Inferentialist, LLC

[quantitative modeling and statistical consulting for big data analytics]

Vision Repaired: The Inferentialist Logo

Dustin Lennon
Lead Statistician
`dustin@inferentialist.com`

Executive Summary

Vision is something that most people take for granted. Our eyes employ incredible algorithms that can efficiently process terabytes of information every day. Humans are naturally wired for making visual inferences. For ninety percent of men and close to one hundred percent of women, seeing happens in color, and color, therefore, has become an integrated component of our data collection process. In the language of data analysis, color is a fundamental component of the feature vector.

For anyone who has ever been tested for color blindness, the Inferentialist logo will undoubtedly evoke memories of Ishihara color tests. These tests create simple shapes or numbers out of small circles set against a background of similarly sized circles. The only distinguishing characteristic between the outline and the background is the color of the circles. Ishihara chose colors that were easily confused by color blind individuals, say green and red, and an inability to recognize the shape was an indication of color deficiency.

Our logo is similar in artistry, but rather different in spirit. Very few people have trouble seeing the blue part of the color spectrum, and setting a dark red against a light blue provides a high contrast experience. Compared to Ishihara, the signal in the Inferentialist logo, namely the letter "I", is screaming to be identified.

This simple visual recognition task requires the tacit processing of many sources of data. Our brains automatically aggregate color and spatial information as well as contextual cues. For example, the logo is followed by the text, "nferentialist," and this certainly primes our subconcsious for character recognition. At the same time, we magically ignore the meaningless variation in the sizes and spacings of the circles comprising the image. A human effortlessly aggregates all of these data streams, and this creates an illusion that identifying a bunch of red circles as being similar to the letter "I" is an easy problem.

At a higher level, the logo underscores this fundamental property of data analysis. Extracting the salient features of data typically requires both selection and aggregation of features from potentially disparate, and sometimes conflicting, data sources. However, with the right feature vector, say, one that includes color, signals are often self-evident.

In general, inference, at least the data preparation stages, is a soft process that more often resembles art than formal mathematics. And, in our view, the best analysis is the one that makes a signal look obvious.

This is a paper about how we created the logo. It is a paper that showcases a set of mathematical tools used to build a piece of art.

From a Keypress to a set of Closed Simple Paths

We've come to expect a certain simplicity from our technology. Fire up a word processor with a mouse click, hit the letter "I" on the keyboard, choose a font from a drop down menu, end of story. In our case, however, we need access to the actual shape of the letter generated. We need a step that takes the image on the screen and turns it into a mathematical recipe for drawing the letter. In order to talk about this, we have to define a few mathematical terms first.

First, define a *simple path* as putting a pencil on the paper and moving it around without ever allowing the curve to intersect. A *closed simple path* requires only a little more brainbending: make sure you finish where you started, closing the curve and, in the process, creating a region of the paper that's inside the curve and the region of the paper that's outside the curve.

A mathematician might want to be a little more precise about deciding which part of the region is "inside" and which part is "outside." It turns out that the convention that's used in this field of mathematics requires you to think of yourself walking along the closed simple path that you've just drawn (and along the same direction that you originally drew it). As you walk along the curve, the region that's always on your left is considered the *inside* and the region on your right is considered the *outside*.

This has non-intuitive consequences. If you drew your curve in a clockwise direction, you and the mathematician would probably disagree on what a normal person might call the "inside." See Figure 1 for a visualization.

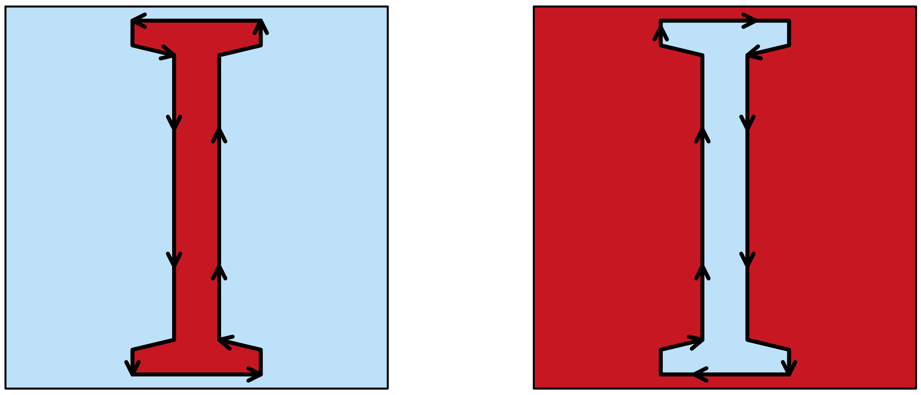

Figure 1: These two closed simple paths illustrate the mathematical "inside" (red) and "outside" (blue). By convention, the "inside" is on the left of a person walking along the path and the "outside" to the right.

The Letter R: A Better Example

Sadly, the letter "I" does not provide the best example for discussing the collection of algorithms used. The problem is that the region associated with an uppercase "I" has no holes. By this, we mean that one can trace the outline of the uppercase "I" without lifting the pencil from the paper. This is in contrast to the outline of the letter "R" which would require two separate closed simple paths: one to draw the shape of the letter and a second to draw the "hole" in the middle. The outline of the number "8" has two holes, one for the top "hole" and a second for the bottom "hole." See Figure 2.

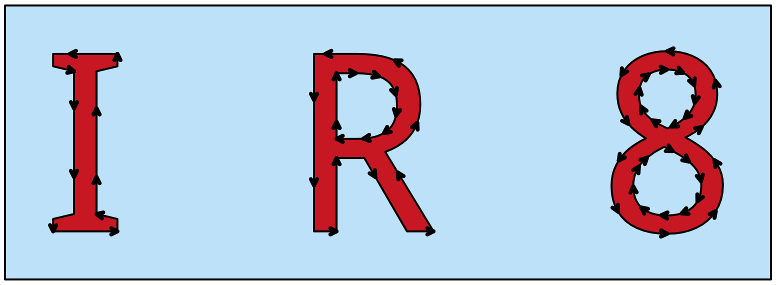
Figure 2: Examples of letters that require multiple closed simple paths. “I” has no holes; “R,” one hole; “8,” two holes.

The consequence of holes is that an internal representation for the outline of a single letter may require muliple closed simple paths.

Again, a picture may make this concept clearer. Using our concept of “inside” and “outside” from before, we see that the “R” in Figure 2 can be defined by the intersection of the “inside” regions of the two closed simple paths defining the outline. See Figure 3.

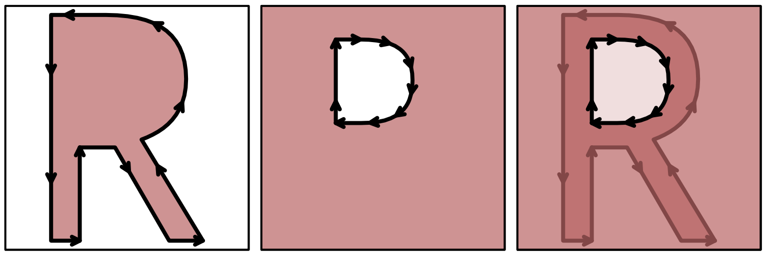
Figure 3: The two closed simple paths associated with the letter “R.” Adding the two leftmost images (alpha channels) together, we obtain the third pane indicating that the intersection of the two “inside” regions does indeed recover the domain of interest.

From an algorithmic perspective, holes add bookkeeping complexity. However, the key ideas remain largely unchanged. Thus, our working example will contain a hole, but the focus hereafter leaves any subtleties as an exercise for the reader.

The goal, then, is to create something similar to Figure 4. Note that we deliberately keep the resolution low in order to make the intermediate figures easier to manage.

In order to make progress, there are three subproblems that need to be addressed:

- Generate and extract the path information.
- Construct the set of circles.
- Identify which circles are inside.

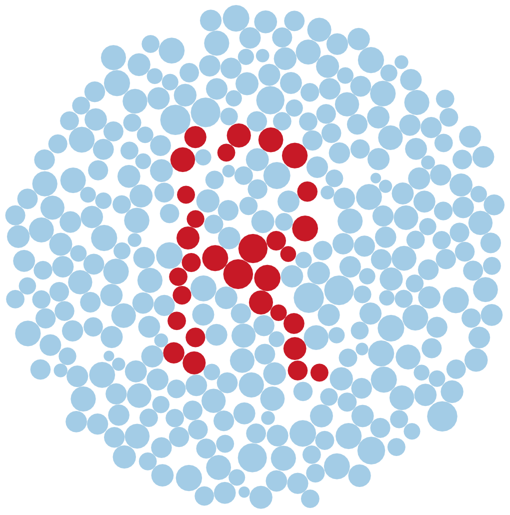

Figure 4: Our end goal.

Path Creation and Extraction

Creating a path is fairly easy using a vector graphics program such as Inkscape. Any text can be easily converted to a set of closed simple paths, and these paths exported to an SVG file. Command line tools such as `xml2` and `grep` are particularly helpful for extracting path information directly from an SVG file. Extracted path information can then be imported as lines of text into `R`, and parsed.

The basic SVG path syntax relies on three primitive operations: pick up the pen and move to a given point; given the current point, draw a line to the next point; given the current point and two control points, draw a Bezier curve to the next point.

This begs two questions: what is a Bezier curve? and how is this related to information that was actually extracted from the SVG file?

A Bezier curve is a fairly simple geometric object. Figure 5 shows an example. The Bezier curve is described by four points in the plane. These are the beginning and end points of the curve (in black) and the two control points (on the blue segments). Moving the control points changes the slope at the endpoints. The distance of a control point from its associated begin/end vertex influences how far out the Bezier curve moves in that direction.

In Figure 5, the red line segments are a coarse, linear approximation to the black, Bezier curve. As we increase the number of red line segments, they tend to provide a better and better approximation.

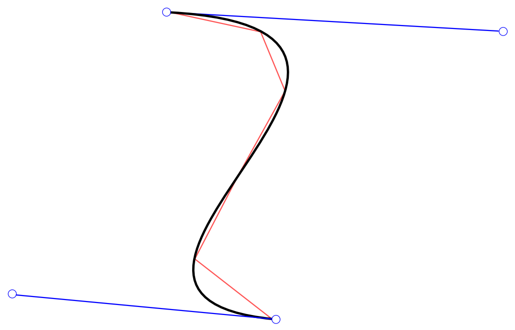

Figure 5: A Bezier curve, its control points, and a coarse linear approximation.

Bezier curves are one of the three primitive operations described in the SVG file. Figure 6 illustrates where the vertex and control point information is for the letter "R."

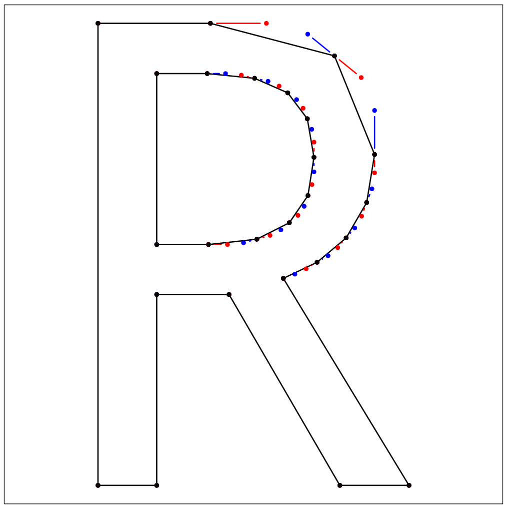

Figure 6: The vertex and control points on the letter “R” after extraction from an Inkscape SVG file.

Figure 6 shows each vertex and its control points. The vertices are connected with line segments. If we were to just connect the dots without taking the Bezier curve information into account, our “R” would look rather sad indeed. Our approach is to approximate the Bezier curves with shorter line segments, as illustrated in Figure 5.

All that is required, then, is to parse the path information contained in the SVG file and adaptively linearize the Bezier sections of each closed simple path.

Generating the Circle Set

A first proposal for generating the circle set might involve taking a collection of random points and expanding them until the growing circles start to bump into each other. A formal way of doing this uses a mathematical construct called a Voronoi diagram. One starts with a set of random points. Think of each point as a cell phone tower. Every point for which that cell phone tower is the nearest is grouped together. A Voronoi diagram is just the crossover boundaries between cell phone towers. As such, it partitions the space. Figure 7 shows an example.

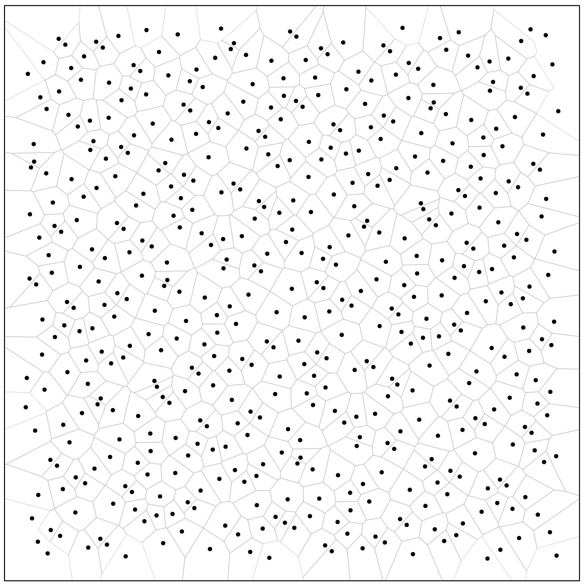

Figure 7: A collection of quasi-random points and their corresponding Voronoi diagram.

An astute reader may note that the caption in Figure 7 mentions quasi-random points. In fact we used a space-filling, Halton sequence to generate the points. This is a subtle detail, but gives us a stronger guarantee that the generated set of points will avoid gaps or clumping which can sometimes happen when using truely random numbers.

Zooming in on the Voronoi diagram, we fit the largest possible circle in each polygonal region. See Figure 8. It is interesting to note that in general, the center of the circle (the red point) is not the same as the point defining the Voronoi region (the black point).

In fact, finding the largest circle that fits in a polygon is not immediate. However, the problem is well understood, and a solution is called the Chebyshev center of the polygon. Fortunately, the underlying optimization structure gives rise to a linear programming problem, and the solution can be obtained efficiently through existing software.

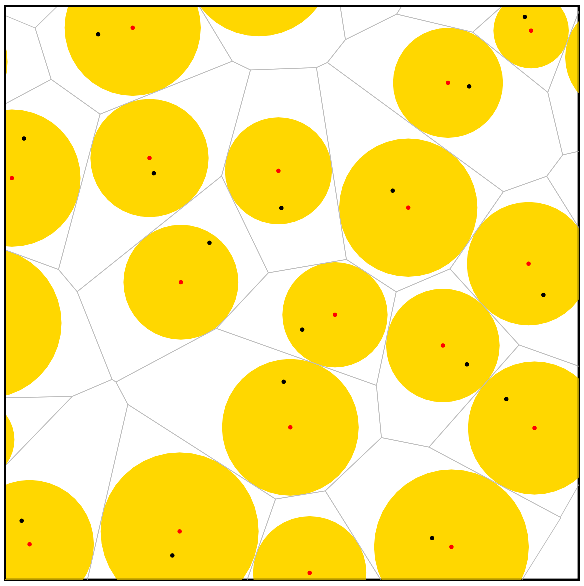

Figure 8: Computing the Chebyshev points. The black point defines the nearest neighbor region. The red point is the center of the circle.

In summary, to generate the circles: take a set of quasi random points and compute the Voronoi diagram; for each Voronoi region, find the Chebyshev center via linear programming.

Circles: Inside or Out

Combining the above, the last step is to determine which circles are inside the "R" and which circles are outside. This allows us to determine the final coloring of the circles shown in Figure 9.

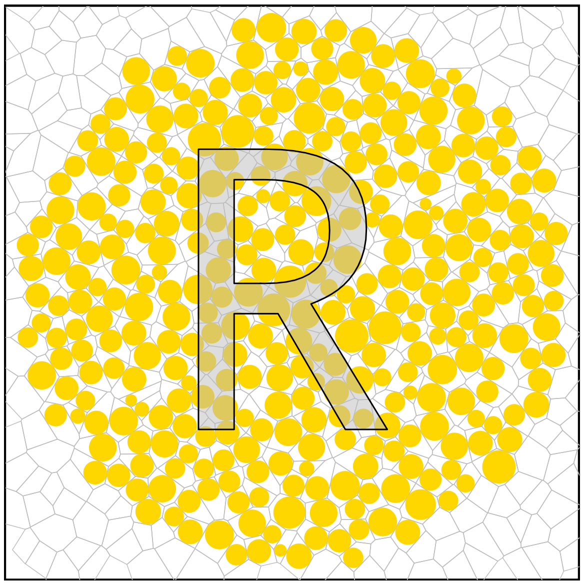

Figure 9: Which circles get colored?

To this end, we require a single, elementary computational geometry operation. We need to determine if a point, namely the Chebyshev center, is inside a closed simple path. Since we've chosen to use non-intersecting paths, the point in polygon problem is easily solved using linear algebra. To formulate this linear algebra problem, first recall that the adaptive linear approximation to the Bezier curves left us with a closed simple path that is, in fact, a polygon.

The key observation for the point in polygon problem is the following: if we take any test point and connect it to a point known to be outside of the polygon, then the following implication must be true: if the original point is "inside" the test segment will intersect the polygon boundary an odd number of times. Thus, we need to check whether the test segment intersects each side of the polygon and count the number of times this happens. However, testing whether two line segments intersection is just a simple linear algebra operation. Thus, we can efficiently determine if any point, in this case the Chebyshev center, is inside or outside of the closed simple path.

We then define a circle as inside if its center is inside, and analogously for the outside. This rule is chosen because it best respects (in expectation) the boundaries established by the closed simple paths for the letter "R."

Figure 10 returns briefly to the hole problem. Here, the circles are on the blue color scale if their center is outside the polygon and on the red color scale if their center is inside the polygon. A whiter color indicates that a circle is further away from the closed simple path. As indicated earlier, the circles that are “inside” both paths are the ones that are inside the “R” and will be colored accordingly in the final image.

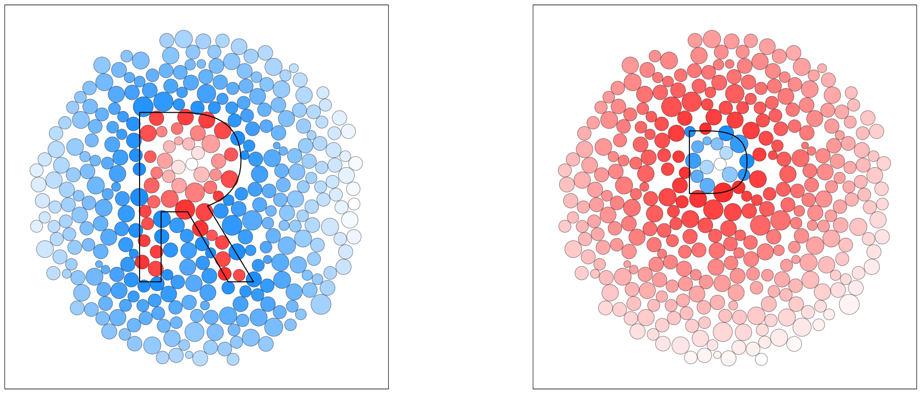

Figure 10: Circles are on the blue color scale if their center is outside the polygon and on the red color scale if their center is inside the polygon.

Our Logo

We add a few aesthetic, final touches, changing the font, removing the circle borders, and shrinking the circles so that they don't touch. For the final logo, we also use more quasi-random points, corresponding to a higher resolution effect.

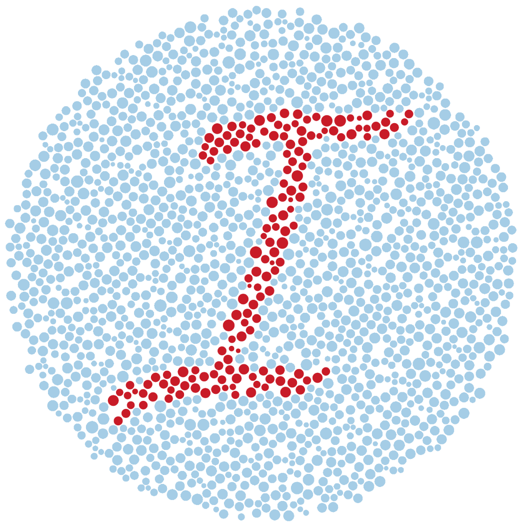

Figure 11: The Inferentialist Logo

R Code

This project was done entirely in `R`. Code is available on request.