Implementing a Univariate Gaussian Classifier

November 8, 2015

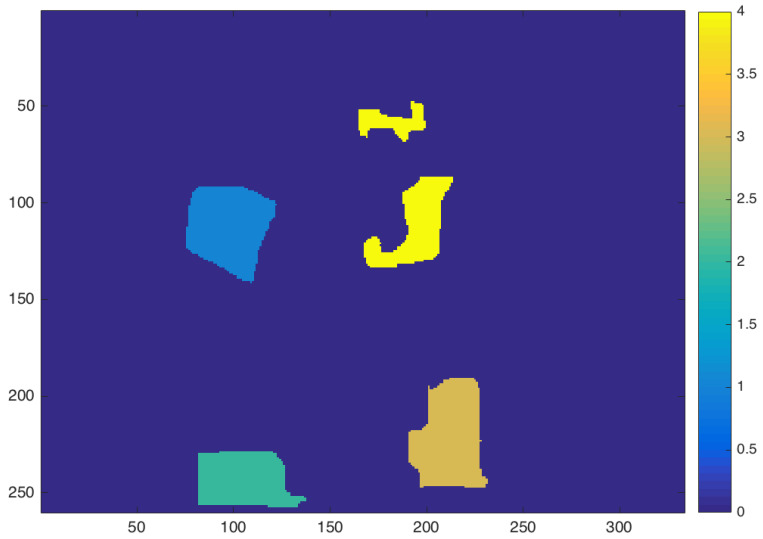

Figure 1: A training mask

0.1 What is classification?

0.2 Steps

1. Train
2. Perform classification
3. Test and estimate error

0.3 Training

0.3.1 What is a mask?

A mask is used to to train the classifier by identifying known regions as belonging to a particular class. In figure1 we can see a typical mask using 4 arbitrary colors plus the background. Each region of color represents a region known to belong to a class. The numeric value that is being visualized as a color is actually just a label. The background of the image, the dark blue sea, is not part of the training data. It is important to remember, the choice of dark blue (0) and the other colors (1-4) are an artifact of Matlab's imagesc function and have no special meaning.

0.3.2 How do we train using a mask?

We are now going to roll through an image and train the classifier.

For an image we are interested in, we apply the training mask and 'learn' how to classify all pixels. Practically speaking, for each image we calculate the μ (mean) and σ^2 (variance) for each class. So if we imagine a small circle in the middle of an image as

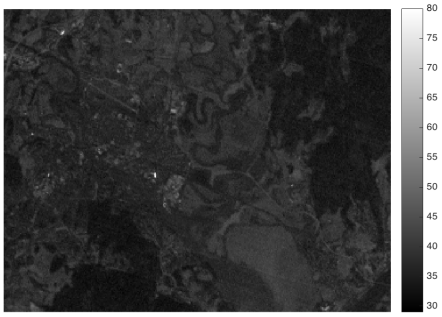

Image 1

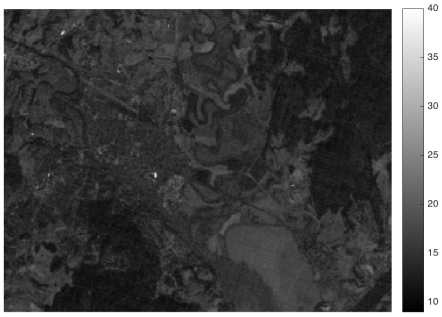

Image 2

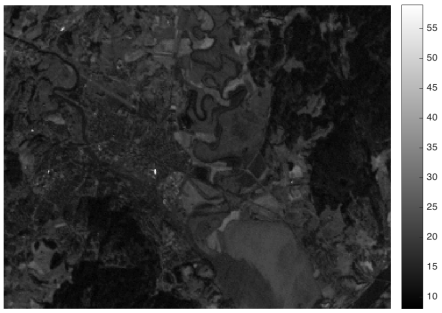

Image 3

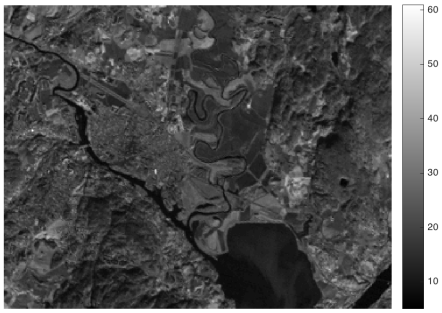

Image 4

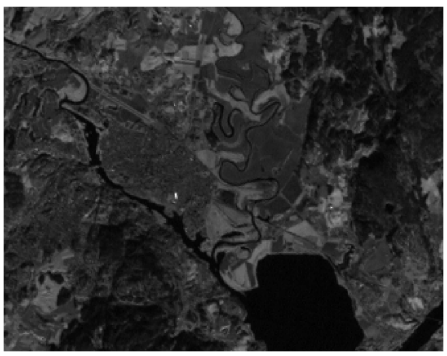

Image 5

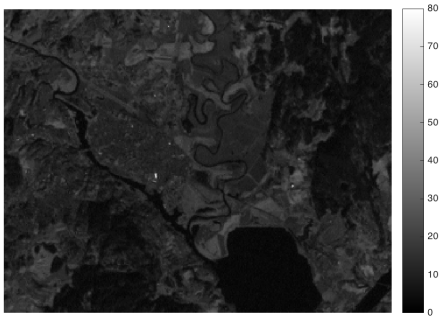

Image 6

Figure 4: Landsat images at different wavelengths

being our training mask for class 1 pixels we would average those pixels and then take the variance of those pixels. We would do this for each class in the mask, which in our example included 1-4, so we end up with 4 pairs of mean and variance.

If you're unsure of how to calculate variance check out Wikipedia's article on Standard

Deviation, it is refreshingly simple.

0.4 Probability Density Function

To use a classifier we need to select a probability density function. The most commonly used probability density function for the purpose of classification is the normal (Gaussian) distribution. We already have μ and σ^2 so the remaining value is x which is the gray level of the pixel.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{1}$$

Visualizing the probability distributions for each of the 4 classes computed on each of the 6 images gives us some intuition into how well the classifier will perform.

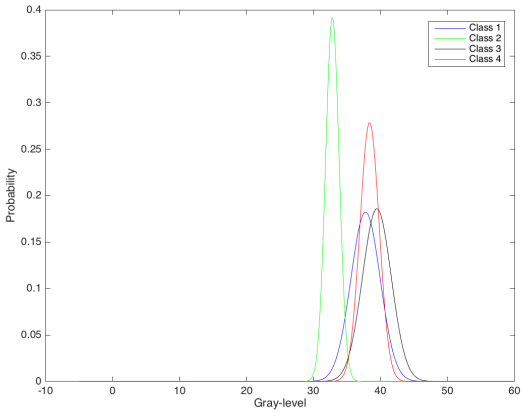

Image 1

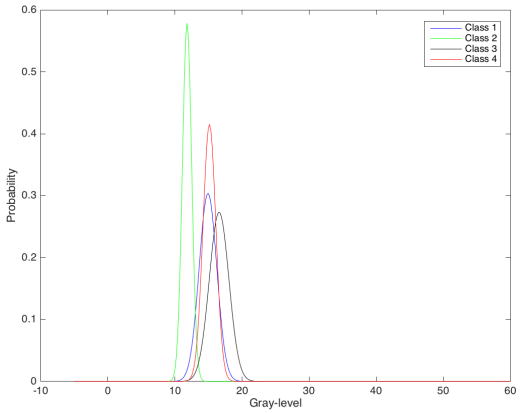

Image 2

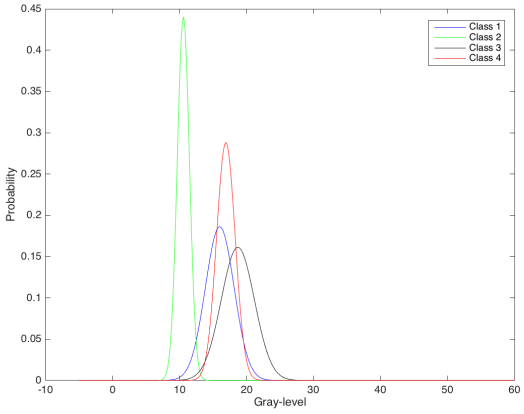

Image 3

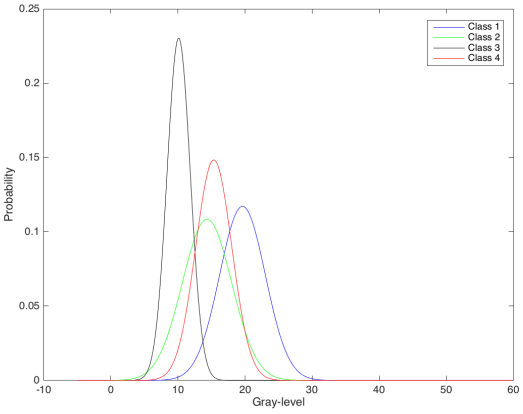

Image 4

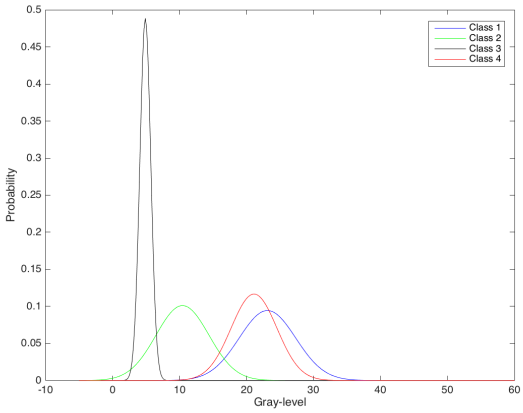

Image 5

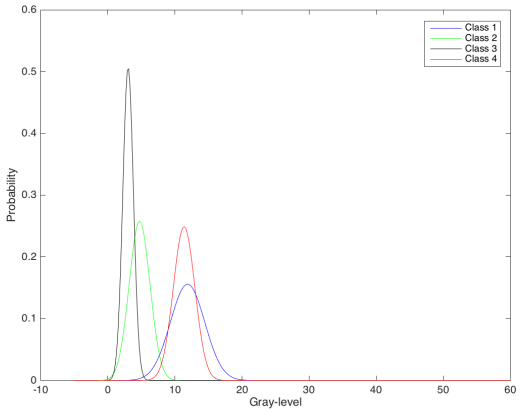

Image 6

Figure 7: Probability distributions for each image

A now for a brief foray into how the Probability Density function is derived, or what is Bayes Theorem and how can it be applied using a normal distribution?

Explain Bayes Theorem and the Gaussian distribution here.

Classifying

So using our probability density function we roll through all the pixels in the image testing the gray level, with each pair of mean and variance. In our case, 4 calculations per pixel. Whichever class yields the highest probability wins. Record the result.

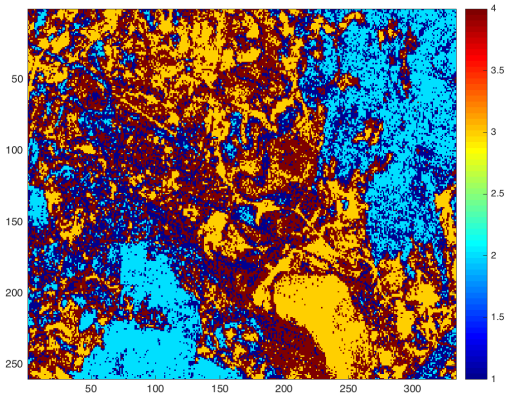

Image 1

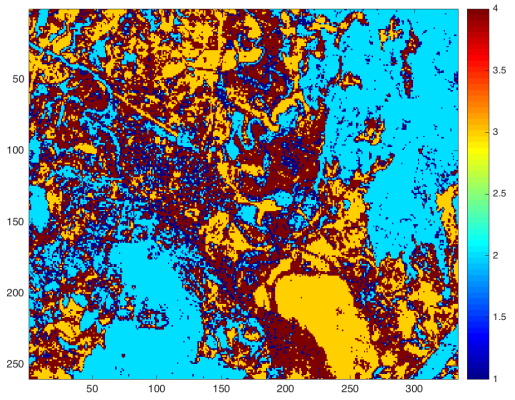

Image 2

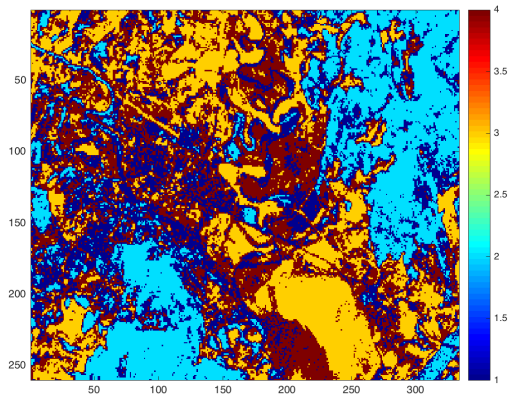

Image 3

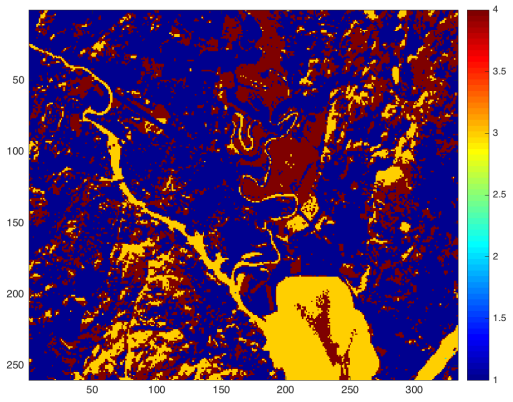

Image 4

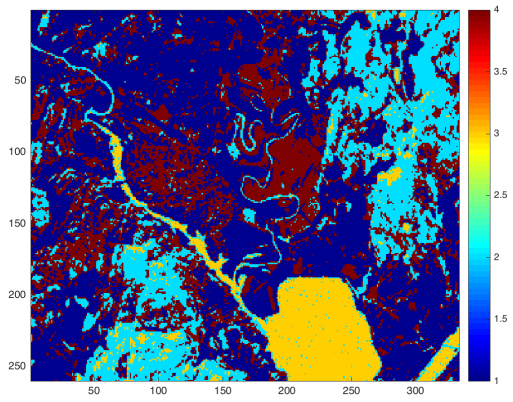

Image 5

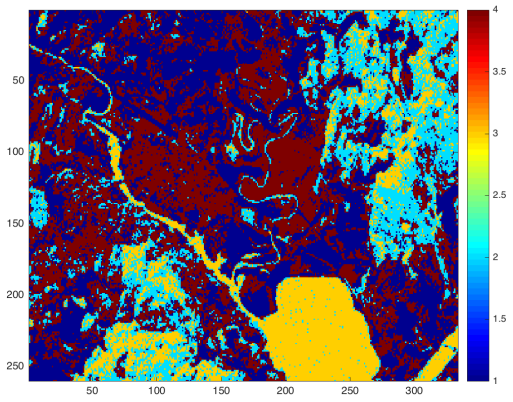

Image 6

Figure 10: Classification results for each image

Testing and measuring error/success

Conclusions

As you can see this is a very simple way of classifying pixels and fails to take advantage of our 6 images and the mask to build a better statistical model for classification. However, it is still pretty cool that it works at all.