Lab Image Sort

By: Tyler Bohrer

Sorting Images with Weka

Introduction:

In this lab I will go through the steps that will be needed for you to be able to have weka sort images of apples, oranges, and bananas. To do this we will need to create a csv file that will hold the attributes that will be needed for weka to be able to properly sort these three fruits. We will need to make sure each image is the same size, and if they are not resize them to the correct size. After the images have been added to the csv file we should be able to use weka to identify the images based on their rgb averages.

What you will need:

1. Python 2 or 3:

   I. Can be downloaded from here https://www.python.org/downloads/.

   II. You may also want to download an IDE that can make programming in python easier. I used pycharm which can be downloaded for free from here https://www.jetbrains.com/pycharm/download/

   III. If you do not like pycharm there are many IDEs for python. To find one that you may like just google for python IDEs, and it should come up with a list of IDEs.

2. Weka:

   I. Go to this previous lab that goes over how to install and use weka if you have not downloaded it or have little experience with weka https://drive.google.com/open?id=1g6mvy-nqxGq1PY2Q1Esnl2cttZ4sv9nJiDRR65oOfIc.

3. Apple, Oranges, and Banana images:

   Here is a folder that contains these images in various sizes https://drive.google.com/open?id=0B_1p15Nw1Vr6LUNiNnNkb0VxSUE

4. And a Computer

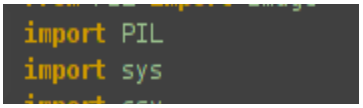
Making the Attributes for the images

In this lab we will have to make several attributes that will allow weka to identify apples from oranges or bananas. The first part is to make a python program that can resize all of the images to the same size. The first step is to make a project in python called WekaIdentifier, with a new python file called NewImage.

Image Sorter

What we will need to import

There are several packages that will be needed for this lab. package s in python are basically predefined functions that can be used to do certain takes. There are several packages that are preinstalled on python that you can use at anytime without installing them separately. To use them all that you have to do is type import followed by that package name. The first one we will need is PIL, this one is not included by default but in the next section I will go over how to install it. PIL will be used to alter the images that we will be using. We will also need to import sys and csv. Sys will be used to get input from the user, while csv will be used to save our files as a csv file. So this is what should be at the top of the file NewImage:



Using Python to resize the Images

Resizing the images is the first step in making sure that we have solid identifiers for weka to be able to use. We will use python to make this process take a lot less time than it would to do it manually change everyone picture. To make this program we will need to install a package for python called PIL or pillow. To do this on pycharm follow these steps.

1. Go to file and click settings
2. From there go to Project: WekaIdentifier.
3. Click the green cross in the right corner of the window.
4. Type pillow in the search bar
5. Select Pillow and click install package

If you are using a different IDE and do not know how to install from there you can install the package through your command line/ terminal. For the command line in windows

1. First start the command line by typing cmd in the start menu and selecting it.
2. Next use ‘cd’ to go to the folder that contains the version of python that you are using. Ex: For me it was cd Python/Python35/
3. Next you will need to run pip to install the package that you want to use. To do this type python -m pip install pillow, or the name of the package that you wish to install. Pip should install the package and any dependencies that it may require.

Image Sorter

You now have installed the PIL library for python.

Now we can start our code to resize the images. Let's first start by making a function and call it resize. To do define a function in python just type def before the name of the function then add parentheses after the name followed by a colon. Def is used by python to identify that the following is a function. The parentheses are used to place parameters in if you want to, wish we will. The colon will be used to help program now that it has reached the end of the function identification. By the end of the declaration it should look like this

```
def resize():
```

Now between the parentheses we will need to add several parameters. These parameters will be used to get important information about what we want the size to be to where we are going to be getting the images from. They are

```
def resize(path, name, amount, newSize):
```

Unlike many popular programming languages python does not need the programmer to declare a type for a variable, the program does it automatically when it runs, that is why we do not need to declare anything for these parameters but there names. It is important to know what each parameter does so I will explain what each one will hold.

1. Path - this parameter will hold the string value of where to find the images
2. Name - the name of the image to be resized, will be a string
3. Amount - the amount of images that are going to be resized, int
4. NewSize - the size that you want the new image to be scaled to, int

Start a new line to begin defining what the function will do. Make sure to indent the line underneath the funtion by 4 spaces or one tab, anytime you see a colon you should indent the next line after it. The first line we will have to increase the parameter amount by one. So we have this

```
amount = int(amount) +1 # increases the amount by one since we start iteration at 1 and not 0
```

The next step is to make sure that newsize is an int and to do this we just have do this simple line of code.

```
newsize = int(newsize)
```

This makes the program know that the parameter newsize is an integer, and if somehow the parameter is not an integer it will fail with a syntax error.

We are now ready to loop through all the images and to change their size. We will use a for loop to iterate over the number of images to resize. For this the for loop will look like this.

Image Sorter

```
for i in range(1,amount):
```

The i is a variable that has the starting value of one and will increase each time we get to the bottom of the loop until it reaches the max in the range which is amount. Next we need to open the image so that python can read it. To do this we will use Image from the PIL package. We will set a variable im to the image. To do this follow this line, make sure this line is indented like it was a function.

```
im = Image.open(path+name+str(i)+'.jpg', 'r') # takes the path plus image name and reads it.
# str(i) is added to the end of the name since we have the images named apple1-10 etc...
```

Next we need to get the percent of the image from its current size to the new size. This is done by this line

```
percent = (newWidth/float(im.size[0])) # this line gets the old size of the image and gets the #percent
that it will need to resize to in order to be a 256*256
```

Now we can get what the new size should be for the image.

```
finalSize = int((float(im.size[1]))*float(percent))# get the ratio that will be used to resize the image
```

The last thing to do is change the image and save it to its location.

```
im = im.resize((newWidth,finalSize),PIL.Image.ANTIALIAS) # resized using PIL
im.save(path+name+str(i)+'.jpg') # overwrites the old image with the resized image
```

This is the last step for the resize function and it should look something like this:

```
#-----------------------to 256*256-----------------------
def resize(path,name,amount,newSize):# name should be followed by the number of that item
    # path is the path to the image
    # amount is total amount of images
    amount = int(amount) +1
    newWidth = int(newSize)
    for i in range(1,amount):
        im = Image.open(path + name + str(i)+'.jpg', 'r')
        percent = (newWidth/float(im.size[0]))
        finalSize = int((float(im.size[1]))*float(percent))
        im = im.resize((newWidth,finalSize), PIL.Image.ANTIALIAS)
        im.save(path+name+ str(i) +'.jpg')
```

Now that the function is declared we will have to start it at the bottom of the code, but first let's add a couple more functions that will allow you to save the average rgb of each image to a csv file.

Image Average RGB

The first thing to do is create a function that will be used to write the header of our csv and then write the rgb averages of all our images. We should have five parameters for this function, one for the amount of

images, one for the path to the images, one to the path to the csv file, one for the image name, and the final one for how many of each type of image will there be.

```
def toCsv(amount, path, types, size, path2): # this will  be the starter function for average rgb
```

We will need to put the header in before we start with the data so we do not get duplicate headers in our csv file. To do this we will open the csv file at the location defined in path2.

```
with open(path2 +'test.csv', 'w', newLine=' ') as fh: # create a csv file with the name test.csv
```

Now make a new variable that will use the csv library to take the data given to it and use a delimiter to separate them. We should also make a variable that holds the data of our header for the csv file. Then finally we should write that data to the csv file. This can be accomplished with the following code:

```
writeto = csv.writer(fh)
data = [['red','green','blue','Type']] # this will be the header for our csv file
writeto.writerows(data) # write the data out in the first row of the csv file
```

Here is what it should look like so far, before we start the next function:

```
    with open(path2+'test.csv','w',newline='') as fh:# creat the header
        writeto = csv.writer(fh)
        data = [['red', 'green', 'blue', 'Type']]
        writeto.writerows(data)
    for x in range(amount):
        writeout(path,types[x],size,path2)
```

Next we need to write all of the images rgb averages to this csv file so we should make a loop that will go through how many different types we have. This can be done by using a for loop:

```
for x in range(amount): # loop through the amount of different types of images
   writeout(path,types[x],size,path2)# pass arguments to writeout that it needs
```

We then will call our new function writeout which we will now make:

First we need to make the function writeout with four arguments:

```
def writeout(path,type,size,path2): # path to image save, path2 to csv file, type is name of      # image,
and size is the amount of images we have to convert
```

We now need to open the csv file again so we use the same code as we did last time:

```
with open(path2 +'test.csv', 'w', newLine=' ') as fh: # create a csv file with the name test.csv
```

Now we probably want the types to be capitalized to make it look better in the csv file. We can do this with the following:

```
Type = str(type).capitalize() # here capitalize the first letter of the string to make it look better
```

Next increase the size by one since we will be looping through the images starting at the index of 1 and not 0:

```
size = int(size)+1
```

With all this done we can now loop through the images and save their rgb averages to the csv file test.csv. We will use another for loop to do this with the range from 1 to size:

```
for i in range(1,size):
```

Now we need to open the new image and get the rgb of that image:

```
im = Image.open(path+type+str(i)+'.jpg', 'r') # open the new image in read only
pix = list(im.getdata()) # get the rgb of the image and put it into a list called pix
pixflat = [x for sets in pix for x in sets] # use a double loop to get all the values of the image
```

We can now get each individual color and the amount of that color used in that image, since we know that the color red elements will be in the first 0 to 256*256 elements of pixflat and that green will be in the next 256*256+1 to 256*256*2 elements and that blue should be in the final 256*256*2+1 to 256*256*3 elements. To do this we just have to take those elements from pixflat:

```
red = pixflat[0:256*256] # get red elements
green = pixflat[256*256+1:256*256*2] # get green elements
blue = pixflat[256*256*2+1:256*256*3] # get the blue elements
# now get the averages of each to be save to the csv file
redav = sum(red)/(256*256) # get the red average
greenav = sum(green)/(256*256) # get the green average
blueav = sum(blue)/(256*256) # get the blue average
```

Now that we got the averages of the image all that is left is to put that into a list and saving it with the csv.writer:

```
writeto = csv.write(fh) # create writer
data = [[float(redav),float(greenav),float(blueav),Type]] # store the data in a list
writeto.writerows(data)# write out the data to the file, but do not overwrite it since we have it      # on
append and not just write
```

We are now done with the two functions now and they should look like this:

Image Sorter

```
    with open(path2+'test.csv', 'w', newline='') as fh:# creat the header
        writeto = csv.writer(fh)
        data = [['red', 'green', 'blue', 'Type']]
        writeto.writerows(data)
    for x in range(amount):
        writeout(path,types[x],size,path2)



def writeout(path,type,size,path2):
    with open(path2+'test.csv', 'a', newline='') as fh:
        Type = str(type).capitalize()
        size = int(size)+1
        for i in range(1,size):
            im = Image.open(path+type+str(i)+'.jpg', 'r')
            pix = list(im.getdata())
            pixflat = [x for sets in pix for x in sets]
            red = pixflat[0:256*256]
            red1 = sum(red)/(256*256)
            green = pixflat[(256*256)+1:256*256*2]
            green1 = sum(green)/(256*256)
            blue = pixflat[(256*256*2)+1:256*256*3]
            blue1 = sum(blue)/(256*256)
            writeto = csv.writer(fh)
            data = [[float(red1),float(green1),float(blue1),Type]]
            writeto.writerows(data)
```

Creating the Main for our program

We are almost done with this program, but we still need to connect all our functions together to be run without have to hard code in all the arguments. To do this let's make a function that will ask us if we want to resize the image, how many images we want to resize, and if we want to save the rgb average of the image to a csv file. Let's make a function called home. In this function we should ask the user these questions we mentioned above as well with so more information about the image:

```
def home():
   confrim1 = input("Enter yes if you want to resize the image to be 256*256") # get yes/no
   if(str(confirm1) =='yes'): # test if it is true
      size = input("Enter how many different types of images you want to resize")
      print("Enter the names of the image(s) separated by a space: ")
      name = [str(x) for x in sys.stdin.readline().split()]
      newSize = input("Enter the new size of the image") # get the size
      path = input("Enter the path: make sure to enter the end /\n For example /home/Documents/ would
be the path to my Documents folder:")
      amount = input("Enter the amount of images to resize:") # get amount
      for x in range(int(size)): # use the int(size) since size will normally be interpreted as a string
         resize(path,name[x],amount,newSize) # pass the arguments to the function resize
```

```
    else:
        print("Skipping resize step")
```

Input is used to get input from the keyboard after the text is displayed. This part will get the user's input that will let the program know whether or not to resize the images. It also gets the information needed to change that image and where to save it. It should look like this once it is typed out:

```
    confirm1 = input("Enter yes if you want to resize the images to be 256x256: ")
    if(str(confirm1) == 'yes'):
        size = input("how many types of images do you want to resize")
        print("Enter the name of the item(s) separated by a space: ")
        name = [str(x)for x in sys.stdin.readline().split()]
        newSize = input("Enter the new size of the image to be: example 256 = 256x256\n")
        path = input("Enter the path: make sure to enter the end /\nFor example /home/Documents/"
                     " would be the path to any item in the Documents folder: ")
        amount = input("Enter the amount of images you want to resize: ")
        for x in range(int(size)):
            resize(path,name[x],amount,newSize)
    else:
```

The next part will be to see if the user would like to convert the image to the rgb csv file, that will be used by weka to determine what the image is.

```
confirm2 = input("Enter yes if you want to convert the images into a csv file for weka")
if str(confirm) == 'yes':
    size = input("Enter the amount of image")
    amount = input("Enter how many different types are you wanting to convert: ") # different types in our case should be 3 since we have banana, apples, and oranges
    path = input("Enter the path to the images: ")
    path2 = input("Enter the path to the csv file: ")
    print("Enter the names of the different types one at a time, with spaces between them, then press enter")
    types = [str(x) for x in sys.stdin.readline().split()]# this reads input from the keyboard and stores it in a list, spaces separate the elements of the list
    toCsv(int(amount),path,types,int(size),path2)
    print("Now you should have a file in "+path2+" that has the file imageRGB.csv\nFinished")
```

Now that is done all we need to do is start it.

```
start = home()
```

At the end the home function should look like this, followed by start:

Image Sorter

```
    confirm1 = input("Enter yes if you want to resize the images to be 256x256: ")
    if(str(confirm1) == 'yes'):
        size = input("how many types of images do you want to resize")
        print("Enter the name of the item(s) separated by a space: ")
        name = [str(x)for x in sys.stdin.readline().split()]
        newSize = input("Enter the new size of the image to be: example 256 = 256x256\n")
        path = input("Enter the path: make sure to enter the end /\nFor example /home/Documents/"
                     " would be the path to any item in the Documents folder: ")
        amount = input("Enter the amount of images you want to resize: ")
        for x in range(int(size)):
            resize(path,name[x],amount,newSize)
    else:
        print("Going to next step")
    confirm2 = input("Enter yes if you want to convert the images into a csv file:")
    if(str(confirm2) == 'yes'):
        size = input("Enter the amount of the images: make sure all the different types has the same "
                     "amount of images\n")
        amount = input("Enter how many different types are you wanting to convert: ")
        path = input("Enter the path to the images: make sure to enter the end /\nFor example /home/Documents/"
                     " would be the path to any item in the Documents folder: ")
        path2 = input("Enter the path where you want to save the csv file: ")
        print("Enter the names one at a time, separated by a space, then press enter to continue: ")
        types = [str(x) for x in sys.stdin.readline().split()]
        toCsv(int(amount),path,types,int(size),path2)
        print("Now you should have a file in " + path2 + " that has the file imageRGB.csv\nFinished")

start = home()
```

And now you should be able to input all the paths through the console, converting the size of the image, and saving the image rgb through the same program.

Now that we have the csv file we can move on to using weka to classify these images based on their rgb averages.

Weka Sorting Images

Since we have the csv file all we need to do now is load up weka and test it. Go to the explorer tab in weka. Click open file and go to where you saved imageRGB.csv and select it, make sure you change the “files of type” section to be .csv files. Here you can see how all the data was sorted. By clicking types we should see that it is split into three even bars with 10 elements in each. Now since we have loaded the file let’s go to the classify tab. Here we can choose a different classifier by clicking on the choose button. We can test a random classifier for this dataset, I will go with ZeroR. This is what I got as my summary:

Image Sorter

```
Correctly Classified Instances          10               33.3333 %
Incorrectly Classified Instances        20               66.6667 %
Kappa statistic                          0
Mean absolute error                      0.4444
Root mean squared error                  0.4714
Relative absolute error                100      %
Root relative squared error            100      %
Total Number of Instances               30

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1.000    1.000    0.333      1.000    0.500      0.000    0.500     0.333     Apple
                 0.000    0.000    0.000      0.000    0.000      0.000    0.500     0.333     Orange
                 0.000    0.000    0.000      0.000    0.000      0.000    0.500     0.333     Banana
Weighted Avg.    0.333    0.333    0.111      0.333    0.167      0.000    0.500     0.333

=== Confusion Matrix ===

  a  b  c   <-- classified as
 10  0  0 |  a = Apple
 10  0  0 |  b = Orange
 10  0  0 |  c = Banana
```

It is easy to see that this classifier did not do a very good job of classifying the data, it actually just had them all be classified as an apple, which is evident by the fact that I got a 33%. Well maybe I can get a better result if I use a different classifier. J48 is usually a decent classifier to use in weka, so let's try that one. Here is its result:

```
Correctly Classified Instances          23               76.6667 %
Incorrectly Classified Instances         7               23.3333 %
Kappa statistic                          0.65
Mean absolute error                      0.2
Root mean squared error                  0.3892
Relative absolute error                 45      %
Root relative squared error             82.5706 %
Total Number of Instances               30

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.800    0.050    0.889      0.800    0.842      0.772    0.853     0.768     Apple
                 0.800    0.200    0.667      0.800    0.727      0.577    0.785     0.617     Orange
                 0.700    0.100    0.778      0.700    0.737      0.617    0.830     0.741     Banana
Weighted Avg.    0.767    0.117    0.778      0.767    0.769      0.655    0.822     0.708

=== Confusion Matrix ===

 a b c   <-- classified as
 8 1 1 | a = Apple
 1 8 1 | b = Orange
 0 3 7 | c = Banana
```

J48 did a much better job with classifying this image then the ZeroR classifier did, but I think we can still do a little better so let's try RandomTree. This classifier is similar to J48 as it uses a tree format to decide which leaves, values, are important and which ones are not. Here is its result:

```
Correctly Classified Instances          24               80      %
Incorrectly Classified Instances         6               20      %
Kappa statistic                          0.7
Mean absolute error                      0.1333
Root mean squared error                  0.3651
Relative absolute error                 30      %
Root relative squared error             77.4597 %
Total Number of Instances               30

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.900    0.100    0.818      0.900    0.857      0.783    0.900     0.770     Apple
                 0.700    0.150    0.700      0.700    0.700      0.550    0.775     0.590     Orange
                 0.800    0.050    0.889      0.800    0.842      0.772    0.875     0.778     Banana
Weighted Avg.    0.800    0.100    0.802      0.800    0.800      0.701    0.850     0.712

=== Confusion Matrix ===

 a b c   <-- classified as
 9 1 0 | a = Apple
 2 7 1 | b = Orange
 0 2 8 | c = Banana
```

This one was the best one of the three, no surprise since trees are very good at distinguishing noise, values that are not needed, and data that is important. You can test around if you like with the rest of the classifiers and see if you can find one better than RandomTree.

Conclusion

We can clearly see that it is possible to take several images and convert some information from them and then have weka iable to sort them with an 80% accuracy, on some classifiers. Our python program was able to take any sized image and convert it to a smaller, or larger, 256x256 image, and then have its average rgb be saved to a csv file, that was latter used in weka to sort the images. This way of sorting apples, oranges, and bananas is not perfect however. Take for example a green apple it would probably not be matched with the group apple, since its color is not red. So to have a better classification that could identify fruit we would most likely need to have better attributes, which would mean taking more than just the averages of rgb, but for this project it did a good job at just being able to identify between a red apple, a banana, and an orange.