Classification of Bananas by Ripeness Using Various Methods

By Edward Venator

Introduction

Several of the algorithms in this class can be applied to image processing tasks, and one such task is determination of fruit ripeness. In this report, I examine the results of applying Self-Ordered Mapping and K-Means Clustering to the challenge of classifying bananas as under ripe, ripe, or overripe. These methods, if refined, could be employed in automated fruit packing plants and in other parts of the food industry to automatically sort fruit.

Image Conditioning

The images used in this project were all obtained from the Internet, and contained various background scenery and other extraneous information. Since isolating the banana(s) in the input image is beyond the scope of the project, all of the images were digitally modified to remove everything but the banana(s), replacing those pixels with a white background. In a food inspection environment with a known image background, this could be easily accomplished with trivial image processing algorithms.

Given a set of banana images, one thousand pixels were sampled from each and used as inputs to the various algorithms. Only pixels that were part of the banana were sampled (the white backgrounds were ignored). Random sampling was used to eliminate problems from different image resolutions. In an industrial environment with known camera characteristics, it would be possible to sample a much larger group of pixels, but this would of course slow the network down. In order to increase the size of the training set, images were sampled multiple times, which was possible because the images were much larger than the sample size.

The set of images were sampled one more time to generate a validation set. This set of 19 images (9 ripe, 5 under ripe, and 5 overripe) were used to evaluate the classification abilities of the network.

Self-Ordered Mapping of Random Pixel Samples

My initial attempts to use self-ordered mapping were unsuccessful. Originally, I arranged the red, green, and blue channel pixel values of the sampled pixels into one long vector and fed these vectors into the algorithm. Upon running the algorithm, I found that the entire validation set was classified into one or at most two clusters.

Originally, I thought this was because I was classifying the images using the three color channel values of the pixel vector. Since color is the most important factor in classifying the ripeness of bananas, I decided to use a vector of pixel hue values as the input to the algorithm. However, this also failed, and for the same reason.

Since the pixels were from random locations, there was not necessarily any correlation between the same pixels in each input vector. In order for self-ordered mapping (or k-means clustering) to work in this application, I would need some way to remove pixel location from the system.

Self-Ordered Mapping of Pixel Group Counts

In order to solve this problem, I changed the input vector to the algorithm. After randomly sampling the input images, the pixels were divided into groups by their hue value. The total number of pixels in each group was counted, and the list of counts was used as the input vector for each pattern. So for each image, a vector of pixel counts was created and fed into the self-ordered mapping network.

With this input, the results were much more promising. Using 100 pixel groups and 5 output bins, the algorithm can correctly identify 7 out of 9 ripe bananas, misclassifying 2 as overripe. It correctly classifies 3 out of 5 under ripe bananas, misclassifying 2 as ripe. It correctly classifies 4 out of 5 overripe bananas, and the one it misclassifies has a greenish tint to the image, which caused all tests to classify it as under ripe. In a real situation, controlled lighting and camera calibration would eliminate this problem.

K-Means Clustering of Pixel Bin Counts

I also applied K-Means clustering to this problem, and had similar success. The input to the k-means algorithm was the same set of pixel-group-count vectors used as an input to the self-ordered mapping algorithm in my second attempt. The K-means clustering algorithm was then run for (up to) 1000 iterations (it converged much faster in every test).

With 20 clusters, the k-means algorithm was able to correctly identify 8 out of 9 ripe bananas, misclassifying one as under ripe. It performed worst on under ripe bananas, classifying 3 out of 5 correctly and misclassifying two as ripe. It performed the best on the overripe bananas, correctly classifying 4 out of 5 and only failing on the greenish-tinted banana that the self-ordered mapping algorithm failed on.

Findings

Over the course of this project, I found that both of these classifying algorithms would be suitable for classifying bananas. Even with images from disparate sources with varying resolution and color cast, the results were acceptable. With more tuning and properly calibrated input images, either of these solutions would be suitable for industrial use.

The algorithms run very quickly, but the preprocessing of images (eliminating background pixels, sampling, etc) runs very slowly, because it is written inefficiently. Since that part of the code was not the focus of the project, little development time was applied to making it robust and efficient. If that part of the code were optimized, it would be reasonable for either of these methods to train from an extensive input data set in under a minute and to classify an input in under one second.

Of course, some of the overhead of sampling could be eliminated if the pixel grouping process were unnecessary. Ideally, the solution would be to use a perceptron-based neural network for this classification task, because it would perform the grouping itself as part of the architecture of the network. I had intended to write a perceptron classifier as a third method for comparison, but time did not allow me to complete it.

Instructions for Running Code

Extract all code and input images in this .zip file, and run main.m. To save time, the images have been presampled. To resample images, edit line 10 of main.m to read resample=1. Resampling takes several minutes.