A numerical, visual method for generating the (non-)linear separating isosurface of two classes of objects in two-dimensional space using Marching Squares and OpenCV

Shawn Halayka*

Monday 2nd May, 2022 18:41

Abstract

With regard to the separating isosurface of two classes of objects in two-dimensional space, the transition from rectilinear to curvilinear is documented. Marching Squares and OpenCV were used to calculate the data that were used for illustrations in this paper. It is found that the curvilinearity is adjustable via variable grid resolution, as well as via convolution such as Gaussian blur.

1 Introduction

In terms of statistical learning, a balance between high variance (curvilinearity) and high bias (rectilinearity) is to be had [1]. However, it is not clear from [1] how to go about obtaining a nonlinear, radial isosurface without having to rely on the Support Vector Machine (SVM) functionality in the popular R library e1071. Thus, it can be daunting to those wishing to perform such a thing from the ground up in other languages, such as C++ or Python.

The goal of this paper is three-fold: 1) to pay homage to [1], and 2) to have a little fun, as well as 3) to show that obtaining such an isosurface is a simple matter, especially so when there are only two classes and they are in only two-dimensional space.

The images passed into the Marching Squares algorithm give our technique an inherent visual property. Simply put: Marching Squares [2–4] works as a radial (spherical) isosurface generator, as well as a rectilinear (planar) isosurface generator – it all depends on the grid resolution. The data generated by Marching Squares are the line segments (sets of contours) that separate the two classes, as well as triangles for the area of each class. Classifying a test point is a matter of ray-triangle intersection, which can be accelerated using a Bounding Volume Hierarchy or quadtree data structure.

OpenCV [5] was used to downsize images, as well as blur images. OpenCV could also have been used to sharpen images, etc. There are many convolutions to experiment with.

Please see [2] for a small introductory Marching Squares code in C++.

*Independent – sjhalayka@gmail.com

2 Regarding the illustrations

Fig. 1 shows a 2x2 pixel bitmap image. This image's pixel values are used as input to the Marching Squares algorithm. The output of the Marching Squares algorithm is shown in Fig. 2 – a rectilinear separation is produced.

Figs. 3 and 4 show the same points, but with a 16x16 pixel image – a rough, curvilinear, radial separation is produced.

Figs. 5 and 6 show the same points, but with a 64x64 pixel image – a smooth, curvilinear, radial separation is produced.

Figs. 7 and 8 show the contour sets, in transparent grey, generated by batches and batches of training data. The 2x2 pixel images used to generate these contour sets are then averaged, and the so-called average contour is shown in black. The result is a rectilinear separation.

Figs. 9 and 10 show the contour sets, in transparent grey, generated by batches and batches of training data. The 64x64 pixel images used to generate these contour sets are then averaged, and the so-called average contour is shown in black. The result is a smooth, curvilinear, radial separation.

Figs. 11 and 12 use the same 64x64 pixel image, except that it's been blurred once. It is apparent that this removes the higher-resolution detail from the isosurface.

Figs. 13 and 14 use the same 64x64 pixel image, except that it's been blurred ten times. It is apparent that this further removes the higher-resolution detail from the isosurface.

As these illustrations show, the amount of curvilinearity (and thus variance versus bias) is easily adjustable using simple computer vision techniques.

3 Conclusion

It is apparent that [1] is a treasure trove of statistical learning techniques, although it does not give many details when it comes to generating a radial isosurface outside of the realm of the SVM. Here we have illustrated how Marching Squares and OpenCV can be used to generate such a radial isosurface. This numerical, visual method of generating the (non-)linear separating isosurface of two classes of objects in two-dimensional space is easy to implement, and is as far as we know, novel.

This paper's reference section is rather small, and the hunt for the literature is left to those who have access to the journals.

References

[1] James, et al. An Introduction to Statistical Learning with Applications in R. ISBN: 978-1-0716-1417-4

[2] Halayka. Marching Squares C++ implementation.
`https://github.com/sjhalayka/Marching-Squares`

[3] Gong and Newman. A corner feature sensitive marching squares. Southeastcon 2013 Proceedings of IEEE, pp. 1-6, 2013

[4] Bobrowski. Estimation of systematic errors of discrete line approximation by triangular tessellation and marching squares algorithm. Micron, vol. 141, pp. 102966, 2021

[5] OpenCV library.
`https://opencv.org/`

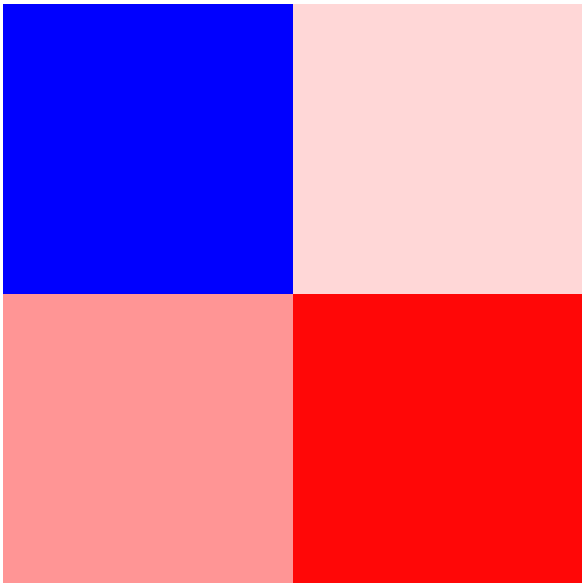

Figure 1: Bitmap image used as input to the Marching Squares algorithm. Image size is 2x2.

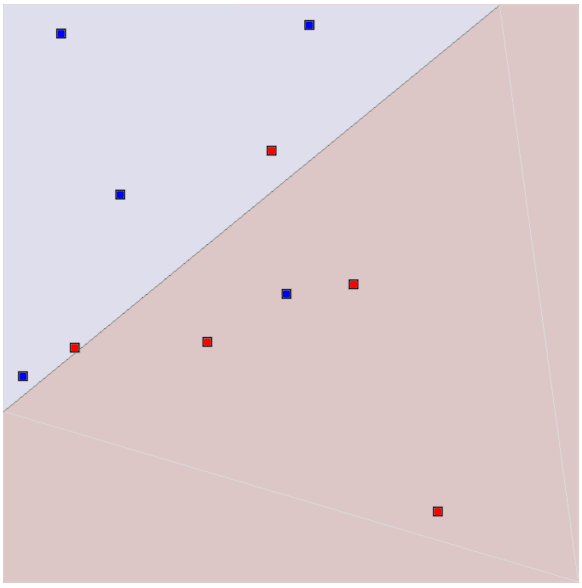

Figure 2: The output of the Marching Squares algorithm: rectilinear separation. Grid resolution is 2x2.

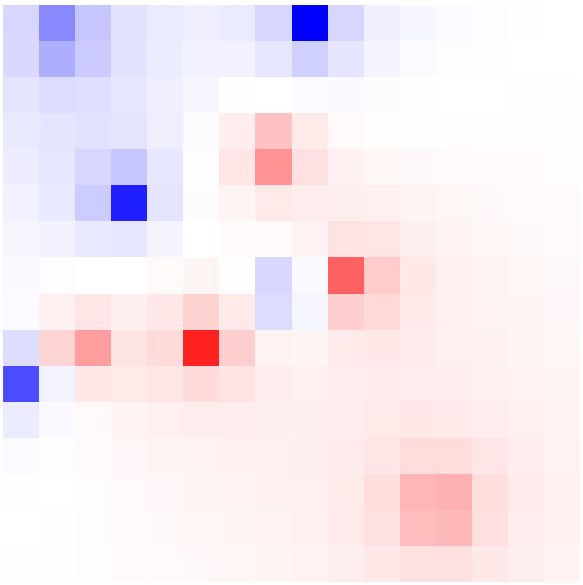

Figure 3: Bitmap image used as input to the Marching Squares algorithm. Image size is 16x16. Colour falls off with distance.

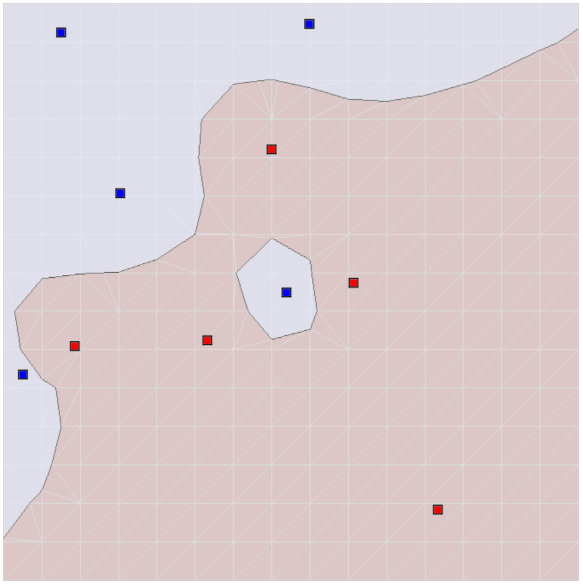

Figure 4: The output of the Marching Squares algorithm: curvilinear, radial separation. Grid resolution is 16x16.

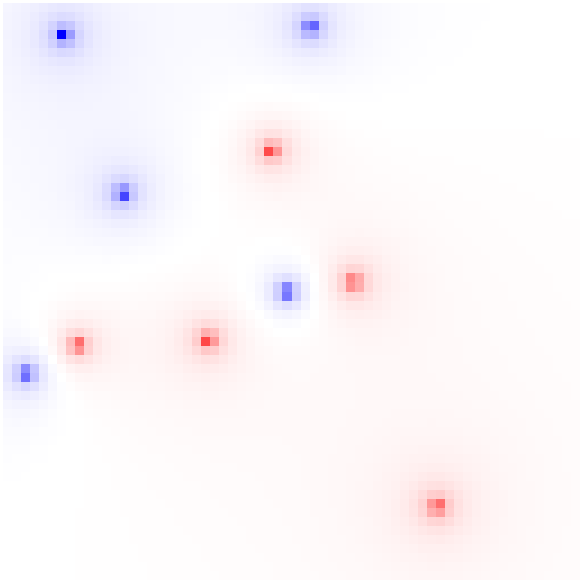

Figure 5: Bitmap image used as input to the Marching Squares algorithm. Image size is 64x64.

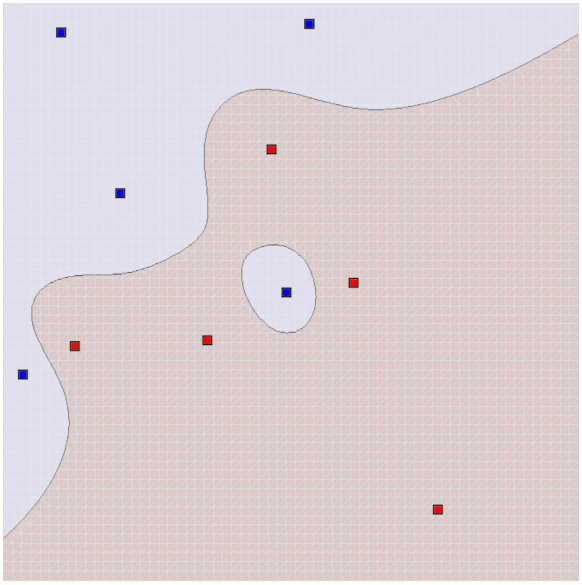

Figure 6: The output of the Marching Squares algorithm: curvilinear, radial separation. Grid resolution is 64x64.

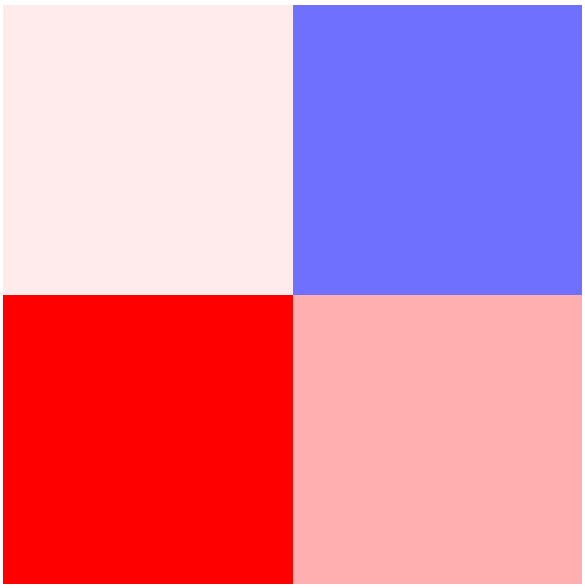

Figure 7: Bitmap image used as input to the Marching Squares algorithm. Image size is 2x2.

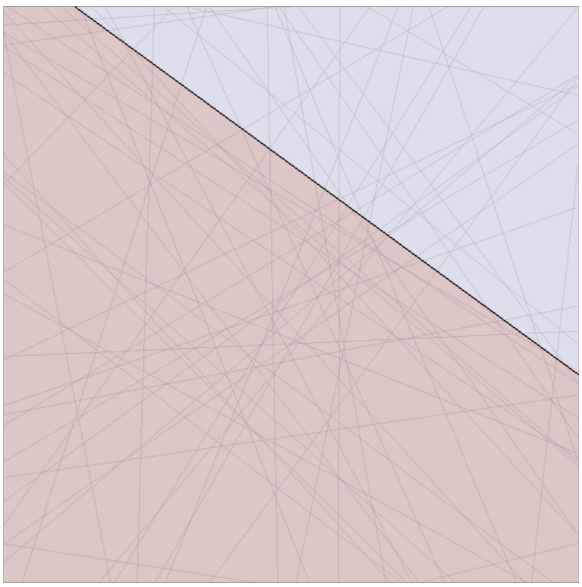

Figure 8: The output of the Marching Squares algorithm: rectilinear separation. Grid resolution is 2x2. All contour sets have been drawn in transparent grey, with the exception of the average contour set, which is drawn in pure black.

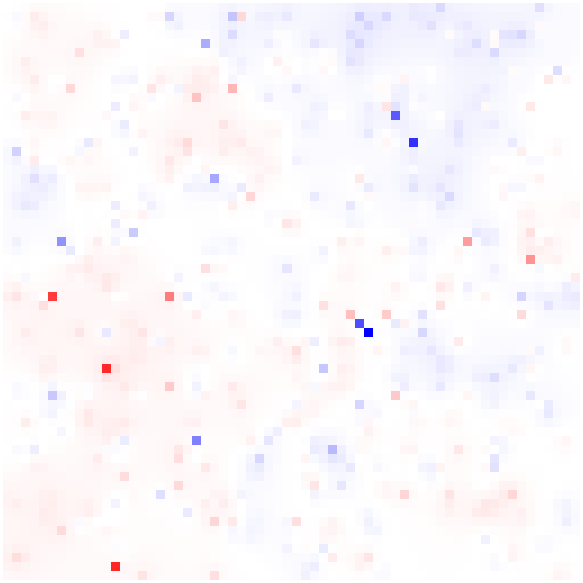

Figure 9: Bitmap image used as input to the Marching Squares algorithm. Image size is 64x64.

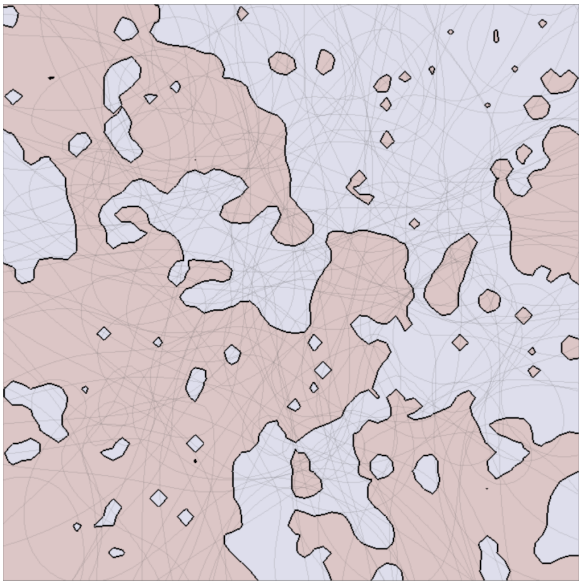

Figure 10: The output of the Marching Squares algorithm: curvilinear, radial separation. Grid resolution is 64x64. All contour sets have been drawn in transparent grey, with the exception of the average contour set, which is drawn in pure black.

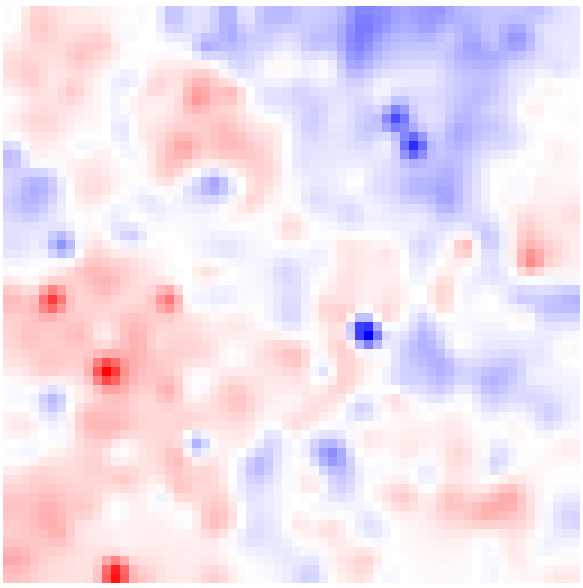

Figure 11: Bitmap image used as input to the Marching Squares algorithm. Image size is 64x64. One iteration of Gaussian blur has been used.

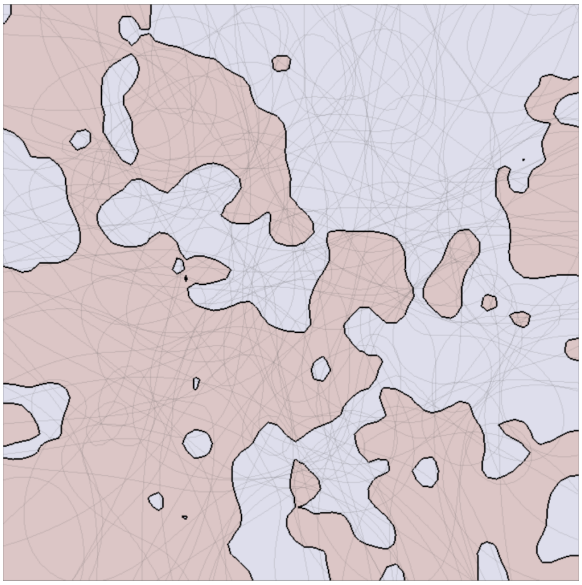

Figure 12: The output of the Marching Squares algorithm: curvilinear, radial separation. Grid resolution is 64x64. All contour sets have been drawn in transparent grey, with the exception of the average contour set, which is drawn in pure black.

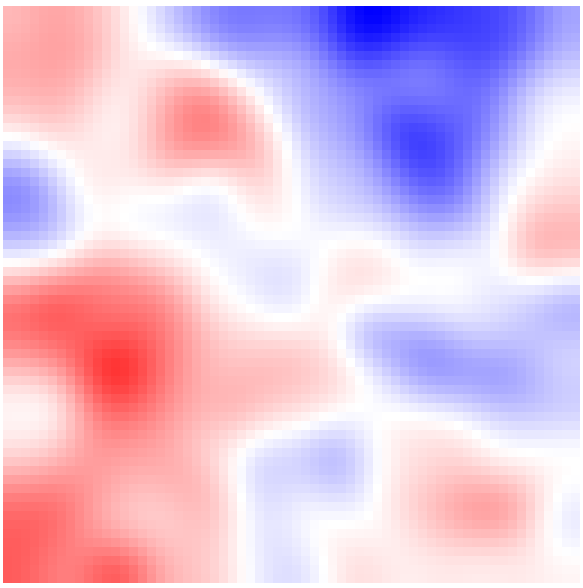

Figure 13: Bitmap image used as input to the Marching Squares algorithm. Image size is 64x64. Ten iterations of Gaussian blur have been used.

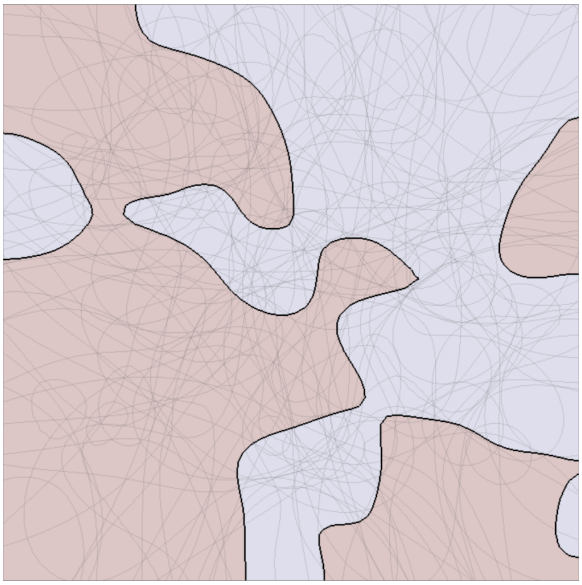

Figure 14: The output of the Marching Squares algorithm: curvilinear, radial separation. Grid resolution is 64x64. All contour sets have been drawn in transparent grey, with the exception of the average contour set, which is drawn in pure black.