Exercise 3

May 13, 2019

Wajid Ghafoor und Benjamin Ostendorf

[26]:
```
from PIL import Image, ImageColor
import numpy as np
from PIL import ImageFilter
import math as m
from functools import reduce
```

The features we selected are the minima of the red and green channel of the images, as well as the variance on both channels. Furthermore we selected the mean of the pixel values in the blue channel as our fifth feature. We also thought about using an edge detection filter on the images. With this we could also use the mean or median of the edge gradient or the absolut number of edges as features. Since using all edges proofed to be of no advantage, we didn't follow up on the idea. Using only edges with a gradient over a certain threshold might increase our results, but we haven't tried it out.

The following represents our model parameters (phi, mue for both classes and our covariance matrix).

[27]:
```
#We devided the avialable data into traning and testing sets.
#The first half is used for training while the second half is used to test␣
 ↪against.



print('Phi (equal for both classes):\n', phi(positive_features_half,␣
 ↪negative_features_half), '\n')

print('Mue for positive examples:\n', mue(positive_features_half), '\n')

print('Mue for negative examples:\n', mue(negative_features_half), '\n')

print('Covariance matrix:\n', covariance_matrix(positive_features_half,␣
 ↪negative_features_half), '\n')
```

```
Phi (equal for both classes):
 0.5

Mue for positive examples:
 [96.93333333333334, 50.733333333333334, 309.82349737975824, 593.8893428096064,
147.94641203703705]
```

```
Mue for negative examples:
 [141.13333333333333, 103.73333333333333, 91.31537382651751, 123.02140199009776,
127.90520833333332]

Covariance matrix:
 [[  306.02222222   222.90888889  -742.40036764  -626.57344732
    123.4933642 ]
 [  222.90888889   271.59555556  -770.92655752 -1202.79414514
    115.52529707]
 [ -742.40036764  -770.92655752  7242.53451075 13000.43416291
    -86.82840413]
 [ -626.57344732 -1202.79414514 13000.43416291 29147.35052597
    169.56172051]
 [  123.4933642    115.52529707   -86.82840413   169.56172051
    113.20803166]]
```

[28]:
```python
def extrema_red(image):
    return image.getextrema()[0][0]
def extrema_green(image):
    return image.getextrema()[1][0]
def variance_red(image):
    return np.var(image.split()[0])
def variance_green(image):
    return np.var(image.split()[1])
def mean_blue(image):
    return np.mean(image.split()[2])
```

[14]:
```python
def create_feature(image):
    er = extrema_red(image)
    eg = extrema_green(image)
    vr = variance_red(image)
    vg = variance_green(image)
    mb = mean_blue(image)
    return [er,eg,vr,vg,mb]

def create_features(img_list):
    return list(map(create_feature, img_list))
```

[15]:
```python
pos_image_list = [Image.open("positives/p0" + str(i) + ".png") if i < 10 else␣
 ↪Image.open("positives/p" + str(i) + ".png")
                  for i in range(1,31)]
neg_image_list = [Image.open("negatives/n0" + str(i) + ".png") if i < 10 else␣
 ↪Image.open("negatives/n" + str(i) + ".png")
                  for i in range(1,31)]

positive_features_full = create_features(pos_image_list)
```

```
negative_features_full = create_features(neg_image_list)

#split dataset into traning and testing
positive_features_half = create_features(pos_image_list[0:15])
negative_features_half = create_features(neg_image_list[0:15])
```

[16]:
```
def phi(positive_features, negative_features):
    return float(len(positive_features)) /␣
 ↪float((len(negative_features)+len(positive_features)))

def mue(features):
    sum_of_features = reduce(lambda a,b: [x+y for x,y in zip(a,b)], features)
    return list(map(lambda x: float(x/len(features)), sum_of_features))

def covariance_matrix(positive_features,negative_features):
    mue0 = mue(negative_features)
    number_of_features = len(negative_features[0])
    neg_mat_list = list(map(lambda x: np.matmul(np.reshape([a-b for a,b in␣
 ↪zip(x, mue0)],  (number_of_features, 1)), np.reshape([n-m for n,m in zip(x,␣
 ↪mue0)], (1, (number_of_features)))), negative_features))

    mue1 = mue(positive_features)
    pos_mat_list = list(map(lambda x: np.matmul(np.reshape([a-b for a,b in␣
 ↪zip(x, mue1)],  (number_of_features, 1)), np.reshape([n-m for n,m in zip(x,␣
 ↪mue1)], (1, (number_of_features)))), positive_features))

    mat_list = np.append(neg_mat_list, pos_mat_list, axis=0)
    #print(mat_list)
    cov_mat = mat_list[0]
    for i in range(1,len(mat_list)):
        cov_mat = np.add(cov_mat,mat_list[i])

    return np.multiply(cov_mat,float(1.0/float(len(mat_list))))
```

[17]:
```
def predict(image,positive_features,negative_features):
    n = len(positive_features)+ len(negative_features)
    phi_1 = phi(positive_features,negative_features)
    phi_0 = phi(negative_features, positive_features)
    #print(phi_1)
    #print(phi_0)
    mue0 = mue(negative_features)
    mue1 = mue(positive_features)
    cov_mat = covariance_matrix(positive_features, negative_features)
    image_features = create_feature(image)
    f_len = len(image_features)
    function_prefix = 1.0/((2.0*m.pi)**float(n/2.0)*np.linalg.
 ↪det(cov_mat)**float(1.0/2.0))
    in_cov_mat = np.linalg.inv(cov_mat)
    f_diff_0 = np.reshape([x-y for x,y in zip(image_features,mue0)], (1, f_len))

    mat0_1 = np.multiply(-0.5, f_diff_0)
    mat0_2 = np.matmul(mat0_1, in_cov_mat)
   # print(np.matmul(mat0_2, np.reshape(f_diff_0, (f_len, 1))))
    px_0 = function_prefix*m.e**(np.matmul(mat0_2, np.reshape(f_diff_0, (f_len,␣
↪1)))[0][0])

    f_diff_1 = np.reshape([x-y for x,y in zip(image_features,mue1)], (1, f_len))

    mat1_1 = np.multiply(-0.5, f_diff_1)
    mat1_2 = np.matmul(mat1_1, in_cov_mat)
   # print(np.matmul(mat1_2, np.reshape(f_diff_1, (f_len, 1))))
    px_1 = function_prefix*m.e**(np.matmul(mat1_2, np.reshape(f_diff_1, (f_len,␣
↪1)))[0][0])

    #print(phi_1, mue0, mue1, cov_mat)
    if phi_0 * px_0 > phi_1 * px_1:
        return "negativ, u are not infected"
    else:
        return "positive, chargas Parasite found!!!"
```

[18]:
```
#Classify all positive images. Learned with full dataset.
for i in range(1,30):
    if i < 10:
        print(predict(Image.open("positives/p0" + str(i) + ".png"),␣
↪positive_features_full,negative_features_full))
    else:
        print(predict(Image.open("positives/p" + str(i) + ".png"),␣
↪positive_features_full,negative_features_full))
```

```
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
```

```
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
```

[19]:
```
#Classify all negative images. Learned with full dataset.
for i in range(1,30):
    if i < 10:
        print(predict(Image.open("negatives/n0" + str(i) + ".png"),␣
 ↪positive_features_full,negative_features_full))
    else:
        print(predict(Image.open("negatives/n" + str(i) + ".png"),␣
 ↪positive_features_full,negative_features_full))
```

```
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
positive, chargas Parasite found!!!
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
```

```
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
```

[20]:
```
#Classify all positive images. Learned first half dataset.
for i in range(1,30):
    if i < 10:
        print(predict(Image.open("positives/p0" + str(i) + ".png"),␣
 ↪positive_features_half,negative_features_half))
    else:
        print(predict(Image.open("positives/p" + str(i) + ".png"),␣
 ↪positive_features_half,negative_features_half))
```

```
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
positive, chargas Parasite found!!!
negativ, u are not infected
positive, chargas Parasite found!!!
```

[21]:
```
#Classify all negative images. Learned with first half dataset.
for i in range(1,30):
    if i < 10:
        print(predict(Image.open("negatives/n0" + str(i) + ".png"),␣
↪positive_features_half,negative_features_half))
    else:
        print(predict(Image.open("negatives/n" + str(i) + ".png"),␣
↪positive_features_half,negative_features_half))
```

```
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
positive, chargas Parasite found!!!
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
negativ, u are not infected
```