## CSSE463: Image Recognition

## Sunset Detector

**Work in pairs on this assignment.**

The overall goal of this project is to implement a system similar to the *baseline* classifier described in my ICME paper (Matthew Boutell, Jiebo Luo, and Robert T. Gray. Sunset scene classification using simulated image recomposition. *IEEE International Conference on Multimedia and Expo*, Baltimore, MD, July 2003; found in the Papers section of Angel and the website). You will write MATLAB code that will extract features from sets of images, train a classifier to classify images using them, and report the results in a short conference-paper quality submission. You will also extend the paper in some novel way.

**Deliverables (SEE THE GRADING RUBRIC FOR MORE DETAILS):**

1. **All MATLAB code**.

2. **Report.** The content of many conference papers follows. I am having you mimic it relatively closely. *This will be good practice for your* *term project and for writing such papers. Other details may be found in the rubric in this folder: look at it.*

*Abstract:* Briefly summarize the entire paper, including background and approximately 1 sentence each for features, classifier, your extension, and **numerical** results.

*Section 1*: Introduction and problem statement. Why do we care about sunset detection (and scene classification as a whole)? Why is sunset detection a challenging problem? Include an image that shows how challenging it is. Why is the proposed solution interesting? This section is mostly non-technical and several paragraphs long.

*Section 2*: What novel work you did to overcome this problem? Describe whatever changes you made to the baseline system to try to improve classification performance, **and the anticipated results**.

*Section 3*: Process followed for feature extraction. Describe the process at a high-level, not merely walking through your code. How did you extract the features? Give lots of details, showing an image with the grid overlaid on it. Give equations for RGB to LST. Describe how you normalized the data, and why your needed to normalize it. What are the theoretical ranges for each band? (It is sufficient to discuss the means, I’ll leave analysis of the standard deviations to a particularly motivated team.) Describe how you handled images with dimensions that weren’t a multiple of the block size, and why that was OK.

*Section 4*: Experimental setup and results. The last section included the details about what you did, regardless of the data set you used. Now include details about the data set: your inputs and outputs. How many images of each type (sunset and non-sunset) did you use in the training and testing sets? Document the experiments conducted. How did you choose your classifier parameters? What was the accuracy of the classifier once the parameters were tuned? How many of the training images became support vectors? Include a table showing evidence of kernel parameter tuning – if it is too lengthy, you can move it to an Appendixand reference it in this section. Once you tune your parameters, you must also show final results in the form of an ROC curve as you vary your decision threshold away from 0.

*Section 5*: Discussion and Conclusion. Show sample images that were classified successfully and unsuccessfully by the system. Include at least 2 of each type (FP, TP, FN, TN) and discuss for each why the classification results seem reasonable. (A good way to choose interesting images is to look at ones in the margin and ones that are far from the margin – use the svmfwd’s y1 output to guide you.) Describe what the next steps would be if you had another 2 weeks to continue working on the problem. What if you had another year? Don’t let this just be an afterthought – consider issues that you identified in the discussion.

**Process and Hints (this is how I would direct a research assistant)**

**Read carefully, this includes other steps you haven’t thought of.**

1. Start by writing your feature extraction code for a single image, that is, calculate the 294-dimension feature vector from the image. You should probably end up encapsulating this as a Matlab function.

The unscaled conversion to LST color space is:

L = R + G + B S = R – B T = R – 2G + B

There is no need to scale them (like dividing L by 3) unless you want to visualize results in this color space. Test your code for bugs (e.g., L should be the grayscale image, and you should have some negatives in S and T). There is sample output on one image in this folder to help you “unit-test” this part of your code. It follows the format below.

1. Write a script that will loop over the whole set of images (a total of N images from a combined training, testing, and difficultTesting sets, I gave you around 1000 images to use), extracting features from each one. Then save the features into a matrix (X). You will also have a vector (Y) of class labels, +1 for sunsets, and -1 for nonsunsets.

Structure of matrix X:

[img1L11, img1L11, img1S11, img1S11, img1T11, img1T11, img1L12, … ;

img2L11, img2L11, img2S11, img2S11, img2T11, img2T11, img2L12, … ;

…

imgNL11, … ];

As an example, imgnLij means the first moment (mean) of the L band of the block in row i, column j of img n, where i = 1:7, j = 1:7, n = 1:N. Each row of the matrix is the 294 features extracted from a single image. L is second moment (standard deviation); actually variance works about the same; that’s what we did in the ICME paper. Note that for a 2D matrix M, std(std(M)) does not compute the correct standard deviation! (try it on [3 3; 1 5]) Instead, convert M to a column vector using the **:** operator first, as in std(M(:)).

Note that some of the S and T values should be negative – if not, maybe you forgot to transform to type double to do math?

1. You will then want to normalize your features, because L, S, and T bands have somewhat different scales, as do the means and variances. (From the formulas, what are they?) The easiest method:

a. Scale each feature type (for example, all the L) so that its max value for any block in the image set = 1 and its min value = 0. To do this for L, you will need to find the max and min of all the L means of all 49 blocks **in all images in all training and testing sets together (don’t normalize each image or set separately)**! [Yes, you should read the ~1000 images from all 4 folders into a ***single*** feature matrix and pass that to the normalize function. Then split it back up when you train and test.] Then the output is out = (orig-min)/(max-min) – you can verify this works. You may use the normalizeFeatures01.m code I gave you to compute this.

Another option, which in theory could work better:

b. Scale so that it’s “zero mean, unit variance”. This is another typical approach: make the mean of that single feature type to be 0 and the standard deviation 1 by first subtracting the mean for the feature type, then dividing by its standard deviation. This is a little less sensitive to outliers, although past students have found that it doesn’t affect performance much.

1. You should save your results in a file so that you don’t have to keep extracting features every time you turn on your machine, or rerun your classifier with a new set of parameters:

save(‘features.mat’, ‘X’, ‘Y’) will do this;

save(‘features.mat’) saves *all* the variables in the workspace.

load(‘features.mat’) will retrieve them back into memory.

1. Train the classifiers, experimenting with kernels and parameters as you see fit. Warning: **this takes time**; feel free to automate this process using a script and looping. <http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf> has a technique worthy of consideration. The svmcv function is also worth investigating. Calculate accuracy on the regular test set (not the difficult images) for each setting so you can check which are best. **Save transcripts of these runs, so you have a record for your report: you need to demonstrate that you tried to tune your parameters.** Don’t expect your accuracy to be the same as the baseline I reported in the paper. Accuracy can vary quite a bit depending on how hard the set of images is. Accuracy could be lower, because one has a smaller training set, and thus haven’t filled the feature space as well, or it could be higher because the sunsets are more “obvious”.

One note: you might wonder how much to vary sigma on your rbf kernel. Intuitively, it shouldn’t ever need to be larger than the maximum distance between any two feature vectors, and the optimal one will likely be far smaller, since the average distance will be smaller. With 0-1 normalized data, the maximum distance should be – can you see why?

You should consider the number of support vectors in your analysis. Find out how many you have – sometimes you can change sigma and drop support vectors while keeping the accuracy fairly constant.

1. Create an ROC curve. What do you vary to get all the points that comprise the curve? Don’t vary the parameters of the SVM.
2. Vary the threshold between what you call a sunset and a non-sunset.   
     
   There are 2 outputs from svmfwd: [y,y1] = svmfwd(…). y is +/1 as you saw in lab and y1 is a real number (distance from margin, pos or negative, depending on which side of the default threshold it's on). This default threshold between classes is 0, but you can move it, say in increments of 0.1.   
     
   Let’s say we classify 4 images, and y1 = [3.2, -1.3, 0.03, -0.01]   
     
   If the threshold (call it t) = 0, then the first and third are detected as sunsets; if -0.1, then the first, third, and fourth, and if +0.1, then only the first would be considered a sunset. (Make sure you understand this.)  
     
   In the general case, assume that strong sunsets have large positive outputs, strong non-sunsets have large negative outputs, and some weak examples of each are near 0 (hard to classify since ambiguous).   
     
   With a high threshold, say t = 5, then all (or nearly all) of the images with output above 5 should be sunsets, so you get few false positives. Thus your false positive rate (FPR) is very low, which is good. However, your true positive rate (TPR, i.e., recall) will be very low as well, since all those true sunsets with output below 5 get called non-sunsets.   
     
   In the other extreme case, t = -5, you get high TPR (good), but a high FPR as well (bad). With t = 0, (the default), you get a balance.   
     
   So once you tune your SVM’s parameters, you run the SVM once and it outputs y1. You then write code to classify the set of images based on a threshold. To get the curve, put that code in a loop, and calculate the true positive rate and a false positive rate for each threshold. Then graph TPR vs. FPR via the ROC plotting code I gave you.
3. Look at some of your successes and failures and include example images of each type in your report. Hypothesize why you got some of them right and some wrong. If a particular result doesn’t make sense, it’s OK to say that too, I’ve come across my share of mystifying images; the point is that you give it some thought.
4. Calculate and report accuracy on the difficult images set and discuss some of the successes and failures on it.
5. Write it up in a report, as described above. Some people, including myself, like to write as they go.
6. *Extensions:* Each team must perform one of the following extensions:
7. Experiment with using the other type of normalization. Does it help, hurt, or neither? [max 7/10 pts]
8. Experiment with using a range of different size grids other than 7. Does another grid size give better results than 7 x7? [max 8/10 pts]
9. Experiment with using overlapping grid blocks with various percentages of overlap. Does overlap give better results? [max 10/10 pts]
10. Use a neural network to classify the data. Does it work better, worse, or the same as an SVM? [max 10/10 pts]
11. Your choice of extension. You can be creative here [max ??/10 pts].

In the spirit of research, you should describe your proposed change in Section 2, then validate the change by providing accuracies or ROC curves for each case. It is easiest to see change if you compare results on the same curve or table.