ESE 650 Project 1: Color Segmentation

Rajat Bhageria

January 26, 2018

1 Introduction:

Color segmentation has important applications in computer vision – for example, allowing a self-driving car to identify lines on the highway or a robot to find a soccer ball on a field and then kick it. In this project, our goal was use a probabilistic machine learning approach in order to find the centroid of an image and then estimate how far the barrel is away from the camera (using linear regression to "learn" the focal length). In this project, I modeled the posterior probability of classifying a particular HSV pixel as pixel with a "red barrel" using a single Multinomial Gaussian PDF (as opposed to using Gaussian Mixture Models with the Expectation-Maximization (EM) algorithm. This combined with a sequence of computer vision filters to engineer the barrel classifier created a model that works just as well if not better than one possible through a Gaussian Mixture Model.

2 Approach:

The general approach used was a single Gaussian Multivariate PDF along with bayes rule find the centroid along with the distance to a red barrel in each of the images.
In particular here is the exact results.

2.1 Label all the training images:

For each of the images, use the roipoly python code in order to draw a bounding box around the red barrel. This results in a mask with the barrel in each of the images. Note that I decided that my classifier would

2.2 Train a classifier to identify whether a pixel is a red barrel or not:

It's important to remember that in this method, I chose to use a single Multinomial Gaussian PDF as the estimator for the posterior probability.

1. For each of the 50 training images
   (a) Get the barrel mask that was generated in step 2.1.
   (b) Convert the original RGB image into the HSV color space using OpenCV.
   (c) Use the mask for the respective image it was derived from and append all the HSV pixels inside the barrel itself inside an [nx3] numpy Xtrain matrix and in the corresponding [nx1] Ytrain matrix, store ones indicating that the pixels all correspond with a red barrel.
   (d) Use all the HSV values outside of the barrel itself and append the HSV pixel values to the Xtrain matrix and in the YTrain matrix store zeros indicating that the pixels all correspond to not-red.
2. Do bagging cross validation with 5 CV iterations and each iteration, take 95% of the 54M training pixels and use them to train a the classifier.
   (a) Chose 95% of Xtrain and Ytrain use it for this fold of the CV.

(b) Find the μ_{barrel} ([1x3]) as the average of all the pixels in Xtrain that were labeled as red barrel pixels. Similarly, find the the average of all the pixels in Xtrain that were labeled as other as μ_{other}.

(c) Use *np.cov* in order to find the [3x3] co-variance matrices cov_{barrel} and cov_{other} for all the pixels that were classified as red pixels and other respectively.

(d) Find the priors for both the red barrels and the other pixels. The row-length of Xtrain should be the number of pixels across all the images in the training set (for our 50 images it was 54M pixels). Count the number of ones in Ytrain to find the prior probability of classifying a pixel as a barrel using P("red barrel") = num pixels classified as "red barrel" / total number of red pixels. Similarly, the P("other") = 1-P("red barrel").

3. Average the results from each of the iterations of the CV in order to find the best estimate for μ_{barrel}, μ_{other}, cov_{barrel}, cov_{other}, and both of the priors.

4. Now that the model parameters are trained, save them all as .npy files so they can be used later.

2.3 Train a linear regressor to predict the distance that the barrel is away from the user

Note that this step happens simultaneously to the last section where we're training the classifier for the red barrel.

1. For each of the images

   (a) Parse the name of the image in order to find the actual distance in meters that the red barrel is from the barrel. Append this actual distance to the [mx1] Ytrain matrix for linear regression.

   (b) Use the regionProp Python library on the respective mask for the image (generated from the labeling in step 2.1) in order to find a bounding box around the barrel in the image.

   (c) Find the pixel height of the image by finding by $max_row - min_row$. Append 1/pixel height it to Xtrain [mx1] for the linear regression algorithm. This is because pixel height of the object is inversely related to the distance away from the camera.

   (d) Use an sk-learn linear regressor in order to fit the parameters of the Xtrain and Ytrain data and to learn the focal length of the camera and the actual height of the barrel.

   (e) Now that the regressor is trained, save it so we can access it in the predict method.

2.4 Predict the centroid and distance away of a new image

Given a new image to find the center of the red barrel in and find the distance away from the camera:

1. Convert the image to an HSV to maintain consistency between the fit and predict methods.

2. Create a barrel mask of the new test image using Bayes rule:

   (a) Load all the parameters for the model saved from the fit step.

   (b) For each of the pixels in the image, use the Multinomial Gaussian PDF along with the mu_{barrel}, cov_{barrel} in order to find the $P(xi|"redbarrel")$. In particular, because of underfilling, I took the log Multinomial PDF and thus had to use the function *np.linalg.slogdet* in order to take the log determinant in the normalization term of the PDF. In order to vectorize the code, I used Einstein Summations (*np.einsum*) to quickly calculate the exponent portion of the PDF. For each of the pixels, similarly find P(xi|"others").

   (c) Get the Naive Bayes "score" (numerator of Bayes rule) for each pixel in the image as "barrel" or not. Remember that since we're taking log of everything, our score for "red barrel" will be $log(P(xi|"barrel")) + log(P("barrel))$. Similarly our score for "other" will be $log(P(xi|"other")) + log(P("others))$.

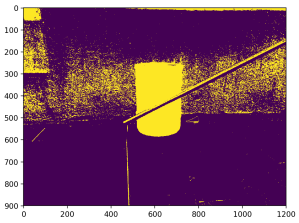

Figure 1: Noise before adding the tuning

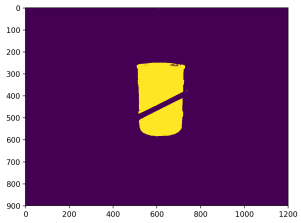

Figure 2: Less noise after adding the tuning

(d) Add some tuning parameters to each of the scores. This was by experimentation and visualization on the training set. After training, it seemed like the score for the "red barrel" was too low since it was classifying too many pixels as "red barrel" (too many false positives); thus I added $900,000$ to it to increase the barrier necessary to be classified as a red barrel pixel. Similarly, I reduced the score for the others by $1,000,000$. This resulted in significantly better results and a lot less noise.

(e) Now classify each pixel in the image, populate that pixel in the mask with 1 if the score for the "red barrel" was higher and 0 if the score for the "other" was higher. This should result in an mask that shows the barrel.

3. Use the OpenCV morphology remove small holes with a min pixel area size of 1,000. Do parameter turning on this value.
4. Use the OpenCV morphology binary erosion to remove noise on the mask
5. Use the OpenCV morphology binary opening to remove noise on the mask
6. Use regionProps on the barrelMask to draw a bbox, find the centroid, and the distance
   (a) Find the bounding box around the main region in the image, the barrel.
   (b) Use the bounding box to predict the distance the barrel is from the camera
      i. Find the pixel height of the bounding box as $max_row - min_row$.
      ii. Load the linear regression model from the fit step
      iii. Predict the distance for the barrel from the linear regression model and return it.
   (c) Use the bounding box to predict the centroid of the barrel. Simply say that the $x_centroid$ is the average of the min_column and max_column of the bbox; and that the $y_centroid$ is the average of the min_row and max_row of the bbox. Return these x and y coordinates.

2.5 Create a testing algorithm

1. Iterate through each of the image in the test folder
2. For each of the images, predict the x, y coordinates of the centroid of the barrel as well as the distance and print.

3 Results

The end result was an algorithm that given a new image that contains a red barrel, will be able to find the center of the barrel, draw the barrel center with a red dot on the image, draw a bounding box around the barrel in the image, and find the distance of the barrel from the photographer with strong accuracy.
Overall the algorithm had extraordinary results: I was able to correctly find the centroid of 46/50 of the training images and have very close distance measurements for all 46 of those images (plus-minus one meter).

Figure 3: Example 1 of correct classification

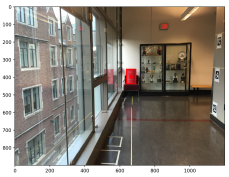

Figure 4: Example 2

3.1 Results for the Test Images:

Here are all the test results for the 10 test images:
ImageNo = [001.png], CentroidX = 853, CentroidY = 440, Distance = 2.2816
ImageNo = [002.png], CentroidX = 619, CentroidY = 417, Distance = 2.6522
ImageNo = [003.png], CentroidX = 642, CentroidY = 438, Distance = 3.0830
ImageNo = [004.png], CentroidX = 630, CentroidY = 444, Distance = 2.5847
ImageNo = [005.png], CentroidX = 645, CentroidY = 466, Distance = 6.9123
ImageNo = [006.png], CentroidX = 633, CentroidY = 427, Distance = 10.5437
ImageNo = [007.png], CentroidX = 0, CentroidY = 0, Distance = 0.0000
ImageNo = [008.png], CentroidX = 741, CentroidY = 475, Distance = 8.8837
ImageNo = [009.png], CentroidX = 671, CentroidY = 429, Distance = 12.5746
ImageNo = [010.png], CentroidX = 664, CentroidY = 395, Distance = 7.5536

4 Discussion

4.1 Things that worked well:

1. Not using GMMs. Interestingly, I was able to accurately find the correct centroid for 46/50 of the training images without using any GMMs and instead using a single Gaussian.

2. Noise reduction. This is arguably where I spent most of my time. Using the erosion and binary opening morphologies along with altering the numerator of the Naive Bayes rules helped reduce the noise and allowed me to use a single Gaussian as opposed to GMMs with quite a bit of accuracy.

3. Speed. The classification algorithm is quite fast and can find the centroid of the barrel, find the distance from the camera, display the image, and display the bounding box of the image within a second. This is very important and I spent quite a lot of time vectorizing the to achieve this speed.

4. Using cross validation. This really allowed me to tune the parameters using the validation set and then ensure that different parts of the training sets are used together to find the average parameters used for prediction.

5. Working well in low-light. The algorithm works quite well in low light mainly because I was using the HSV color space and so saturation was taken into consideration into the model.

4.2 How I tested:

Visual debugging was how I mostly debugged my code. After finding the barrel mask of the image, I would ask matplotlib to display the image. Then I would see how much noise is in the image and if there's too much, I'd add a filter or change tune the parameters for each of the filters or add a constant to the priors for Bayes rule.
I'd additionally ask matplotlib to draw the bounding box as well as the predicted centroid of the barrel and see how close the bounding box was to the real barrel and how close the centroid was to the center of the barrel. If there was some error, I'd go back and fix and reduce noise.

4.3 Problems encountered:

1. Underflow when multiplying by the prior. One of the major problems I faced was no matter what, when I multiplied P("barrel") by P(xi|"barrel"), no matter what, every pixel in the image went to 0. This was because it was under-flowing. And thus, I had to take the log of Bayes rule.

2. Vectorizing the Multinomial Bayes algorithm. Vectorizing is quite important and at one point I was iterating through each of the pixels in the image using a double for loop leading to a $O(n^2)$ algorithm. This was very inefficient and took over 10 mins to analyze a single image. Instead, after using Einstein summation to find the PDF, I was able to do run the model in less than one second.

3. Deciding whether a Gaussian Mixture Model was even necessary. Although I did try the GMM, I decided to stick with my single Gaussian since it was preforming extremely well and had a 46/50 accuracy for the training data. I was using the Orccam's razor of using the simplest model as my mental model.

4. Which color space to use. One of the major decisions I had to make was which color space to use between RGB and HSV (or others). This decision became simple once I was able to visualize the barrel mask and see that no matter which image I used, the barrel mask had significantly less noise for HSV than for RGB and thus I decided to use it.

5. How many colors to classify and whether the simple "barrel" vs "not-barrel" classifier would work. My initial hypothesis was that I'd have to label multiple different color spaces in my algorithm. However, I found that simply labeling a pixel as "red barrel" or "not red barrel" provided quite reasonable and accurate results. And since this worked well, I chose not to go back and relabel and add more colors to the color space.

4.4 Ideas for Further Improvement in the Future

1. Be able to recognize multiple barrels. Currently the algorithm cannot recognize multiple objects. For example, as this picture shows, the algorithm simply assumes that there are two barrels and then places the centroid in the middle. A better solution would of course be two classify for example the bigger barrel, the barrel closer to us, or even classify both.

2. Work more and play with GMMs more. Sure, I was able to correctly find the centroid and distances for 46/50 images. Still, I did try to use GMMs and since my single Gaussian worked so well, I decided to use it over using GMMs.

3. Be able to draw bounding boxes around rotated barrels. This isn't a massive problem since the centroid of the barrel is still correct. However, it is a problem since the diagonal of the barrel has a longer height than the height itself and thus the regressor thinks the barrel is closer to the camera than it actually is.