SUNSET DETECTOR

Owen Trudt – trudtoj@rose-hulman.edu

CSSE463 Image Recognition

18 July 2023

ABSTRACT

When it comes to identifying what a sunset is, the human eye and the human brain can easily determine what is and what is not a sunset. However, computers find this task to be much more difficult, as there can be many different hues of color and objects that can disrupt identifying what is and is not a sunset. To tackle this issue, we have trained a support vector machine to identify images that are sunsets. To do this we break up the image into different sub images and feed the support vector machine the mean and standard deviation of pixel data. To increase the performance of our SVM we optimize the hyper parameters and decide on a threshold that will determine what is classified as a sunset and non-sunset. After these efforts our SVM boasts an accuracy of 90.5%, a true positive rate of 89.6% and a false positive rate of 8.6%.

1. INTRODUCTION

When analyzing images and determining what these images contain, scene classification plays a very important role. Typically, if a scene contains straight edges like buildings that can be slightly easier to classify, but when we begin to analyze images of natural scenes like sunsets that is where image recognition can become more complicated. Sunsets can contain many different colors and many different natural objects, so there can be many irregularities that are not standard across different images. Figure 0 depicts how two images that are sunsets can be very different.

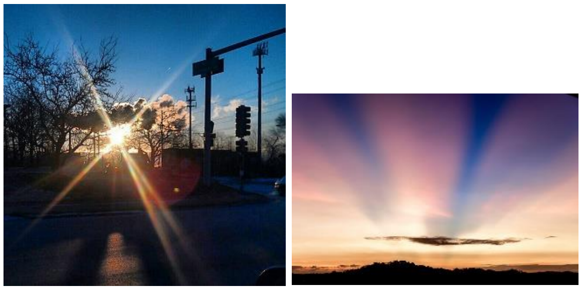

Figure 0: *Two sunset images that are incredibly different.*

The benefits a solution to this problem can be vast, for example if a home wants to have windows that become tinted when the sun starts to set, this solution to identifying sunsets can be very useful.

There are many different approaches to this problem. Like our last project we could tune color thresholds to emcompass the colors we would expect to be included in sunsets. And to handle the objects we could implement some sort of edge detection. However this approach would be extremely time intensive and difficult to gather accurate results.

What we decided to do is to determine different features in the images and train a support vector machine on these features. Which will only take a fraction of the time it would take to manually tune threshold values.

The features we compute from the images are then fed into a SVM, these features then train the SVM. We then use another set of features to tune the SVM. Then we finally use a test set of features to test the final classifier. Overall this process is more computationally expensive, but the results prove that the computational nature is worth it.

2. PROCESS

2.1 Image Segmentation

We decided it would be best to segment our image into smaller sub images. If we tried to compute features on the image a lot of the data would be lost, and important details would be missed. So, to counter this, we decided to segment our image into a 7x7 grid, creating 49 sub-images per 1 image. These sub-images do include the edges of the original image, causing some irregularities due to rounding, however, this does not affect the data we collected from the images. [1]

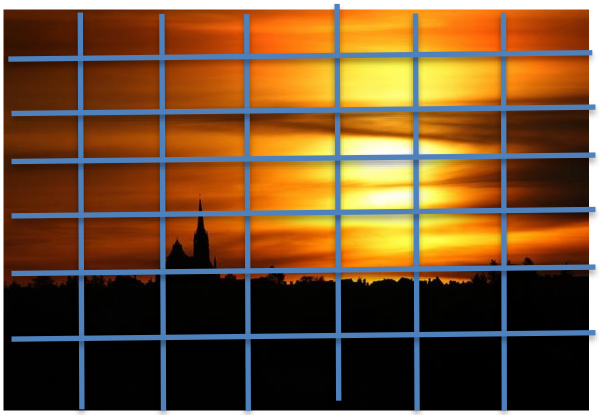

Figure 1: *An example of an image containing 49 sub-images.*

2.2 Color Conversion to LST

Our images were currently in the RGB color space, however, we determined to use the LST color space due to advice given to us from our professor. [1]

$$L = R + G + B$$

$$S = R - B$$

$$T = R - 2*G + B$$

Once we calculated these values, we determined the mean and the standard deviation within each sub-image. Following that step we normalized the data to minimize the possibility of any piece of data being weighed heavier than the others.

2.2 Feature Calculation

To calculate our features, we extract the mean and standard deviation of the L, S, and T values. Overall, this provides us with 294 features to classify our images with (7*7*6).

3. CLASSIFICATION

3.1 Support Vector Machines

We used support vector machines to determine if an image is a sunset or not. Support vector machines are fed data and based on that data they create boundaries to separate these different data points. Therefore, classifying the data into different categories. To optimize performance, we can augment some hyper parameters. In line with this we can change the kernel function to be linear, polynomial and gaussian. The gaussian kernel typically performs the best and has two parameters that we can change, the box constraint and the kernel width. The box constraint determines the cost of a misclassification, and the kernel width determines the curvature of the boundary. Ultimately, we would want to optimize these parameters to get the best performance without overfitting our data.

4. IMAGE SETS

In total we used 3200 images, divided into 3 sets.

Table 1: *This table contains the image splits for each image set.*

	Sunset	Non-Sunset
Training	800	800
Validation	300	300
Test	500	500

Each image in these sets is extremely unique. They range from different colors to different types of sunsets, different intensities of sunsets and of course images that are not sunsets. For each one of these images, we calculated 294 features. These features are then input into our support vector machine. We used the training set of images to create our support vector machine. The validation set was then used to tune the hyper parameters. And finally, the test set was used to calculate the accuracy of our support vector machine.

5. RESULTS

5.1 SVM Training

When we extracted our features, we found that 2 images were missing from our initial calculation of 3200 images, to account for this we had to alter the number of images we passed to our SVM. So instead of giving 1600 images to train our SVM, we passed in the features of 1599 images: a 1599x294 array. We used the fitcsvm function with the standardize parameter set to true so our data would be weighted the same. Due to previous practice with SVMs we decided to use the gaussian (rbf) kernel to fit our data.

5.2 Hyperparameter Optimization

To optimize our hyperparameters we used the validation set of images: a 599x294 array of features. Varying our hyperparameters was a challenge at first but then we quickly thought of the idea to use a double for loop and have one loop calculate the box parameter and the other for loop calculate the kernel width using the power function provided by MATLAB. We then used the predict function to calculate a net using these hyperparameters. The results of these calculations can be seen in Appendix A. After these calculations we decided to use the box constraint of 16 and the kernel width of 16 as this boasted an accuracy of 93% (0.929), a true positive rate of 95% (0.9567) and a false positive rate of 9% (0.0929) while having only 758 support vectors. Which is very important as this shows us that our SVM is not overfitting the data and based on the calculated metrics it is not underfitting our data either. I am sure with further experimentation we can increase our accuracy, true positive rate, and false positive rate.

5.3 Deciding the Threshold

Once we decided what values of hyperparameters would work we needed to decide what threshold we would choose to maximize our true positive rate and minimize our false positive rate. This step is where we used our test set of images: 1000x294 array of features. To calculate the threshold value that would maximize our TPR and minimize our FPR we calculated which point (FPR, TPR) had the smallest distance from (0,1). The threshold value was -0.08.

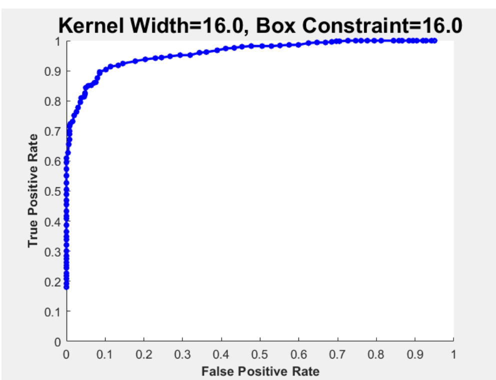

Figure 2: *The ROC curve of our SVM using the shown constraints.*

The data calculated from the thresholds can be seen in Appendix B.

The threshold value of the -0.08 resulted in an accuracy rating of 90.5% (0.905), a true positive rating of 89.5% (0.8958) and a false positive rating 8.5% (0.0858).

6. DISCUSSION

6.1 Performance

The SVM performed relatively well with a true positive rating of 89.5% and a false positive rating of 8.5%. Ultimately the goal would be to get our true positive rating into the upper nineties and lower our false positive rate to be as close to zero as we can. Let's look at a couple of the images our SVM classified.

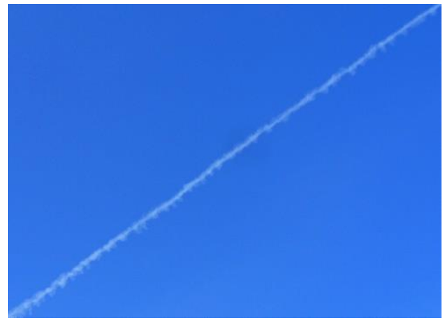

Figure 3: *This image is a non-sunset that was a false positive with a score of -1.020.*

Figure 4: *This image is a sunset that was a true positive with a score of 1.5.*

Figure 4 was an image correctly classified as a sunset, while figure 3 was an image incorrectly classified as a sunset. Figure 3 being misclassified is perplexing and honestly, I am not exactly sure what detail about this image caused confusion for my SVM. However, figure 4 is correctly classified and is quite obviously a sunset showing us that our SVM does work correctly.

Figure 5: *This image is a sunset that was a true positive with a score of 0.936.*

Figure 6: *This image is a sunset that was a false negative with a score of -0.0389.*

Figure 5 was a sunset that was classified correctly, this makes sense as the image is dominated by orange colors. And as most sunsets have hues of orange, the SVM might have found that the color to be weighted heavier. Figure 6, however, is also dominated by orange but was classified as a non-sunset, resulting in a false negative. I believe this could be due to the nature of the scene and the colors being obstructed by the trees in the image.

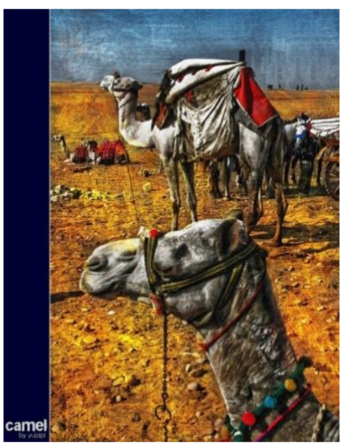

Figure 7: *This image is a non-sunset and was a true negative with a score of -0.277.*

Figure 8: *This image is a non-sunset and was a true negative with a score of -0.42.*

Figure 7 depicts a drawing of some camels and colors not typical of a sunset and our SVM classifies it as such. Figure 8 is an image of a flower that does carry some hues that could be seen in a sunset, but it is correctly classified as a non-sunset.

Figure 9: *This image is a sunset that was a false negative with a score of -0.2053.*

Figure 10: *This image is a non-sunset that was a false positive with a score of -0.03.*

Figure 9 depicts a sunset image that was classified as a false negative, I believe this to be similar to figure 6, as the scene caused the SVM confusion, the skyline of the city and the cityscape caused were the possible cause of this confusion. Figure 10 depicts a non-sunset that was a false positive, I believe this to be because there are lots of hues of orange, and based on past reasoning, the SVM may weigh orange more than other colors, causing this image to be falsely classified.

7. CONCLUSION

Overall, our SVM is up to the task of separating images of sunsets and images of non-sunsets. We saw how our SVM possibly weighed orange more than the other colors. This would make sense as our SVM is trained on the mean and standard deviation of colors in our image. To further our accuracy, we would like to implement a form of edge detection to help us separate parts of the image that are objects and skew the data we are collecting. Also, a larger set of data would be beneficial as this would help our SVM create better boundaries. As the next part of our project includes using a CNN (convolutional neural network), this implementation could hold performance benefits.

8. REFERENCES

[1] M. Boutell, J. Luo, and R. T. Gray, “Sunset scene classification using simulated image recomposition,” *2003 International Conference on Multimedia and Expo. ICME ’03.*, 2003.

APPENDIX

APPENDIX A:

Kernel	Box	TPR	FPR	Accuracy	Support Vectors
2	128	1	1	0.500834724540902	1592
2	256	1	1	0.500834724540902	1592
2	512	1	1	0.500834724540902	1592
2	1024	1	1	0.500834724540902	1592
4	2	0.990000000000000	0.852842809364549	0.569282136894825	1588
4	4	0.990000000000000	0.852842809364549	0.569282136894825	1588
4	8	0.990000000000000	0.852842809364549	0.569282136894825	1588
4	16	0.990000000000000	0.852842809364549	0.569282136894825	1588
4	32	0.990000000000000	0.852842809364549	0.569282136894825	1588
4	64	0.990000000000000	0.852842809364549	0.569282136894825	1588
4	128	0.990000000000000	0.852842809364549	0.569282136894825	1588
4	256	0.990000000000000	0.852842809364549	0.569282136894825	1588
4	512	0.990000000000000	0.852842809364549	0.569282136894825	1588
4	1024	0.990000000000000	0.852842809364549	0.569282136894825	1588
8	2	0.970000000000000	0.274247491638796	0.848080133555927	1380
8	4	0.970000000000000	0.280936454849498	0.844741235392321	1387
8	8	0.970000000000000	0.280936454849498	0.844741235392321	1387
8	16	0.970000000000000	0.280936454849498	0.844741235392321	1387
8	32	0.970000000000000	0.280936454849498	0.844741235392321	1387
8	64	0.970000000000000	0.280936454849498	0.844741235392321	1387
8	128	0.970000000000000	0.280936454849498	0.844741235392321	1387
8	256	0.970000000000000	0.280936454849498	0.844741235392321	1387
8	512	0.970000000000000	0.280936454849498	0.844741235392321	1387
8	1024	0.970000000000000	0.280936454849498	0.844741235392321	1387
16	2	0.943333333333333	0.0836120401337793	0.929883138564274	762
16	4	0.940000000000000	0.0869565217391304	0.926544240400668	764
16	8	0.950000000000000	0.0936454849498328	0.928213689482471	774

16	16	0.956666666666667	0.0969899665551840	0.929883138564274	758
16	32	0.953333333333333	0.0969899665551840	0.928213689482471	754
16	64	0.953333333333333	0.0969899665551840	0.928213689482471	754
16	128	0.953333333333333	0.0969899665551840	0.928213689482471	754
16	256	0.953333333333333	0.0969899665551840	0.928213689482471	754
16	512	0.953333333333333	0.0969899665551840	0.928213689482471	754
16	1024	0.953333333333333	0.0969899665551840	0.928213689482471	754
32	2	0.910000000000000	0.0936454849498328	0.908180300500835	612
32	4	0.923333333333333	0.0769230769230769	0.923205342237062	584
32	8	0.920000000000000	0.0802675585284281	0.919866444073456	572
32	16	0.923333333333333	0.0836120401337793	0.919866444073456	547
32	32	0.916666666666667	0.0869565217391304	0.914858096828047	542
32	64	0.916666666666667	0.0869565217391304	0.914858096828047	527
32	128	0.926666666666667	0.0836120401337793	0.921535893155259	507
32	256	0.923333333333333	0.0769230769230769	0.923205342237062	502
32	512	0.923333333333333	0.0769230769230769	0.923205342237062	502
32	1024	0.923333333333333	0.0769230769230769	0.923205342237062	502
64	2	0.883333333333333	0.0936454849498328	0.894824707846411	699
64	4	0.866666666666667	0.0869565217391304	0.889816360601002	637
64	8	0.866666666666667	0.0936454849498328	0.886477462437396	593
64	16	0.890000000000000	0.103678929765886	0.893155258764608	554
64	32	0.896666666666667	0.0903010033444816	0.903171953255426	531
64	64	0.896666666666667	0.0903010033444816	0.903171953255426	526
64	128	0.896666666666667	0.0936454849498328	0.901502504173623	508
64	256	0.896666666666667	0.100334448160535	0.898163606010017	495
64	512	0.880000000000000	0.100334448160535	0.889816360601002	475
64	1024	0.883333333333333	0.103678929765886	0.889816360601002	458
128	2	0.833333333333333	0.0802675585284281	0.876460767946578	882
128	4	0.856666666666667	0.0836120401337793	0.886477462437396	778
128	8	0.863333333333333	0.0936454849498328	0.884808013355593	695
128	16	0.850000000000000	0.0903010033444816	0.879799666110184	640
128	32	0.850000000000000	0.100334448160535	0.874791318864775	594

128	64	0.856666666666667	0.103678929765886	0.876460767946578	556
128	128	0.860000000000000	0.103678929765886	0.878130217028381	532
128	256	0.870000000000000	0.103678929765886	0.883138564273790	515
128	512	0.883333333333333	0.110367892976589	0.886477462437396	498
128	1024	0.866666666666667	0.107023411371237	0.879799666110184	479
256	2	0.800000000000000	0.0735785953177258	0.863105175292154	1186
256	4	0.813333333333333	0.0802675585284281	0.866444073455760	1017
256	8	0.823333333333333	0.0769230769230769	0.873121869782972	881
256	16	0.843333333333333	0.0869565217391304	0.878130217028381	780
256	32	0.856666666666667	0.0969899665551840	0.879799666110184	695
256	64	0.846666666666667	0.0969899665551840	0.874791318864775	635
256	128	0.843333333333333	0.0936454849498328	0.874791318864775	592
256	256	0.840000000000000	0.107023411371237	0.866444073455760	560
256	512	0.850000000000000	0.100334448160535	0.874791318864775	537
256	1024	0.836666666666667	0.117056856187291	0.859766277128548	509
512	2	0.650000000000000	0.0501672240802676	0.799666110183639	1542
512	4	0.760000000000000	0.0635451505016722	0.848080133555927	1373
512	8	0.800000000000000	0.0735785953177258	0.863105175292154	1185
512	16	0.810000000000000	0.0802675585284281	0.864774624373957	1016
512	32	0.826666666666667	0.0769230769230769	0.874791318864775	879
512	64	0.846666666666667	0.0869565217391304	0.879799666110184	777
512	128	0.853333333333333	0.0936454849498328	0.879799666110184	696
512	256	0.843333333333333	0.0969899665551840	0.873121869782972	638
512	512	0.836666666666667	0.0936454849498328	0.871452420701169	590
512	1024	0.836666666666667	0.107023411371237	0.864774624373957	558
1024	2	0.00666666666666667	0	0.502504173622705	1598
1024	4	0.273333333333333	0.00668896321070234	0.632721202003339	1596
1024	8	0.650000000000000	0.0501672240802676	0.799666110183639	1540
1024	16	0.760000000000000	0.0635451505016722	0.848080133555927	1373
1024	32	0.800000000000000	0.0735785953177258	0.863105175292154	1184
1024	64	0.810000000000000	0.0802675585284281	0.864774624373957	1016
1024	128	0.826666666666667	0.0769230769230769	0.874791318864775	879

1024	256	0.846666666666667	0.0869565217391304	0.879799666110184	777
1024	512	0.853333333333333	0.0936454849498328	0.879799666110184	695
1024	1024	0.843333333333333	0.0969899665551840	0.873121869782972	636

APPENDIX B:

Threshold	**TPR**	**FPR**	**Accuracy**
-2	1	0.950099800399202	0.524000000000000
-1.96000000000000	1	0.950099800399202	0.524000000000000
-1.92000000000000	1	0.942115768463074	0.528000000000000
-1.88000000000000	1	0.942115768463074	0.528000000000000
-1.84000000000000	1	0.928143712574850	0.535000000000000
-1.80000000000000	1	0.920159680638723	0.539000000000000
-1.76000000000000	1	0.904191616766467	0.547000000000000
-1.72000000000000	1	0.894211576846307	0.552000000000000
-1.68000000000000	1	0.884231536926148	0.557000000000000
-1.64000000000000	1	0.866267465069860	0.566000000000000
-1.60000000000000	1	0.858283433133733	0.570000000000000
-1.56000000000000	1	0.858283433133733	0.570000000000000
-1.52000000000000	1	0.844311377245509	0.577000000000000
-1.48000000000000	1	0.812375249500998	0.593000000000000
-1.44000000000000	1	0.794411177644711	0.602000000000000
-1.40000000000000	1	0.776447105788423	0.611000000000000
-1.36000000000000	1	0.760479041916168	0.619000000000000
-1.32000000000000	1	0.746506986027944	0.626000000000000
-1.28000000000000	1	0.726546906187625	0.636000000000000
-1.24000000000000	0.997995991983968	0.704590818363273	0.646000000000000
-1.20000000000000	0.997995991983968	0.696606786427146	0.650000000000000
-1.16000000000000	0.995991983967936	0.684630738522954	0.655000000000000
-1.12000000000000	0.993987975951904	0.668662674650699	0.662000000000000
-1.08000000000000	0.993987975951904	0.646706586826347	0.673000000000000
-1.04000000000000	0.991983967935872	0.624750499001996	0.683000000000000

-1	0.985971943887776	0.598802395209581	0.693000000000000
-0.960000000000000	0.985971943887776	0.574850299401198	0.705000000000000
-0.920000000000000	0.983967935871744	0.550898203592814	0.716000000000000
-0.880000000000000	0.981963927855711	0.528942115768463	0.726000000000000
-0.840000000000000	0.981963927855711	0.504990019960080	0.738000000000000
-0.800000000000000	0.981963927855711	0.477045908183633	0.752000000000000
-0.760000000000000	0.979959919839679	0.453093812375250	0.763000000000000
-0.720000000000000	0.975951903807615	0.433133732534930	0.771000000000000
-0.680000000000000	0.973947895791583	0.411177644710579	0.781000000000000
-0.640000000000000	0.967935871743487	0.389221556886228	0.789000000000000
-0.600000000000000	0.961923847695391	0.361277445109780	0.800000000000000
-0.560000000000000	0.959919839679359	0.343313373253493	0.808000000000000
-0.520000000000000	0.951903807615230	0.319361277445110	0.816000000000000
-0.480000000000000	0.951903807615230	0.293413173652695	0.829000000000000
-0.440000000000000	0.947895791583166	0.267465069860279	0.840000000000000
-0.400000000000000	0.943887775551102	0.245508982035928	0.849000000000000
-0.360000000000000	0.941883767535070	0.229540918163673	0.856000000000000
-0.320000000000000	0.937875751503006	0.203592814371257	0.867000000000000
-0.280000000000000	0.931863727454910	0.177644710578842	0.877000000000000
-0.240000000000000	0.923847695390782	0.145708582834331	0.889000000000000
-0.200000000000000	0.917835671342685	0.133732534930140	0.892000000000000
-0.160000000000000	0.913827655310621	0.113772455089820	0.900000000000000
-0.120000000000000	0.903807615230461	0.101796407185629	0.901000000000000
-0.0800000000000000	0.895791583166333	0.0858283433133733	0.905000000000000
-0.0400000000000000	0.891783567134269	0.0858283433133733	0.903000000000000
0	0.875751503006012	0.0798403193612775	0.898000000000000
0.0400000000000000	0.861723446893788	0.0758483033932136	0.893000000000000
0.0800000000000000	0.859719438877756	0.0718562874251497	0.894000000000000
0.120000000000000	0.851703406813627	0.0638722554890220	0.894000000000000
0.160000000000000	0.849699398797595	0.0558882235528942	0.897000000000000
0.200000000000000	0.843687374749499	0.0499001996007984	0.897000000000000
0.240000000000000	0.825651302605211	0.0479041916167665	0.889000000000000

0.280000000000000	0.821643286573146	0.0479041916167665	0.887000000000000
0.320000000000000	0.813627254509018	0.0459081836327345	0.884000000000000
0.360000000000000	0.809619238476954	0.0379241516966068	0.886000000000000
0.400000000000000	0.795591182364730	0.0359281437125749	0.880000000000000
0.440000000000000	0.777555110220441	0.0299401197604790	0.874000000000000
0.480000000000000	0.763527054108216	0.0259481037924152	0.869000000000000
0.520000000000000	0.751503006012024	0.0199600798403194	0.866000000000000
0.560000000000000	0.731462925851703	0.0159680638722555	0.858000000000000
0.600000000000000	0.723446893787575	0.00998003992015968	0.857000000000000
0.640000000000000	0.715430861723447	0.00798403193612774	0.854000000000000
0.680000000000000	0.699398797595190	0.00798403193612774	0.846000000000000
0.720000000000000	0.689378757515030	0.00798403193612774	0.841000000000000
0.760000000000000	0.671342685370742	0.00798403193612774	0.832000000000000
0.800000000000000	0.655310621242485	0.00598802395209581	0.825000000000000
0.840000000000000	0.627254509018036	0.00399201596806387	0.812000000000000
0.880000000000000	0.609218436873748	0	0.805000000000000
0.920000000000000	0.595190380761523	0	0.798000000000000
0.960000000000000	0.573146292585170	0	0.787000000000000
1	0.551102204408818	0	0.776000000000000
1.04000000000000	0.527054108216433	0	0.764000000000000
1.08000000000000	0.505010020040080	0	0.753000000000000
1.12000000000000	0.488977955911824	0	0.745000000000000
1.16000000000000	0.468937875751503	0	0.735000000000000
1.20000000000000	0.454909819639279	0	0.728000000000000
1.24000000000000	0.432865731462926	0	0.717000000000000
1.28000000000000	0.416833667334669	0	0.709000000000000
1.32000000000000	0.408817635270541	0	0.705000000000000
1.36000000000000	0.386773547094188	0	0.694000000000000
1.40000000000000	0.364729458917836	0	0.683000000000000
1.44000000000000	0.348697394789579	0	0.675000000000000
1.48000000000000	0.338677354709419	0	0.670000000000000
1.52000000000000	0.320641282565130	0	0.661000000000000

1.56000000000000	0.300601202404810	0	0.651000000000000
1.60000000000000	0.284569138276553	0	0.643000000000000
1.64000000000000	0.274549098196393	0	0.638000000000000
1.68000000000000	0.274549098196393	0	0.638000000000000
1.72000000000000	0.262525050100200	0	0.632000000000000
1.76000000000000	0.252505010020040	0	0.627000000000000
1.80000000000000	0.244488977955912	0	0.623000000000000
1.84000000000000	0.226452905811623	0	0.614000000000000
1.88000000000000	0.218436873747495	0	0.610000000000000
1.92000000000000	0.208416833667335	0	0.605000000000000
1.96000000000000	0.192384769539078	0	0.597000000000000
2	0.180360721442886	0	0.591000000000000