Biostatistics (STA5195) Lab 4 Report

Rice Seed Classification

Christopher Rutherford

September 29, 2020

Abstract

Computer algorithms have been used to process and "see" images to learn what objects are contained in images. [1] More often than not, these machine and deep learning algorithms are used to identify and classify large quantities of images. We can simplify these algorithms in order to classify images of rice seeds as proper or broken. By assessing the brightness of each pixel and converting the image to pure black and white, we present a method for classifying these rice seeds.

1 Introduction

Training machine and deep learning algorithms for image classification is often computationally and time intensive, particularly when dealing with larger datasets. Instead of using a specific algorithm, we will instead analyze a set of 397 images of rice seeds - some broken, and the rest normal ("proper").

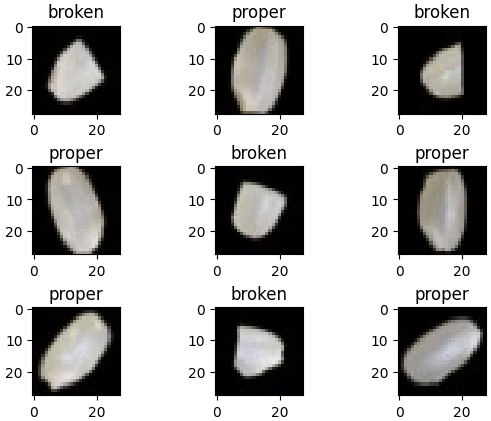

Figure 1: Assortment of rice seed images

2 Theory

Because the rice seed images are quite simple images - a white/gray seed on a black background - we can take advantage of this when working with these images. We convert each image from RGB to grayscale so that each pixel is only defined by how bright/dark it is. The value 0 is associated with completely black pixels while 255 is associated with white pixels. We use scikit-image's `threshold_otsu` function to map anything in between 0 and 255 to either be black if it is closer to 0, and white if it is closer to 255. An example of how this conversion works:

Figure 2: Conversion: RGB → grayscale → black & white

3 Procedures

To begin, we import the libraries and functions that we will be using throughout this lab:

```
import numpy as np
import math, os, sys
import itertools

import matplotlib.pyplot as plt
plt.style.use('default')
from scipy import ndimage

#functions for image processing
from skimage import measure, morphology
from skimage.io import imsave, imread
from skimage.color import rgb2gray
from skimage.filters import threshold_otsu
from skimage.transform import resize

from sklearn import svm, datasets
from sklearn.metrics import confusion_matrix
import pandas as pd
```

And now we need to download the dataset that we are going to be using:

```
!#we will use the rice seed image datset
!apt-get install subversion > /dev/null
```

```
!svn export https://github.com/totti0223/
   deep_learning_for_biologists_with_keras/trunk/notebooks/data/
   image image > /dev/null
```

We can make sure the images imported properly by plotting one of them:

```
#let's visualize a single file
image = imread("image/train/proper/100.jpg") #read in the image
plt.figure(figsize=(3,3)) #plot the image on a 3x3 figure
plt.imshow(image)
```

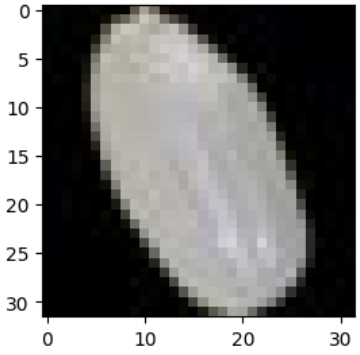

Figure 3: Proper rice seed

Since we have the images that we need, we can start preparing our data for our algorithm by splitting it into train/test splits. First we prepare the training set:

```
#load training dataset
X_train = [] #create empty lists for training data
y_train = []

#cycle through the images
for root, dirs, files in os.walk("image/train"):
    files = [x for x in files if x.endswith(".jpg")]
    for file in files:
        image_path = os.path.join(root, file)

        image = imread(image_path)/255.
        image = resize(image,(28,28)) #make all the images 28x28
        X_train.append(image)
        category = os.path.split(root)[-1]
        if category == "proper":
            y_train.append(0) #0 for proper rice seeds
        else:
            y_train.append(1) #1 for broken rice seeds
X_train = np.array(X_train)
y_train = np.array(y_train)
```

A good portion of this code is fail-safes to avoid importing the wrong files, but all we are really doing is adding images to `X_train`, and making the corresponding `y_train` value 0 for proper rice seeds and 1 for broken seeds. We also do the same for the testing set:

```
#load test dataset
X_test = [] #create empty lists for testing data
y_test = []

for root, dirs, files in os.walk("image/test"):
    files = [x for x in files if x.endswith(".jpg")]
    for file in files:
        image_path = os.path.join(root, file)

        image = imread(image_path)/255.
        image = resize(image,(28,28))
        X_test.append(image)
        category = os.path.split(root)[-1]
        if category == "proper":
            y_test.append(0)
        else:
            y_test.append(1)
X_test = np.array(X_test)
y_test = np.array(y_test)

print("train dataset shape is:", X_train.shape,y_train.shape)
print("test dataset shape is:", X_test.shape,y_test.shape)
```

```
train dataset shape is: (377, 28, 28, 3) (377,)
test dataset shape is: (20, 28, 28, 3) (20,)
```

Before converting the image to pure black and white, we convert it from RGB to grayscale:

```
#gray conversion
gray = rgb2gray(image)
print(gray.shape)
plt.figure(figsize=(3,3))
plt.imshow(gray, cmap=plt.cm.gray)
plt.title("gray converted")
```

Then, by using the threshold function, pixels that are darker than the specified threshold are made black, while those above the threshold are made white.

```
#binary conversion
# white/gray pixels above a certain threshold are made white
# black/darker gray pixels below the threshold are made black
threshold = threshold_otsu(gray)
binary = gray > threshold
plt.figure(figsize=(3,3))
plt.imshow(binary, cmap=plt.cm.gray)
```

We can then calculate the area of the rice seed in each image by counting how many white squares are in the image.

```
# calculates the area and lengths of major and minor axes
label_im, nb_labels = ndimage.label(binary)
regionprops = measure.regionprops(label_im, intensity_image=gray)
regionprop = regionprops[0]

print("area is",regionprop.area)
print("major axis length is", regionprop.major_axis_length)
print("minor axis length is", regionprop.minor_axis_length)
```

```
area is 355
major axis length is 28.54477065932296
minor axis length is 15.906083837305355
```

For the sake of convenience, we can wrap this up into a function and test it out.

```
#bundling the above into a function
def quantify_area(image):
    gray = rgb2gray(image)
    threshold = threshold_otsu(gray)
    binary = gray > threshold
    label_im, nb_labels = ndimage.label(binary)
    regionprops = measure.regionprops(label_im,
                                      intensity_image=gray)
    regionprop = regionprops[0]
    area = regionprop.area
    return area

#test
area = quantify_area(image)
print(area)
```

```
355
```

We now create new train/test splits that contain the values of the area of each rice seed's image.

```
X_train_area = []
for image in X_train:
    area = quantify_area(image)
    X_train_area.append(area)

X_test_area = []
for image in X_test:
    area = quantify_area(image)
    X_test_area.append(area)
```

```
#check the calculated data area value of training dataset
plt.scatter(range(len(X_train_area)), X_train_area, c=y_train, cmap=
    "jet")
plt.xlabel("rice seed images")
plt.ylabel("area (px)")
```

```
plt.show()
```

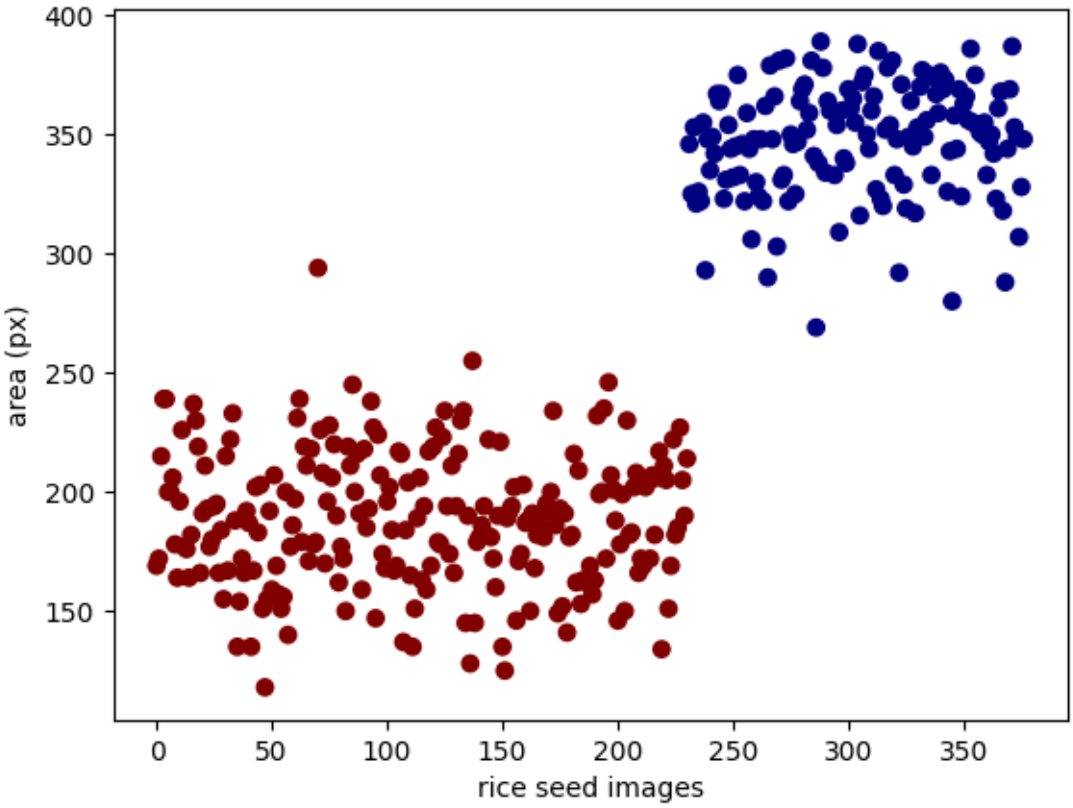

Figure 4: Area of rice seeds in training set (blue = regular, red = broken)

4 Analysis

Now that we have a visual of how our rice seeds are distributed in terms of white pixel area, we can create our own algorithm to classify the seeds. Based on the above plot, we add a horizontal line at x=260 to separate the two groups.

```
#define an area threshold that can separate the two classes
area_threshold = 260

#classify whether the image is a proper seed or a broken seed
   according to the area_threshold value
train_y_pred = []
for area in X_train_area:
    if area > area_threshold:
        train_y_pred.append(0)
    else:
        train_y_pred.append(1)

```

```
#plot scatter with threshold line
plt.figure(figsize=(5,3))
plt.scatter(range(len(X_train_area)),X_train_area,c=y_train,cmap=plt
    .cm.coolwarm)
plt.axhline(y=area_threshold)
plt.title("blue:proper seed, red: broken seed")
plt.show()

#calculate confusion matrix
cnf = confusion_matrix(y_train, train_y_pred)

#confusion matrix in figure
plt.figure(figsize=(3,3))
plot_confusion_matrix(cnf, classes=["proper","broken"])

plt.show()
```

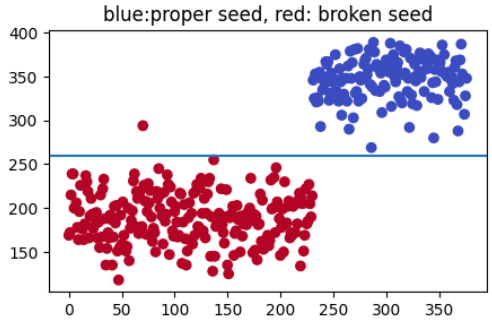

Figure 5: Rice seed areas with threshold line

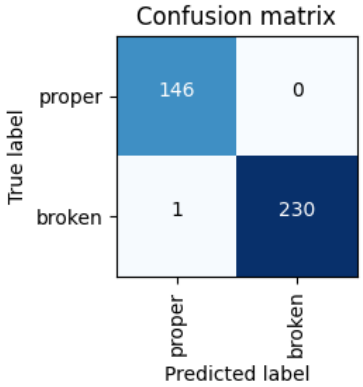

Figure 6: Confusion matrix for training set

Our line seems to do about as well as it could for a simple horizontal line - only one broken seed is incorrectly classified as proper. Simply adding a slope to this would fix this issue, but we're keeping things quite simple.

Now we evaluate this on the test set:

```
#evaluate it with the test dataset
test_y_pred = []
for area in X_test_area:
    if area > area_threshold:
        test_y_pred.append(0)
    else:
        test_y_pred.append(1)

#plot scatter with threshold line
plt.figure(figsize=(5,3))
plt.scatter(range(len(X_test_area)),X_test_area,c=y_test,cmap=plt.cm
   .coolwarm)
plt.axhline(y=area_threshold)
#plt.plot([100,0],[100,350],'k-',lw=2)
plt.show()

#calculate confusion matrix
cnf = confusion_matrix(y_test, test_y_pred)

#confusion matrix in figure
plt.figure(figsize=(3,3))
plot_confusion_matrix(cnf, classes=["proper","broken"])
plt.show()
```

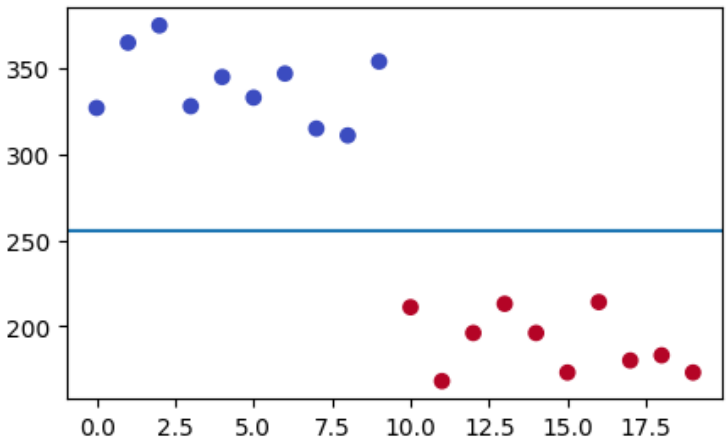

Figure 7: Rice seed areas with threshold line (testing set)

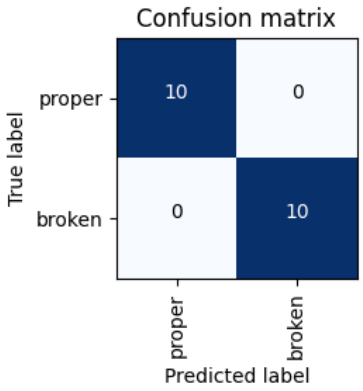

Figure 8: Confusion matrix for testing set

With such little error in the training set, it's no surprise that this managed to perform well on the testing set as well.

We can verify how this works on another, single image from the test set and a threshold of 350:

```
#build a classifier

def manual_classifier(image,area_threshold):
    gray = rgb2gray(image)
    threshold = threshold_otsu(gray)
    binary = gray > threshold
    label_im, nb_labels = ndimage.label(binary)
    regionprops = measure.regionprops(label_im, intensity_image=gray
   )
    regionprop = regionprops[0]
    area = regionprop.area
    if area > area_threshold:
        return 0
    else:
        return 1

# get a image from test dataset
#value must be lower than the size of the test dataset(20-1)
n = 10
image = X_test[n]
label = y_test[n]
area_threshold = 350
prediction = manual_classifier(image,area_threshold)

plt.imshow(image)
print("correct label is: ",label)
print("predicted label is: ",prediction)
```

```
correct label is:  1
predicted label is:  1
```

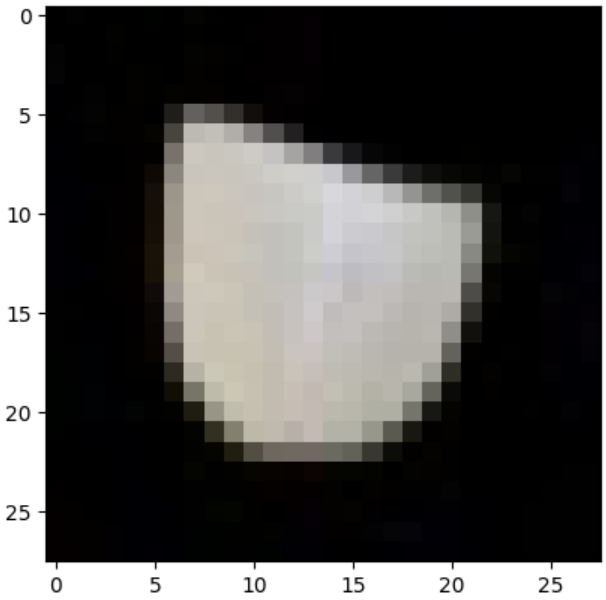

Figure 9: Seed image used for single test

With an area threshold of 350 pixels, our classifier was able to correctly classify this image.

5 Conclusions

Without using any sort of high-level machine learning or deep learning algorithms, we were able to manually create a very accurate rice seed classifier by using a simple threshold. If the image contained more white pixels than the threshold, it would be considered proper, and containing less than the threshold would be considered broken. The naive classifier performed quite well, and it would likely be able to classify the seeds perfectly just by adding a second parameter: the slope. Regardless, this is a testament to how simplicity can sometimes be just as useful as very complex models.

References

[1] *Image analysis*, available at https://en.wikipedia.org/wiki/Image_analysis.