fillna((ytr.mean()),inplace=True)
xtt.fillna((xtt.mean()),inplace=True)
ytt.fillna((ytt.mean()),inplace=True)

```

Fig. 25. Imports for Elastic-Net Model

```

# Train model
en = ElasticNet(random_state=0)
en.fit(xtr,ytr)

ElasticNet(random_state=0)

# Testing with 20% data
yhat_test = en.predict(xtt)
#print(ytt.shape)

# Create confusion matrix and save to a csv
CC_test = confusion_matrix(ytt, yhat_test.round())
pd.DataFrame(CC_test).to_csv('C:/Users/itali/Assignment1_410/ConfusionMatrix/Confu
#print(CC_test)

TN = CC_test[0,0]
FP = CC_test[0,1]
FN = CC_test[1,0]
TP = CC_test[1,1]

FPFN = FP+FN
TPTN = TP+TN

# Confusion Analytics
Accuracy = 1/(1+(FPFN/TPTN))
print("Our_Accuracy_Score:",Accuracy)
Precision = 1/(1+(FP/TP))
print("Our_Precision_Score:",Precision]

Our_Accuracy_Score: 0.7899099614120337
Our_Precision_Score: 0.7643255475632426

```

Fig. 26. Elastic Net Training

```

#Training
rF = RandomForestClassifier(random_state=0, n_estimators=1000, oob_score=True, n_jobs=-1)
model = rF.fit(xtr, ytr)

```

Fig. 27. Random Forest Training

```

# Out of bag error calculation
oob_error = 1 - rF.oob_score_
oob_error

0.03438080605973992

# Testing
yhat_test = rF.predict(xtt)
#print(y_test.shape)

# Confusion Matrix
CC_test = confusion_matrix(ytt, yhat_test.round())
pd.DataFrame(CC_test).to_csv('C:/Users/itali/Assignment1_410/ConfusionMatrix/Confusion02.csv')

TN = CC_test[0,0]
FP = CC_test[0,1]
FN = CC_test[1,0]
TP = CC_test[1,1]

FPFN = FP+FN
TPTN = TP+TN

# Calculated Measures
Accuracy = 1/(1+(FPFN/TPTN))
print("Calculated Accuracy:", Accuracy)
Precision = 1/(1+(FP/TP))
print("Calculated Precision:", Precision)

from sklearn import metrics
print("BuiltIn_Accuracy:",metrics.accuracy_score(ytt, yhat_test.round()))
print("BuiltIn_Sensitivity (recall):",metrics.recall_score(ytt,
yhat_test.round(), average='weighted'))

Calculated Accuracy: 0.9645919679862799
Calculated Precision: 0.9726265299371486
BuiltIn_Accuracy: 0.9645932330558433
BuiltIn_Sensitivity (recall): 0.9645932330558433

```

Fig. 28. Random Forest Code

Cluster name

FruitCluster

Databricks runtime version ?

Runtime: 7.3 LTS ML (Scala 2.12, Spark 3.0.1) | v

Instance

Free 15 GB Memory: As a Community Edition user, your cluster will automatically terminate. For more configuration options, please upgrade your Databricks subscription.

Instances Spark

Availability zone ?

auto | v

Fig. 29. Cluster Creation

Create New Table

Data source

Upload File S3 DBFS Other Data Sources Partner Integrations

DBFS Target Directory

/FileStore/tables/ (optional)

Files uploaded to DBFS are accessible by everyone who has access to this workspace. [Learn more](#)

Files

merged01.csv

32.4 MB

Remove file

File uploaded to /FileStore/tables/merged01-4.csv

Select a Cluster to Preview the Table

Choose a cluster with which you will read and preview the data.

Cluster

FruitCluster

Specify Table Attributes

Specify the Table Name, Database and Schema to add this to the data UI for other users to access

Table Name

merged01_4.csv

Create in Database

default

File Type

CSV

Column Delimiter

☐ First row is header

☒ Infer schema

☐ Multi-line

Table Preview

Fig. 30. Database Creation

```

1 logged_model = 'runs:/38b53809e04c4bc8b5a268424a2c1cf5/random_forest_model'
2
3 # Load model as a PyFuncModel.
4 loaded_model = mlflow.pyfunc.load_model(logged_model)
5
6 print(f'AUC: {roc_auc_score(y_test, loaded_model.predict(X_test))}')

```

AUC: 0.9873639894761497

Command took 1.89 seconds -- by jaiacouc@uncg.edu at 11/26/2021, 10:05:30 PM on FruitCluster

Fig. 33. Auc Score

Cmd 2

```

1 sql_df = sqlContext.sql("SELECT * FROM table1")
2 df = sql_df.select("*").toPandas()
3 #y = df['64']
4 #df.drop('64', axis=1, inplace=True)
5 #x = df

```

► (1) Spark Jobs

Command took 6.11 seconds -- by jaiacouc@uncg.edu at 11/26/2021, 9:54:...

Fig. 31. Sql Import

▼ (1) MLflow run

Logged 1 run to an experiment in MLflow. [Learn more](#)

```

/databricks/python/lib/python3.8/site-packages/mlflow/models/signature.py:129:
eger column(s). Integer columns in Python cannot represent missing values. If y
ence time, it will be encoded as floats and will cause a schema enforcement err
fer the model schema based on a realistic data sample (training dataset) that i
declare integer columns as doubles (float64) whenever these columns may have mi
ng Values <https://www.mlflow.org/docs/latest/models.html#handling-integers-wit
inputs = _infer_schema(model_input)

```

Command took 8.66 seconds -- by jaiacouc@uncg.edu at 11/26/2021, 9:56:24 PM on FruitCluster

Fig. 32. Databricks Computation Time