FINAL REPORT: SKIN CANCER: MALIGNANT VS. BENIGN

CENG 3521, DATA MINING

Emre Ertürk, Ülkem Güleç
emreerturk3@posta.mu.edu.tr, ulkemgulec@posta.mu.edu.tr

Friday 22nd January, 2021

Abstract

Skin cancer is the out-of-control growth of abnormal cells in the epidermis, the outermost skin layer, caused by unrepaired DNA damage that triggers mutations. Cancer begins when healthy cells change and grow out of control, forming a mass called a tumor. A tumor can be cancerous or benign. A cancerous tumor is malignant, meaning it can grow and spread to other parts of the body. That report includes brief information on how to identify malignant and benign moles using image segmentation and image classification method

1 Introduction

Image segmentation is the process of partitioning an image into multiple different regions (or segments). The goal is to change the representation of the image into an easier and more meaningful image. We informed you about the skin cancer dataset in the first topic of our report. After that part, we have two main parts, one of them is image segmentation and another one is image classification. We mentioned clustering algorithms in the image segmentation part. Our report's last topic is about classifying segmented images.

2 Skin Cancer Dataset

This dataset contains a balanced dataset of images of benign skin moles and malignant skin moles. The malignant data has 1197 images and benign data has 1440 images. These images are 224x224 image dimensions.

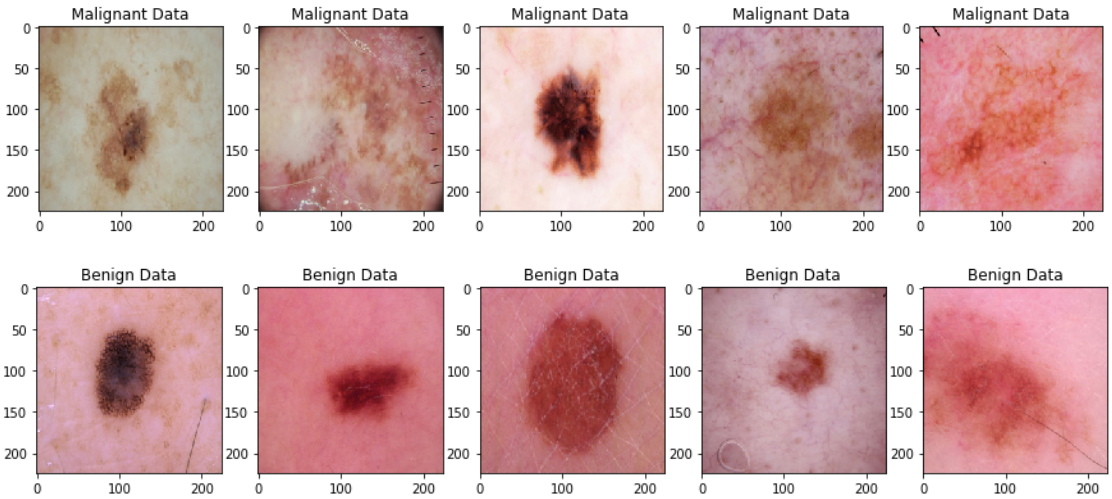

Figure 1: Examples of malignant data and benign data

3 Image Segmentation

Image segmentation involves converting an image into a collection of regions of pixels that are represented by a mask or a labeled image. By dividing an image into segments, you can process only the important segments of the image instead of processing the entire image.

3.1 k-means Clustering

K-means clustering is one of the popular unsupervised machine learning algorithms that groups objects into k groups based on their characteristics. The grouping is done minimizing the sum of the distances between each object and the group or cluster centroid.

Figure 2: Example of clustered data

3.2 Elbow Algorithm

The Elbow method is a very popular technique and the idea is to run k-means clustering for a range of clusters k and for each value, we are calculating the sum of squared distances from each point to its assigned center. When the center are plotted and the plot looks like an arm then the "elbow" is the best value of k. In this project, we take an image to calculate the elbow number and as you can see in Figure 3. We found the best value of k as 2. So we assigned this number to the k_cluster variable to use later.

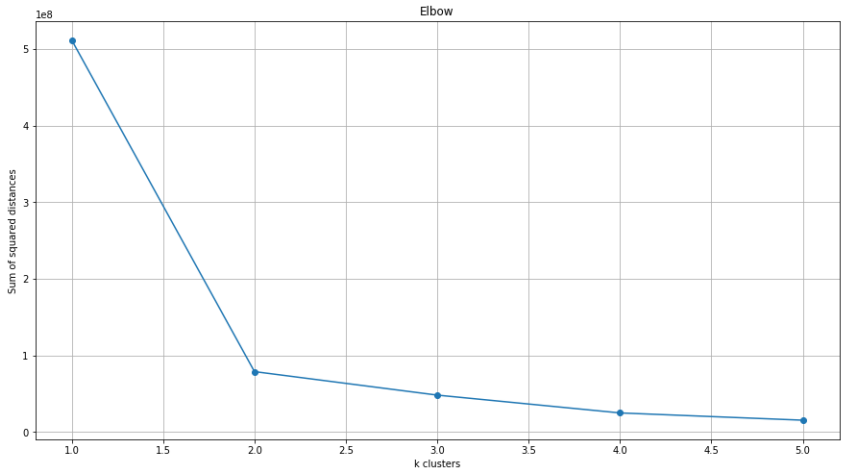

Figure 3: Elbow Graph

3.3 Color Processing

We did some processes on images to put a mask. That mask helps us to classifying images.

3.3.1 Rgb to Grey

In this task firstly, we chose a malignant image. We converted this image into two colors using the rgb2grey() method. Then we gave this two-color image and optimal k_clusters number as a parameter to the d2Kmeans function. Also, the d2Kmeans function is to calculate the k-means of a given image.

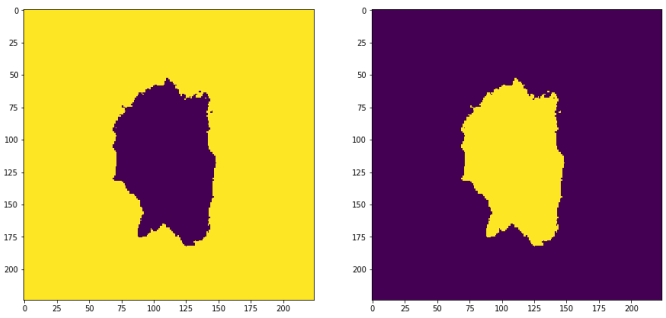

Figure 4: An image converted to two color

3.3.2 Masked Image

We applied the mask to an image. We used a mean filter algorithm to get a more smooth image. Then, we used the threshold_otsu algorithm to get the binary image

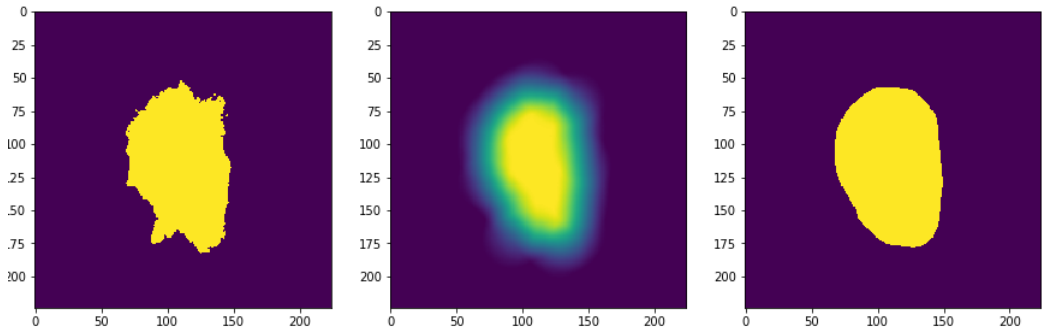

Figure 5: The left one is the just grey image, the middle one is the median filtered image, the right one is the binary image

We deleted the dark parts of the image. So we created a mask and we applied that mask to every image.

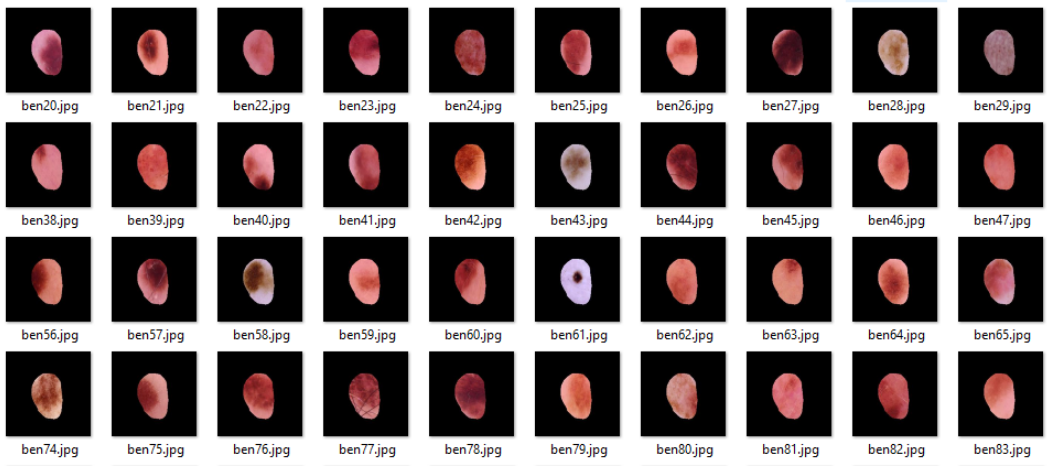

Figure 6: Masked images

4 Image Classification

Image classification is a supervised learning problem: define a set of target classes, and train a model to recognize them using labeled example photos.

4.1 Hu Moments

Firstly we converted image to grayscale using color.BGR2GRAY(). we used cv2.HuMoments() function to extract Hu Moments features from the image. The argument to this function is the

moments of the image cv2.moments() flattened. It means we computed the moments of the image and convert it to a vector using flatten().

```
def fd_hu_moments(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    feature = cv2.HuMoments(cv2.moments(image)).flatten()
    return feature
```

Figure 7: fd_hu_moments funtion

4.2 Haralick Textures

We converted our color image into a grayscale image as haralick feature descriptor expect images to be grayscale. We used mahotas.features.haralick() to extract Haralick Texture features from the image.

```
def fd_haralick(image):
    # convert the image to grayscale
    gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
    haralic = mahotas.features.haralick(gray).mean(axis=0)
    return haralic
```

Figure 8: fd_hu_moments funtion

4.3 Color Histogram

We used the cv2.calcHist() function to extract Color Histogram features from the image. The arguments it expects are the image, channels, mask, histSize and ranges for each channel (typically 0-256). Then we normalized the histogram using normalize() function of OpenCV and return a flattened version of this normalized matrix using flatten().

```
def fd_histogram(image, mask=None):
    # convert the image to HSV colors-space
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    #COMPUTE THE COLOR HISTOGRAM
    hist  = cv2.calcHist([image],[0,1,2],None,[bins,bins,bins],
                         [0, 256, 0, 256, 0, 256])
    # normalize the histogram
    cv2.normalize(hist, hist)
    # return the histogram
    return hist.flatten()
```

Figure 9: fd_hu_moments funtion

4.4 Features of an image

we extracted the three global features and concatenate these three features using NumPy's np.hstack() function. These features came from color histogram, haralick textures and hu moments.

```
# Concatenate global features
global_feature = np.hstack([fv_histogram, fv_haralick, fv_hu_moments])
```

Figure 10: Concatenation of features

4.5 Labels of our dataset

We used LabelEncoder() method to encode our labels in a proper format. Our labels are 'mal' and 'ben', these are our subfolders name and types of moles.

```
le = LabelEncoder()
target = le.fit_transform(labels)
```

Figure 11: LabelEncoder()

4.6 Training Data

After all of these processes, we get proper data to use in classification methods. We compared seven different classification method using cross_val_score(). We observed the Random Forest Classifier method has the best accuracy score. .

```
LR: 0.809944 (0.019807)
LDA: 0.805744 (0.017590)
KNN: 0.816686 (0.018476)
CART: 0.766537 (0.031915)
RF: 0.827210 (0.027503)
NB: 0.625357 (0.025411)
SVM: 0.700362 (0.029948)
```

```
('LR', LogisticRegression(random_state=seed)))
('LDA', LinearDiscriminantAnalysis()))
('KNN', KNeighborsClassifier()))
('CART', DecisionTreeClassifier(random_state=seed)))
('RF', RandomForestClassifier(n_estimators=num_trees, random_state=seed)))
('NB', GaussianNB()))
('SVM', SVC(random_state=seed)))
```

Figure 12: Comparisons of the seven classification

We split our data that randomly selected 70% tuples are used for training while 30% are used for testing. We applied our classification method and we fitted our train data to model. Then we calculated predictions using predict() method.

4.7 Test Data

At the end of the project we tested our algorithm using samples from segmented dataset. As you can see at the Figure 13 mostly results are okey but some results are wrong.

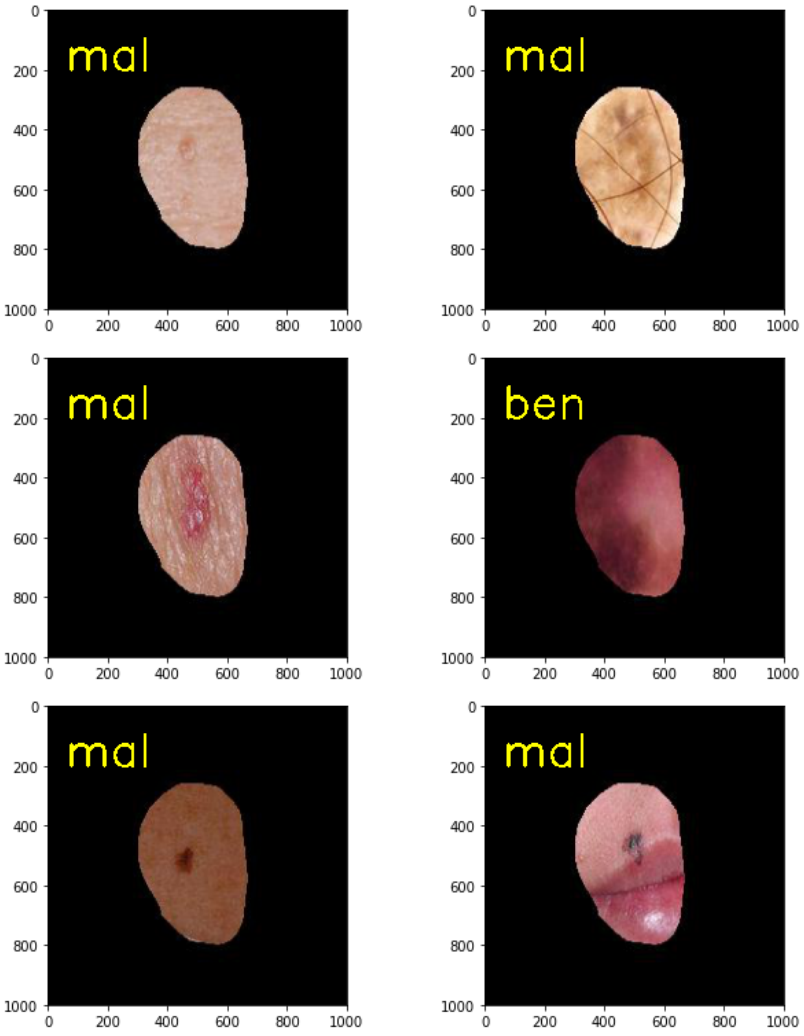

Figure 13: These images taken from the malignant dataset

5 Conclusion

In conclusion, image segmentation methods help us to get better classifying results. We used the clustering method in the image segmentation part. After the segmentation part, we decided which classification method is the most proper for our dataset. We compared seven different classification methods using a cross-validation score. Before doing that, we converted data to readable for classification methods. As a result, we learned the image segmentation and classifying algorithms.