Using Logistic Regression Approach for Color Segmantation

Jingpei Lu
Jacobs School of Engineering
University of California, San Diego
jil360@ucsd.edu

***Abstract*—This paper presented my work of using a discriminative model to segment the colors in a single image. Our goal is to use the logistic regression to find the decision boundary between two classes.**

Keywords—Logistic Regression, Color Segmentation

I. Introduction

In many applications of computer vision or image procession, the goal is to extract information from image data. Although humans can easily segment the objects in an image, performing segmentation is not straight forward for computers. Color segmentation is to a very useful technique to extract regions of interest in an image by segmenting the colors. It is widely used in computer vision and machine learning problems nowadays. It is a fundamental process of image processing. Successfully extract the region of interest can aid the higher-level process such as classification and detection.

Numerous segmentation techniques have been proposed in this literature such as thresholding or edge detection. However, most of the techniques are done on gray scale image. In this paper, we are going to solve the segmentation problem using colors, which is classified using a logistic regression model. It uses a discriminative model for discrete labels and is trained using gradient descent.

In our case, we are trying to separate the blue barrels from a color image. Our approach is to train a discriminative model to classify each pixel into "blue barrel" or "not blue barrel". Then we segment these "blue barrel" regions and determine the "barrelness" of these regions. We will split our data into training set and test set. We will train our model on training samples and test on test samples.

II. Problem Formulation

A. Logistic Regression

The Logistic Regression is a discriminative model $p(y|X,w)$ for the discrete labels y that is a product of sigmoid function

$$p(Y|X,w) = \prod_{i=1}^{n} \sigma(y_i x_i^T w) \tag{1}$$

where $X \in \mathbb{R}^{n\times 3}$ and $Y \in \mathbb{R}^{n\times 1}$ is our training data and corresponding labels of n examples. Each label $y \in \{-1,1\}$, where 1 indicate it is blue and -1 indicate it is not blue. Each data point $x \in \mathbb{R}^3$ is a vector consist RGB value of a pixel. The weight of the discriminative model is $w \in \mathbb{R}^3$. $\sigma(x)$ is sigmoid function where

$$\sigma(x) = \frac{1}{1+\exp(-x)} \tag{2}$$

Our problem is to obtain the optimal weights of this discriminative model. To solve this, we are going to use the Maximum Likelihood Estimation (MLE), which leads to

$$w_{MLE} = arg\max_{w} p(Y|X,w) \tag{3}$$

After we obtain the optimal weights w^*, we can predict the label for a given example x^* as

$$y^* = \begin{cases} -1, & x^{*T}w^* < 0 \\ 1, & x^{*T}w^* \geq 0 \end{cases} \tag{4}$$

where y^* is the prediction of a given example x^*. If $y^*$is 1, the pixel is classified as blue and y^* = -1 indicates the pixel is not blue.

B. Find blue barrels

For a given color image I, we can present each pixel as $I(i,j) \in \mathbb{R}^3$ where i indicates ith row and j indicates jth column of the image I. We want to extract the blue region in image I by applying our discriminative model to each pixel.

After we classified every pixel in image I, we can get a mask of the image. Then we can extract the regions of interest from the mask and determine the "barrelness" of those regions. Finally we want to draw a bounding box for each blue barrel in the image and output the coordinates of the bounding box $(x_{min}, y_{min}, x_{max}, y_{max})$, where (x_{min}, y_{min}) is the coordinate of the top left corner of the bounding box and (x_{max}, y_{max}) is the coordinate of the bottom right corner of the bounding box.

III. Technical Approach

A. Obtain the training data

Our training set consists color images which contain blue barrels. To obtain the training samples, we are going to use *roipoly* [1] to hand-label appropriate regions in the images with labels "blue" and "not blue".

Roipoly only takes in grayscale image, so we first convert the color images to grayscale. To obtain the positive samples, we need to draw polygons within the blue barrels. Then, we can obtain the masks of selected regions from *roipoly*. Finally, we apply the masks on the original color images and extract the pixel value from the masked images, and label them as y = 1.

To obtain the negative samples, we use the same process, but instead of drawing polygons within blue barrels, we draw polygons on not blue barrel regions, and we label them as y = -1.

B. Train the model

The discriminative model $p(Y|X,w)$ we are going to use is the logistic regression stated in (1). Combined with the sigmoid function stated in (2), we can write our model as

$$p(Y|X,w) = \prod_{i=1}^{n} \frac{1}{1+\exp(-y_i x_i^T w)} \tag{5}$$

We want to optimize the weights w using the maximum likelihood estimation as we stated in (3). Noting that applying

logarithm to the argument of maxima does not affect the result, so we can write our MLE problem as

$$w_{MLE} = arg\max_{w} \log p(Y|X,w) \quad (6)$$

Combined (6) with (5), we can get

$$w_{MLE} = arg\max_{w} \sum_{i=1}^{n} \log\left(\frac{1}{1+\exp(-y_i x_i^T w)}\right) \quad (7)$$

$$w_{MLE} = arg\min_{w} \sum_{i=1}^{n} \log(1+\exp(-y_i x_i^T w)) \quad (8)$$

So now, solving this MLE problem is equivalent to find the minimum of this function as we showed in (8). One way to find the global minimum of a function is checking this function is convex, and then taking the derivative of this function and set to 0. We know $\log(1+exp(x))$ is convex and $-y_i x_i^T w$ is an affine function. The sum of convex functions is also a convex function. Then, we just need to take the derivative of the function in (8) with respect to w and set it to 0. In our case is

$$\nabla_w \sum_{i=1}^{n} \log(1+\exp(-y_i x_i^T w)) \quad (9)$$

Which is equivalent to

$$\nabla_w(-\log p(Y|X,w)) \quad (10)$$

However, the function in (10) does not have a closed form solution. Therefore, we are going to minimize this function using an iterative approach, gradient descent. This algorithm can be presented as

$$w_{MLE}^{(t+1)} = w_{MLE}^{(t)} - \alpha\nabla_w(-\log p(Y|X,w))|_{w=w_{MLE}^{(t)}}$$

$$= w_{MLE}^{(t)} - \alpha\sum_{i=1}^{n}\frac{1}{\left(1+\exp\left(-y_i x_i^T w_{MLE}^{(t)}\right)\right)}\left(\exp\left(-y_i x_i^T w_{MLE}^{(t)}\right)\right)(-y_i x_i)$$

$$= w_{MLE}^{(t)} + \alpha\sum_{i=1}^{n}(y_i x_i)(1-\sigma\left(y_i x_i^T w_{MLE}^{(t)}\right)) \quad (11)$$

where $w_{MLE}^{(t)}$ indicates the weights result in t[th] iteration, $w_{MLE}^{(t+1)}$ indicates the weights result in (t+1)[th] iteration, and α is the step size of gradient descent.

C. Draw bounding box on blue barrels

After we obtain the optimal weights w^*, we can determine if a pixel is belong to "blue barrels" on test images using this discriminative model as we stated in (4).

To segment the blue barrels from the image, we want to generate a binary mask of blue pixels. Let I be a test image and $I(i,j) \in \mathbb{R}^3$ indicates an RGB pixel value of the pixel in i[th] row and j[th] column. We can generate a mask of detected "blue barrel" region by implementing this:

```
for i from 0 to height of I:
    for j from 0 to width of I:
        if I(i,j)^T w* > 0 :
            mask(i,j) = 1
        else:
            mask(i,j) = 0
```

After we get the mask of the detected "blue barrel" regions, we can easily find the contours of these regions using *findContours* [2] function in opencv. There might be some small noises in our detection. However, on way for us to filter out those small noises is thresholding the contour's area, which can be computed by *contourArea*. Another way to deal with the noises is using dilation and erosion method [3] which is also provided by OpenCV.

Next, we can use *regionprops* [4] to get the bounding box coordinates $(x_{min}, y_{min}, x_{max}, y_{max})$, from those contours and draw the bounding boxes on original test images.

IV. Result

For my implementation, I trained the Logistic Regression model for 10 iterations with step size $\alpha = 0.0001$ and initial weights $w^{(0)} = (0,0,0)^T$.

For each iteration the weights and the error rate are below:

Iteration	Weights	Error rate
1	(-9.25, -6.89, -3.79)	0.0187
2	(-8.19, -3.14, 3.54)	0.0065
3	(-8.09, -6.02, 2.07)	0.0043
4	(-9.33, -2.94, 5.73)	0.001
5	(-9.64, -2.8, 6.6)	0.0028
6	(-10.16-, 3.97, 2.9)	0.0027
7	(-10.59, -3.14, 7.31)	0.0029
8	(-10.99, -3.28, 7.66)	0.003
9	(-11.35, -3.4, 7.98)	0.0031
10	(-11.69, -3.52, 8.28)	0.00316

For the error rate, I hand-label a mask as the ground truth and summing the square of the difference between prediction and ground truth mask.

$$error = \sum_{i=0}^{height}\sum_{j=0}^{width}\frac{(groundtruth(i,j) - prediction(i,j))^2}{width * height}$$

As we can see, the error is the lowest on the 4[th] iteration, so I choose my weights to be $w^* = (-9.33, -2.94, 5.73)^T$

Below is my result for 5 test images.

Test image 1

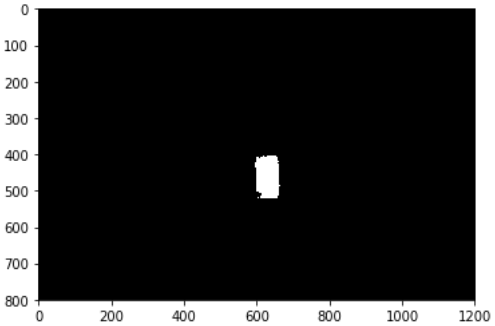

Fig. 1 Mask, test image 1

Fig. 2 Resulting bounding box, test image 1

For test image 1, the coordinates of the bounding box are $(x_{min}, y_{min}, x_{max}, y_{max}) = (592, 404, 669, 528)$, which is what we expected.

Test image 2

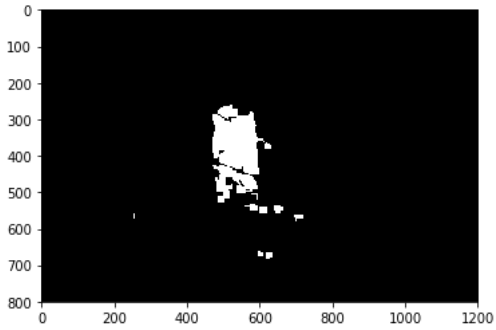

Fig. 3 Mask, test image 2

Fig. 4 Resulting bounding box, test image 2

For test image 2, the coordinates of the bounding box are $(x_{min}, y_{min}, x_{max}, y_{max}) = (466, 260, 615, 531)$. The right side of the bounding box is not exactly lie on blue barrel. I think that is because the blue tubes around the barrel cause some false positives on my detection. As we can see in the Fig. 3, those small white dots are false positives caused by blue tubes, that might be the reason why the bounding box is a little bit off.

Test image 3

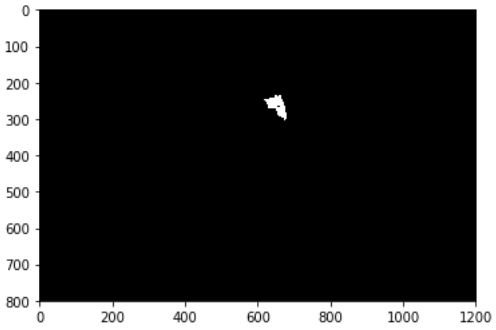

Fig. 5 Mask, test image 3

Fig. 6 Resulting bounding box, test image 3

For test image 3, the coordinates of the bounding box are $(x_{min}, y_{min}, x_{max}, y_{max}) = (608, 226, 681, 306)$. This result is not so good as before, the bounding box is only circling upper half of the blue barrel. I think that is because the lower half of the too dark to be detected by my classifier. Our model is not robust enough to detect the barrel blue in other lighting conditions.

Test image 4

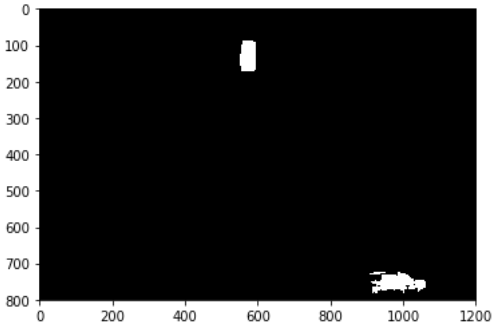

Fig. 7 Mask, test image 4

Fig. 8 Resulting bounding box, test image 4

For test image 4, the coordinates of the bounding box are $(x_{min}, y_{min}, x_{max}, y_{max}) = (548, 86, 597, 176)$. This resulting bounding box is also good. From the mask in the Fig. 7, we can see that our model outputs false positives caused by the blue trash can. However, through some morphological transformation and filtering, we can eliminate those false positives from the final result.

Test image 5

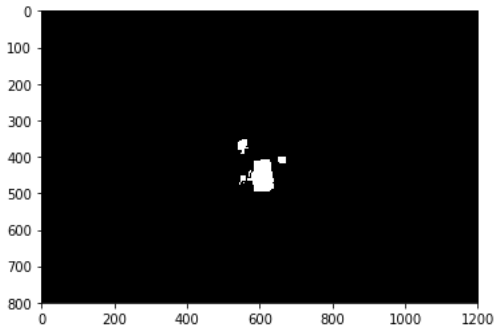

Fig. 9 Mask, test image 5

Fig. 10 Resulting bounding box, test image 5

For test image 5, the coordinates of the bounding box are $(x_{min}, y_{min}, x_{max}, y_{max}) = (539, 352, 571, 397)$, $(562, 407, 641, 497)$. We can see result here is not quite good. In the image, the blue barrel is in front of a blue wall, which makes our classifier generates many false positives just around our target. This indicates that our model is not robust enough to distinguish the blue of the barrels from the blue of other things. In order to get better bounding boxes, we might need some higher-level image processing or analysis other than color segmentation.

CONCLUSION

In this project, we are trying to detect the blue barrels using color segmentation which is implemented by a logistic regression model. After training with provided samples in RBG color space, we can see that our model works well in most of the cases. However, our model is not robust enough to distinguish the barrel blue and not barrel blue, which might be solved using more complex discriminative model.

REFERENCES

[1] Roipoly. Retrived from https://github.com/jdoepfert/roipoly.py

[2] OpenCV, findContours. Retrived from https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

[3] OpenCV, Mophological Transformation. Retrived from https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html

[4] Skimage, regionprops. Retrived from http://scikit-image.org/docs/dev/api/skimage.measure.html#skimage.measure.regionprops