Industrial & Applied Mathematics

4-Vector.org

A Blog with Annotated R-Notebooks: Math, Statistics, and Multiphysics for Learned Peer Discussion on the Science of Data

Analytical Construction of a Linear Decision Boundary for Machine-Learning Classification by Regression

Michael A. X. Izatt
E | Michael.Izatt@Alum.MIT.Edu (mailto:Michael.Izatt@Alum.MIT.Edu)
E | izatt@UChicago.Edu (mailto:izatt@UChicago.Edu)
W | 4-Vector.org
In | www.linkedin.com/in/max-izatt

January 23, 2020

Version 2020-01-23-1800/MAI

Abstract

Support Vector Machines (SVM) is the crown jewel of optimization theory and has become a very popular tool for classification. SVM is easily invoked by any machine-learning library. It can calculate a curvilinear decision boundary by lifting the inseparable classes into a higher N-Dimensional space where an [N-1]-Dimensional hyperplace can separate the data with remarkable accuracy, precision, and recall; however, like many advanced algorithms in optimization theory, notwithstanding the algo's ability to classify a class, no insight into the problem set's data-generating function is obtained by applying the SVM algorithm. Of course, if the DGF's causal factors are unknown or unknowable, and a classification must be made, SVM is unsurpassed, but in industrial and applied mathematics, there is a strong interest in indentifying causal factors that affect the revenue, costs, margins, and profits of the business firm so that predictive and prescriptive algorithmic tools can be deployed to optimize operations. In this problem space, multivariate linear regression with causal factors is unsurpassed. This article shows how a curvilinear decision boundary that is the purview of SVM can be transformed to a linear decision boundary with elementary analytical techniques, where logistic regression can make a perfect classification. We celebrate Vapnik's SVM, but we argue that its problem domain is far downstream from the typical industrial problem set, and indeed that elementary analytics can take you far when coupled with causal factors and traditional regression algorithms for scoring and classification.

Discussion

William Faulkner and Ernest Hemingway had an erudite, public disagreement. Faulkner resented Hemingway's terse, concise, compact prose, which no doubt was informed if not preordained by Hemingway's long stint as a newspaper correspondent with the Kansas City Star and Toronto Star Weekly. Western Union charged by the word, so concision was key. Furthermore, Hemingway was in esteemed company of journalism alumni - Mark Twain, Stephen Crane, Theodore Dreiser, and Sinclair Lewis, among others, once wrote by the column inch. Faulkner was unimpressed by the association. He said of Hemingway that Hemingway had no courage, that he had "never been known to use a word that might send the reader to the dictionary."

Hemingway was having none of it. "Poor Faulkner. Does he really think big emotions come from big words? He thinks I don't know the ten-dollar words. I know them all right. But there are older and simpler and better words, and those are the ones I use."

Simple tools, in the hands of the master, take you far.

I reflect on this exchange often in my professional practice of industrial and applied mathematics. There are many powerful software packages available to machine-learning technicians, and the algorithms are the result of decades of advancements in optimization theory. The algorithms can mine the data to recognize even the most subtle and obscure patterns, and cluster, associate, correlate, score, and classify even the most seemingly intractable datasets. For many of the algorithms, very little understanding of the underlying causal factors that produce the data is required, in which case, the impressive algorithmic results give little business or industrial insight into the data source and character. Root cause can be as elusive after the successful application of the algorithm as before. Nevertheless, buoyed by black-box prestidigitation, aspiring data scientists increasingly rely on such statistical packages at the expense of root-cause business analysis.

(At least) two things are missing in this approach.

First, the role of the scientist is understanding. In the context of data-science discovery, understanding emerges from investigating, hypothesizing, testing, identifying, and validating the mathematical, physical, statistical, and geometrical properties that characterize the problem set's data-generating function. It is from this model that reproducible, tunable, predictive, and prescriptive analysis is possible.

This vocation is not for the faint at heart. The analyst must lean into the problem and challenge the mathematics, perhaps with unrequited enthusiasm; despite the analyst's best efforts, the problem may not budge. We kiss many proverbial frogs. We punt. A lot. Progress sometimes becomes regress. Alternating periods of disappointment, confusion, elucidation, sleeplessness, and fleeting excitement are constant companions until the problem is cornered in a cul-de-sac of analysis and the solution stands tall and alone. It is the answer because it could be no other. And no sooner than that solution is deployed do we start again.

It is this dogged, blue-collar, full-contact, actively-engaged, erudite analysis that is the corpus of scientific discovery. It is the nimble, dexterous, virtuosi application of maths, Multiphysics, statistics, hypothesis testing, and simulation all to isolate and explain a conjectured effect that may have no

discernable cause using an experiment that may not work, and subsequently prevailing after all of that. It is for this immutable reason that we scientists can immediately recognize one of our own; the craft is both wide and deep. Especially deep.

Secondly, despite the outrageous claims of software vendors and the preternatural aspirations of young data “scientists,” black-box software algorithms cannot replace this understanding, else dexterity with TurboTax would qualify one as a tax attorney or Quicken as a chartered accountant.

Armed with software that **seemingly** – but importantly, not **actually** – outpaces one’s numeracy, it is common sport for young aspiring data scientists to apply the most powerful machine-learning algorithms to even the most tractable analytic problem sets and, in the process, to forego mathematical and statistical analysis en route to classifying, scoring, clustering, and associating data. Sometimes the software is faithful to the Hippocratic Oath to first do no harm, then help, but sometimes not. It is a fool’s errand exactly because the goal of the craft is not to make the classifications and calculate the scores, but rather to discern the causal factors that comprise the data-generator of the data.

This small article uses an illustrative problem set in a seemingly non-linear decision boundary to benchmark one of optimization theory’s crown jewels, the Support Vector Machine, to highlight the power of this amazing algorithm to learn a seemingly intractable decision boundary.

We then ***improve*** on this result by applying analysis and invoking an old workhorse, the logistic-regression, to illustrate how domain knowledge and elementary mathematical tools can take us far. Vapnik and the 20th century’s optimization-theoretical gems are to be celebrated, but Archimedes, Pythagoras, Decartes, Newton, and Gauss are our guides for a good long while in this business.

Let’s get started.

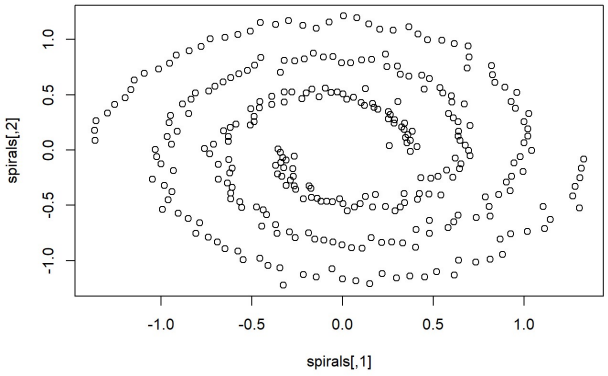
Analysis

We will use the kernlab library and its spiral dataset as our muse.

https://cran.r-project.org/web/packages/kernlab/vignettes/kernlab.pdf (https://cran.r-project.org/web/packages/kernlab/vignettes/kernlab.pdf)

The kernlab library contains several datasets, and its spirals dataset is particularly useful for our experiment. Load the spirals data and take a look/see of the data.

```
data(spirals)

plot(spirals)
```



Examine the structure of the spirals dataset

```
str(spirals)
```

```
##  num [1:300, 1:2] 0.812 -0.268 0.374 0.258 -0.847 ...
```

It seems that the spirals dataset comes to us as a 2-D matrix. Confirm that spirals is an R matrix

```
class(spirals)
```

```
## [1] "matrix"
```

It is indeed a matrix object. Convert it to a dataframe and examine its structure.

```
df_ <- as.data.frame(spirals)

str(df_)
```

```
## 'data.frame':    300 obs. of  2 variables:
##  $ V1: num  0.812 -0.268 0.374 0.258 -0.847 ...
##  $ V2: num  -0.9871 -0.3255 -0.0129 0.0413 0.3294 ...
```

So it is a spiral system of two interlaced rotations. There are two features in the dataframe, which we can see are an x-abscissa and y-ordinate in rectangular coordinates. Rename the features ‘x’ and ‘y’ and confirm by observing the structure, above.

```
names_ <- c('x_','y_')

names(df_) <- names_

str(df_)
```

```
## 'data.frame':    300 obs. of  2 variables:
##  $ x_: num  0.812 -0.268 0.374 0.258 -0.847 ...
##  $ y_: num  -0.9871 -0.3255 -0.0129 0.0413 0.3294 ...
```

There are two interlaced spirals with abscissa and ordinate for each point, but there is no class membership indicated in the data. We can do spectral decomposition on the spirals and attach a class identification to the data frame using the kernlab package.

Physically, the two spirals suggest a data-generating function (and perhaps its twin) that is producing two identical telemetry streams that have a phase difference of pi radians. If we were *gathering* the data from our own experimental apparatus, we would design the experiment to denote the phase of the generating function, but in this illustrative case the data comes to us without labels, so we must apply spectral methods. We will apply the kernlab specc function and ask for two centers, which we determine from observation of the preliminary visualization, above. Store the specc object in the spiral_centers_ variable and examine the structure of the specc object.

```
spiral_centers_ <- specc(spirals, centers=2)

str(spiral_centers_)
```

```
## Formal class 'specc' [package "kernlab"] with 5 slots
##   ..@ .Data   : int [1:300] 2 2 1 1 2 1 1 1 2 1 ...
##   ..@ centers : num [1:2, 1:2] -0.0177 0.02 0.1775 -0.1761
##   ..@ size    : int [1:2] 150 150
##   ..@ kernelf :Formal class 'rbfkernel' [package "kernlab"] with 2 slots
##   .. .. ..@ .Data:function (x, y = NULL)
##   .. .. ..@ kpar :List of 1
##   .. .. .. ..$ sigma: num 369
##   ..@ withinss: num [1:2] 118 117
```

There is a LOT of information in the spiral_centers_ specc object. Convert the specc object to a dataframe and table the class assignments.

```
df_sc_ <- as.data.frame(spiral_centers_)

table(df_sc_)
```

```
## df_sc_
##   1   2
## 150 150
```

There are 300 points in the spirals matrix, with an exact 50/50 split between classes. The authors of kernlab package engineered 150 points for each of the two series. Extract the class from the .Data attribute of the sc_ object

```
rows_ <- dim(df_sc_)[1]

class_ <- df_sc_[1:rows_,1]

class_
```

```
##   [1] 2 2 1 1 2 1 1 1 2 1 1 2 2 1 1 2 2 2 2 2 1 1 2 1 1 1 1 2 2 2 1 2 1 1 2 1 2
##  [38] 1 2 2 1 1 1 1 2 2 2 2 2 1 2 1 2 2 1 1 1 2 2 2 2 2 1 1 2 1 2 2 2 1 1 2 1 1
##  [75] 1 2 2 2 2 2 1 2 1 2 1 2 2 2 2 2 2 1 2 2 1 1 1 2 1 1 1 1 2 2 2 1 1 2 1 1 1 2
## [112] 1 2 2 2 2 2 1 1 2 2 1 2 2 2 1 2 1 2 2 2 2 2 1 1 1 1 1 2 2 2 1 1 2 1 2 2 2 1
## [149] 1 1 2 1 1 1 1 1 1 2 2 2 2 1 2 1 2 2 2 1 2 2 2 2 1 2 1 2 2 2 1 1 2 2 2 1 1
## [186] 1 2 2 1 1 1 1 1 1 1 1 2 2 2 2 1 2 1 1 2 1 2 1 1 1 1 1 2 1 2 1 2 1 2 2 2 1
## [223] 1 1 1 2 2 2 1 2 2 1 1 1 1 1 2 2 1 1 1 1 2 2 2 1 1 1 1 1 2 2 1 1 2 2 2 2 2
## [260] 2 2 2 1 1 1 1 2 1 2 1 2 2 1 1 1 2 1 2 2 2 2 1 1 2 1 2 1 1 1 2 1 2 1 1 2 2
## [297] 1 1 1 2
```

In general, we are agnostic about labels, but in this case, we are going to write a *lot* of downstream processing that relies on the orientation of the labels on the two spirals. We need the labels to be

1,1,2,2,1,2,2,2,1,2,2,…2,2,2,1

and *not*

2,2,1,1,2,1,1,1,2,1,1,…1,1,1,2

So having let the specc function do its magic, let's manhandle the labels and set them as we require. Do the substitution and examine the class assignments once more.

```
if(class_[1] == 2){

  class_ <- ifelse(class_ == 1, 2, 1)

}

class_
```

```
##   [1] 1 1 2 2 1 2 2 2 1 2 2 1 1 2 2 1 1 1 1 1 2 2 1 2 2 2 2 1 1 1 2 1 2 2 1 2 1
##  [38] 2 1 1 2 2 2 2 1 1 1 1 1 2 1 2 1 1 2 2 2 1 1 1 1 1 2 2 1 2 1 1 1 2 2 1 2 2
##  [75] 2 1 1 1 1 2 1 2 1 2 1 1 1 1 1 1 2 1 1 2 2 2 1 2 2 2 2 1 1 1 2 2 1 2 2 2 1
## [112] 2 1 1 1 1 2 2 1 1 2 1 1 1 2 1 2 1 1 1 1 1 2 2 2 2 2 1 1 1 2 2 1 2 1 1 1 2
## [149] 2 2 1 2 2 2 2 2 2 1 1 1 1 2 1 2 1 1 1 2 1 1 1 1 2 1 2 1 1 1 2 2 1 1 1 2 2
## [186] 2 1 1 2 2 2 2 2 2 2 2 1 1 1 1 2 1 2 2 1 2 1 2 2 2 2 2 1 2 1 2 1 2 1 1 1 2
## [223] 2 2 2 1 1 1 2 1 1 2 2 2 2 2 1 1 2 2 2 2 1 1 1 2 2 2 2 2 2 1 1 2 2 1 1 1 1
## [260] 1 1 1 2 2 2 2 1 2 1 2 1 1 2 2 2 1 2 1 1 1 1 2 2 1 2 1 2 2 2 1 2 1 2 2 1 1
## [297] 2 2 2 1
```

Now attach the class to the dataframe.

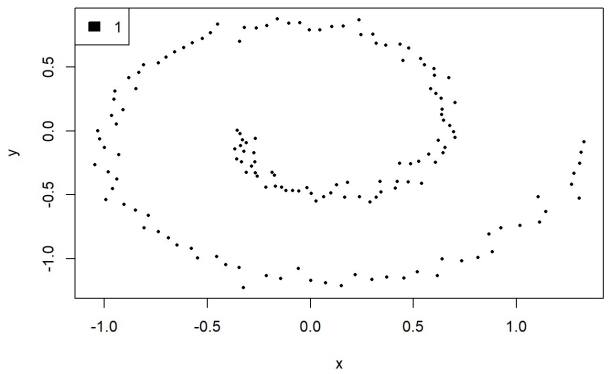
```
df_$class_ <- class_

str(df_)
```

```
## 'data.frame':    300 obs. of  3 variables:
##  $ x_    : num  0.812 -0.268 0.374 0.258 -0.847 ...
##  $ y_    : num  -0.9871 -0.3255 -0.0129 0.0413 0.3294 ...
##  $ class_: num  1 1 2 2 1 2 2 2 1 2 ...
```

Now that we have identified class membership to each Cartesian coordinate, let's plot the two spirals separately to get a reference and make it easy on the eyes once we begin analysis. Let's start with the first class and plot with the class labels and fill Class # 1 with black color.

Note that from a polar-coordinate point-of-view, the series spans {r,theta} approximately from {0.40, 1 * pi} to {1.5 , 4 * pi}.

```
subset_1_ <- subset(df_,df_$class_ == 1)


filled_circle_ = 16

plot(x=subset_1_$x_, y=subset_1_$y_, type='p',col="black", cex=subset_1_$class_*.5, pc
h=filled_circle_, xlab='x', ylab='y')

legend("topleft",legend=c("1"),fill=c("black"))
```



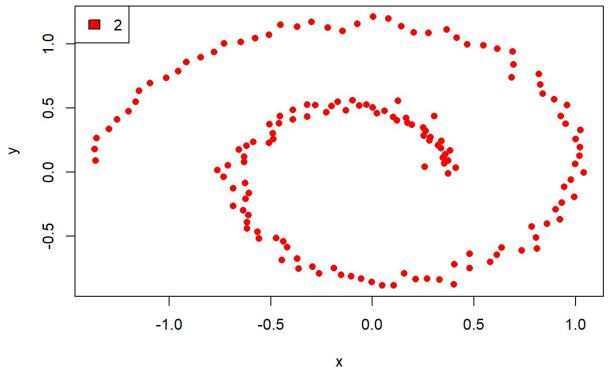
Note that I have introduced a bias from my work in the complex plane. The inner points at the start of the spiral data series with abscissa ~ -0.4 and ordinate ~ 0.0 are most-likely in the 3rd quadrant, which in terms of such analysis is often said to be - 1 x pi **clockwise** from the x-axis. And indeed if you ask a software package's arc-tangent function, it would indeed be labeled as such, and we will indeed see that behavior *infra.* Those who have worked in the complex plane or polar coordinates might prefer to think in an anti-clockwise orientation from the positive x-axis, in which case the very same points are said to be + 1 x pi anti-clockwise from the x-axis. It is seemingly the same place, if you consider the

plane to be 360-degrees in range, but the complex plane imposes no such limitation. In the metaphor of the complex plane, the points at {x,y} ~ {1.5, 0.0} are said to be + 4 * pi anti-clockwise from the positive x-axis.

A few quick observations. Notice that most of the black #1 series sit on a nice curvilinear spiral, but there are regions of the arc where the points are experiencing some kind of stochastic or systemic noise. For example, the points in the region from {x,y} = {+0.25, +0.6} to {-0.5, +0.6} fall off of the smooth arc. Likewise for those another pi/2 along the arc in the {x,y} region ~ {-1.0, +0.4} to {-1.0, -0.5}. And most of the points along the final length of arc where y ~ -1.0 are jittery. More on this *infra.*

Now for class #2. From a polar-coordinate point-of-view, the series spans {r,theta} approximately from {0.40, 0} to {1.5 , 3 * pi}

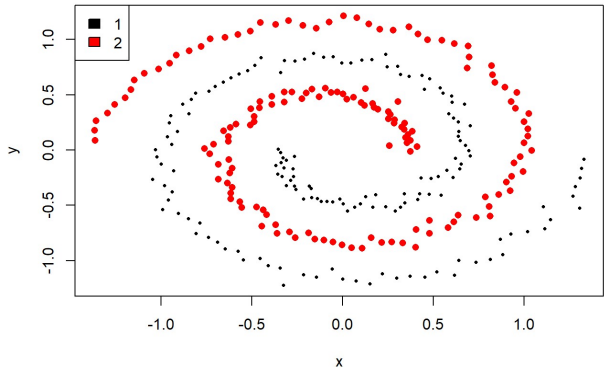
```
subset_2_ <- subset(df_,df_$class_ == 2)

filled_circle_ = 16

plot(x=subset_2_$x_, y=subset_2_$y_, type='p',col="red", cex=subset_2_$class_*.5,pch=f
illed_circle_, xlab='x', ylab='y')

legend("topleft",legend=c("2"),fill=c("red"))
```



Note the areas of stochastic or systemic jitter.

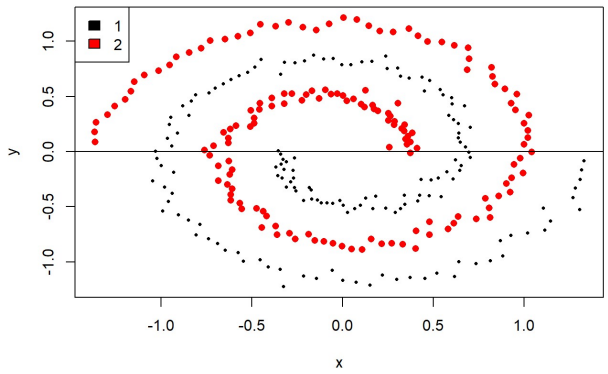
With that reference work done, we are ready to re-plot the entire data set and show class membership. Let's plot the spirals with the class labels and fill based on class.

```
filled_circle_ = 16

col_ <- c("black","red")

plot(x=df_$x_, y=df_$y_, type='p',col=df_$class_, cex=df_$class_*.5 ,pch=filled_circle
_, xlab='x', ylab='y')

legend("topleft",legend=c("1","2"),fill=col_)
```



The spirals are somewhat unsatisfying if we wish to express these points in a rotational reference frame. The first few inner points of each spiral seem to straddle the x-axis, and it would very pleasing to have those points cleanly demarked by the Cartesian coordinate system so that Cartesian-to-Polar and trigonometric transforms would be maximally elegant. Let's add an AB line as a straight edge to test and validate that observation

```
filled_circle_ = 16

col_ <- c("black","red")


plot(x=df_$x_, y=df_$y_, type='p',col=df_$class_, cex=df_$class_*.5, pch=filled_circle
_, xlab='x', ylab='y')

abline(h=0)

legend("topleft",legend=c("1","2"),fill=col_)
```



It is really only a couple of data points, but it would be much more satisfying to rotate the system to begin the spirals in a more analytically-tractable starting position. Let's rotate the system by 10 degrees = 10 * pi / 180 radians.

Recall from linear algebra that we can rotate Cartesian coordinates by constructing the rotation matrix R where R is

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

A vector v can be rotated in Cartesian coordinates through an angle theta by calculating Rv in this manner

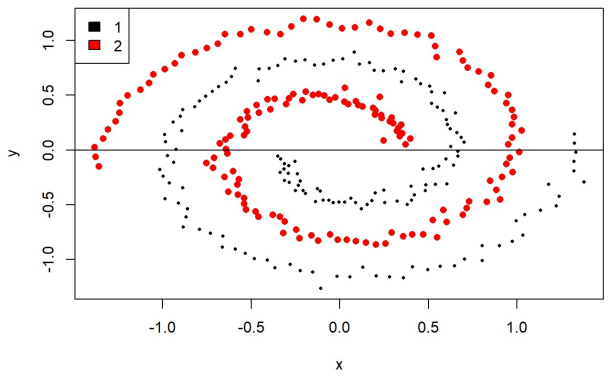
$$R\mathbf{v} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$

The R statistical language has a code line to accomplish this, but for illustrative purposes, let's build the rotation matrix R from scratch in Base R.

First we build the [2x2]-dimensional rotation matrix R, then copy the data frame's x and y coordinates into a [300 x 2]-dimensional vector v. We are then in position to rotate the v matrix by calculating Rv. There is a small linear-algebraic task. Matrix multiplication requires that the two inner dimensions match, and as it is written, R is [2x2]-dimensional and v is [300 x 2]-dimensional, so we must transpose v to do the multiplication, then transpose matrix product back to its [300 x 2]-dimensional state. The t() function does transposition in Base R, so the linear algebraic gymnastics are t(R %*% t(v)), where the '%%' operator is matrix multiplication in Base R. Then we will re-attach the feature names for x, y, and class, and store the rotated dataset in a new dataframe called df_rot.

So, to rotate the dataset 10 degrees = 10 * pi/180 radians counterclockwise, we must do the following:

```
theta_ = 10 * pi / 180
R <- matrix(1:4, nrow = 2, ncol = 2)
row_1_ <- c(cos(theta_), -sin(theta_))
row_2_ <- c(sin(theta_), cos(theta_))
R <- rbind(row_1_, row_2_)

v <- as.matrix(cbind(df_$x_, df_$y_))

df_rot_ <- as.data.frame(t(R %*% t(v)))
names_ <- c('x_','y_')
names(df_rot_) <- names_

df_rot_$class_ <- df_$class_


filled_circle_ = 16

col_ <- c("black","red")

plot(x=df_rot_$x_, y=df_rot_$y_, type='p',col=df_rot_$class_, cex=df_rot_$class_*.5, p
ch=filled_circle_, xlab='x', ylab='y')

abline(h=0)

legend("topleft",legend=c("1","2"),fill=col_)
```

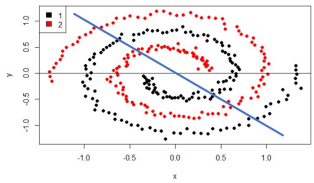


That cleaned up our data quite nicely. Our spirals now begin cleanly in a single Cartesian quadrant, which will facilitate our downstream analysis.

Now the question is whether we can build a classifier that can learn this crazy dataset and classify a given point to its class, and if so, then with what accuracy, precision, and recall? It is a very heavy lift to do so since the intuitive decision boundary is not simply curvilinear, but it is curvilinear of increasing radius; that is, the decision boundary is itself a spiral that is situated between the two interlaced spirals of the dataset. I mean, if our toolbox was say the logistic regression algorithm, then we would be tasked

with fitting a linear decision boundary in this crazy spiral data. This problem bears no resemblance to the illustrative problem sets for algorithms such as the logistic regression that draw a linear decision boundary between two classes.

We can certainly do that calculation, but the quality metrics – accuracy, precision, recall, the lot – are, shall we say, not of industrial quality, at least not if we have value at risk in the form of life, limb, or money damages.



Actually, the facious illustration above can be improved; try to convince yourself that the actual boundary would be along the x-axis. That decision boundary would orphan the fewest number of both black and red points and capture as much of the zeroth-order linear classifications possible.

There is an amazing classifier called the Support Vector Machine that was invented in 1963 by Vladimir Vapnik and Alexey Chervonenkis, who defined the linear classifier that was extended to the non-linear classifier in 1992 by Bernhard Boser, Isabelle Guyon, and Vapnik. For the non-linear case, they employed a technique termed the kernel trick that lifts the N classes of data into an N-dimensional space where it can be separated by an [N-1]-dimensional hyperplane.

The SVM is an inspirational application of Lagrangian optimization theory that allows the algorithm to “learn” the characteristics of an unseen dataset, construct the widest practicable margins between the classes, and predict a classification of the classes for excellent accuracy, precision, and recall.

In the R statistical language, we invoke SVM with the ksvm function. We first load the class into the y variable, and construct the analytic data set (ADS) X as the x and y Cartesian coordinates of the rotated data frame. We then invoke the ksvm algorithm and ask it to use the Gaussian radial basis kernel, which is simply the exponential function.

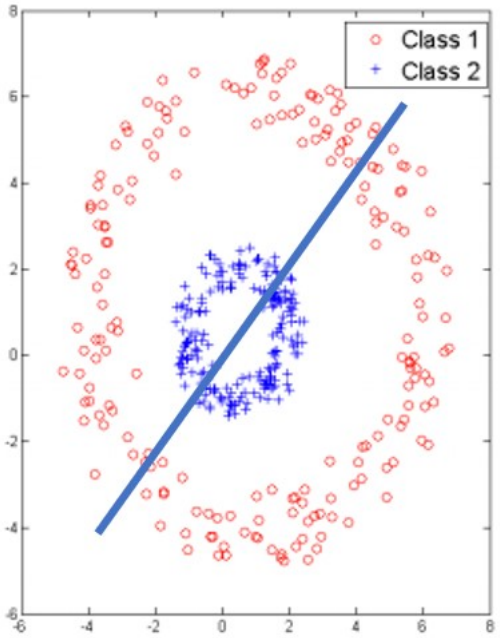
$$k(\vec{x_i}, \vec{x_j}) = \exp(-\gamma \|\vec{x_i} - \vec{x_j}\|^2) \text{ for } \gamma > 0$$

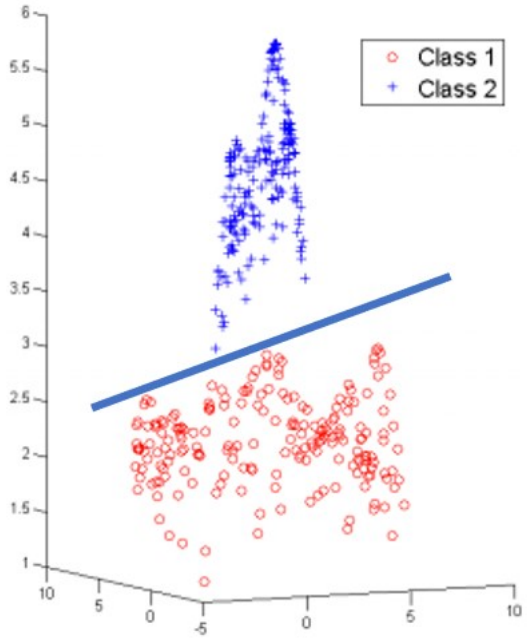
Recall from the Calculus that the Taylor exapansion of the Exponential function has all powers of the argument and converges for all values of the ordinate.

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Operationally, the SVG kernel trick using the Gaussian Radial Basis Kernel lifts the data into a very high dimensional space using synthetic features of all powers of x and y, where the two classes can be separated with a hyperplane.

In the illustration, the Gaussian Radial Basis Function Kernel is able to classify data that is separated by a circular (not spiral in the illustration, but circular) by lifting the dataset into a high-dimensional space where a hyperplane can easily separate and classify the data.





The R statistical language exposes the ksvm wrapper to the Support Vector Machine algorithm. We load the target class into the y variable and the Cartesian abscissa and ordinate into an analytic data set (ADS), X. We choose the Gaussian Radial Basis Function kernel, rbfdot, and invoke the algo. For this purpose, we predict directly back on the training set and write a confusion matrix. I have been working on this writing for a few days, and every so often, I copy the results of the SVM into the code block and comment it for comparison. You can see that the SVM performs brilliantly with only a handful of false positives and false negative among the 300 data points, for extraordinary accuracy, recall, and precision.

```
y_ <- df_rot_$class_

y_ <- y_ - 1

X_ <- as.data.frame(cbind(df_rot_$x_, df_rot_$y_))

ksvm_ <- ksvm(y_ ~ ., data=X_,kernel='rbfdot')

y_hat_ <- predict(ksvm_, newdata=X_, type='response')

table("class"=y_, "classifier" = y_hat_ >= 0.5)
```

```
##      classifier
## class FALSE TRUE
##     0   101   49
##     1    52   98
```

```
#      classifier
# class FALSE TRUE
#     0   TN   FP
#     1   FN   TP


# Confusion matrix from a prior run
#      classifier
# class FALSE TRUE
#     0   147    3
#     1     6  144


# Confusion matrix from a prior run
#      classifier
# class FALSE TRUE
#     0   147    3
#     1     3  147

# Confusion matrix from a prior run
#      classifier
# class FALSE TRUE
#     0   147    3
#     1     2  148
```

This is raw genius. Having no knowledge whatsoever of the data-generating function, the SVM was able to classify all but five data perfectly using the radial-basis kernel to lift the points into a higher-dimensional space. In the Cartesian plane,they are separated by an intractable non-linear decision boundary, but in the higher-dimensional space of the Gaussian Radial Basis Function, the SVM was able to classify nearly perfectly.

Having applied our crown jewel of optimization theory to the problem, let us now apply a bit of mathematical analysis to see just how far elementary techniques can take us toward classifying the two classes.

The work of a scientist, even a data scientist, is very often asking and answering the question, “What is the natural coordinate frame for this problem set?” Very often, a difficult problem in one coordinate system is trivial in another. Our system is spiral and rotational in nature, and it is natural to wonder how these data visualize under a Cartesian-to-Polar mapping. Let’s do that work and take stock once we are there.

Recall from your junior high school days that we can transform {x,y} to {r, theta} with the transformations

$$r = \sqrt{x^2 + y^2}$$

$$\varphi = \operatorname{atan2}(y, x)$$

where the atan2 function returns the proper quadrant for the point based on the signs of x and y when the atan function would be confused based on the sign of the ratio y/x.

$$\operatorname{atan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & \text{if } x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & \text{if } x < 0 \text{ and } y \geq 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & \text{if } x < 0 \text{ and } y < 0 \\ \frac{\pi}{2} & \text{if } x = 0 \text{ and } y > 0 \\ -\frac{\pi}{2} & \text{if } x = 0 \text{ and } y < 0 \\ \text{undefined} & \text{if } x = 0 \text{ and } y = 0. \end{cases}$$

Calculate the r and theta values, add them to the rotated dataframe, and plot once again. Bear in mind that

Black Class = #1, {r,theta} from {0.25, +1 * pi} to {1.5 , +4 * pi} Red Class = #2, {r,theta} from {0.40, 0} to {1.5 , +3 * pi}

but that we expect the atan2 function to return these values as follows:

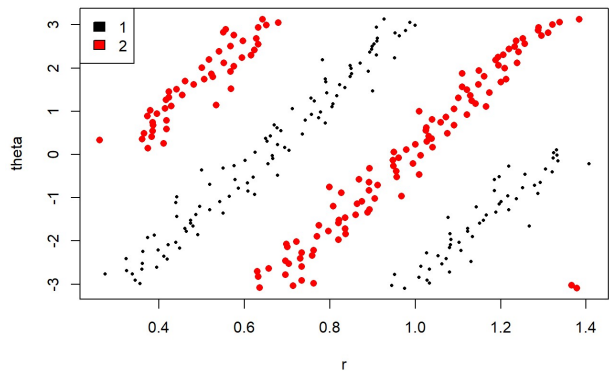
Black Class = #1, {r,theta} from {0.25, - 1 * pi} to {1.00 , + 1 * pi} AND from {1.00, - 1 * pi} to {1.50 , + 1.* pi}

and Red will be ever so slightly messier:

Red Class = #2, {r,theta} from {0.10, zero } to {0.7 , + 1 * pi} AND from {0.60, -1 * pi } to {1.4 , + 1 * pi} AND just a couple of data points at {1.4, - 1 * pi }

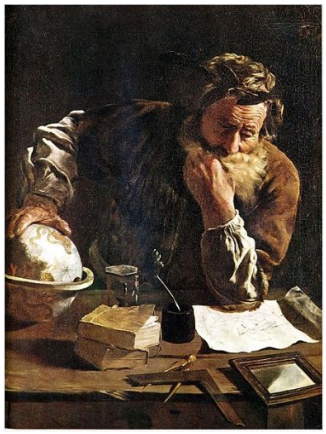
Whew! That makes the head spin! Let's let the math do our work:

```
df_rot_$r_ <- sqrt(df_rot_$x * df_rot_$x + df_rot_$y * df_rot_$y)

df_rot_$theta_ <- atan2(df_rot_$y,df_rot_$x)

plot(y=df_rot_$theta_, x=df_rot_$r_, type='p',col=df_rot_$class_, cex=df_rot_$class_*.
5, pch=filled_circle_, xlab='r', ylab='theta')

legend("topleft",legend=c("1","2"),fill=col_)
```



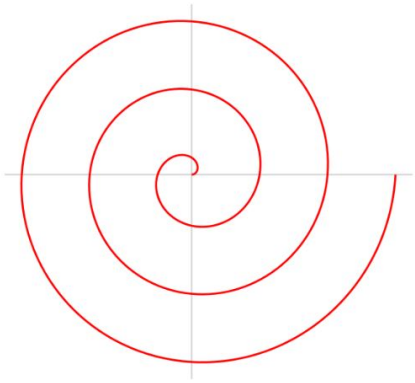
In {r, theta} space, the data that are spirals in {x, y} space, are linear. Now, here is an interesting fact about this plot. Each 2 * pi traunch of data is depicted in r-theta space as a linear association of points. That has a rather deep meaning; it means that the derivative dr/dtheta is constant.

This result is not a revelation. Archimedes observed this relationship in the 3rd-century BC when working with his collaborator, Pappus, but Archimedes reached back even further in the history of mathematical thought to attribute the spiral analytics to Conon of Samos.



Archimedes

Archimedes and Pappas were able to concisely describe the spiral curve with a simple linear equation in {r, theta}



$$r = a + b\theta$$

Now the fact(oid) that the spiral is within a simple transform of being **linear** is astonishing, and it puts an entire 2500-year old analytic toolbox in our hands with which to analyze the mathematical, physical, and statistical character of these data.

So having transformed the spirals into polar coordinates, we can read off the characteristic gradient and use it to separate the points.

There are 3 tasks in this POC

1. The two red points at {1.4, -3.0} must be raised dy = 2 * pi
2. The black points starting at {1.0, -3.0} must be raised dy = 2 * pi
3. The red points starting at {0.6, =3.0} must be raised by dy = 2 * pi

Let us focus on the two points at {1.4, -3.0}

There is a boundary between these two points and the black scatter line that begins at {1.0, -3.0}. If we were working on an industrial problem where an approximation would not do, we could perform the regression on Archimedes' Spiral; that is, we could perform a linear regression on those points to discover the gradient of that scatter plot, then translate it downward algebraically, and indeed that is what we would do in an IIoT Use Case, since such a use case would require accuracy and precision, but in this POC, we can approximate perfectly well for illustrative educational purposes.

The gradient dr/dtheta = m = delta_y/delta_x = (-1.25 + 3)/(1.4 - 1.2).

In this first disambiguation exercise, we want to add 2 * pi to the theta value of those two points. Let's use the point {1.2, -3.0} to write down the equation for the point-slope form of the line. Remember, the point-slope form is:

$$y - y_1 = m(x - x_1)$$

In this context then, we have

(y +3) = (-1.25 + 3)(x - 1.2)/(1.4 - 1.2).

That's it.

Remember, that is a characteristic slope of this Multiphysics and the slope of these scatter plots are inticately related to the "tightness" or "coil" of Archimedes spiral. It is a global characteristic of the Multiphysics of this data-generating function, so once we have it, we don't need to do that hard work any more for the duration of this IIoT project.

Let's calculate it explicitely right now:

```
m_ <- (-1.25 + 3)/(1.4 - 1.2)

paste('The slope is:', m_, sep=" ")
```

```
## [1] "The slope is: 8.75"
```

Ok, so the characteristic slope, which is intricately related to the dr/dtheta gradient of Archimedes spiral is 8.75, and that is a constant for the duration of this analysis. Indeed, we could use that value, and deviations from that value, do perform change-point detection on the telemetry streams to flag anomalies, which is how you couple machine learning with reliability engineering, but that is the topic for another day.

Now we simply solve for y as a function of x, and we are ready to shift those two red points at {1.5, -3 * pi} We have:

$$y = mx + y_1 - mx_1$$

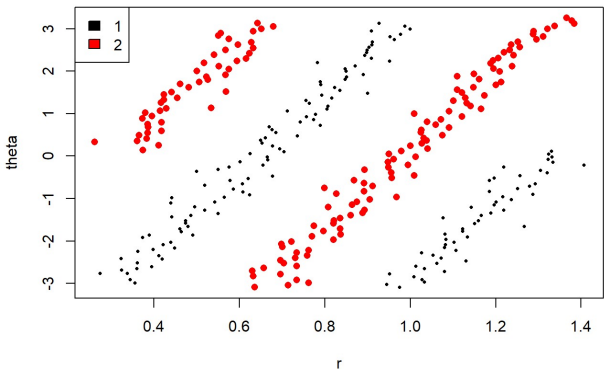
This lets us parameterize the problem so that we could programmatically clean up this transformed data traunch-by-traunch. Those two points that we are after {1.4, - 1 x pi}, so we can programmatically refer to them unambiguously as

“the two points that are under the line (y - y1) = m(x - x1) –> (y + 3) = 8.75(x - 1.2).”

The point-slope form of the line allows us to lazily observe a single charactistic point that is in the margin region between the black and red scatter lines, plug them into the point-slope equation, and lift the points by the requisite 2 * pi radians.

Here we go:

```
x1 <- 1.2
y1 <- -3.0


df_rot_$theta_ <- ifelse(df_rot_$class_ == 2 & df_rot_$r_ > x1 & df_rot_$theta_ < (m_
* df_rot_$r_  + y1 - m_ * x1),
                         df_rot_$theta_ + 2 * pi,
                         df_rot_$theta_)


plot(y=df_rot_$theta_, x=df_rot_$r_, type='p',col=df_rot_$class_, cex=df_rot_$class_*.
5,pch=filled_circle_, xlab='r', ylab='theta')

legend("topleft",legend=c("1","2"),fill=col_)
```

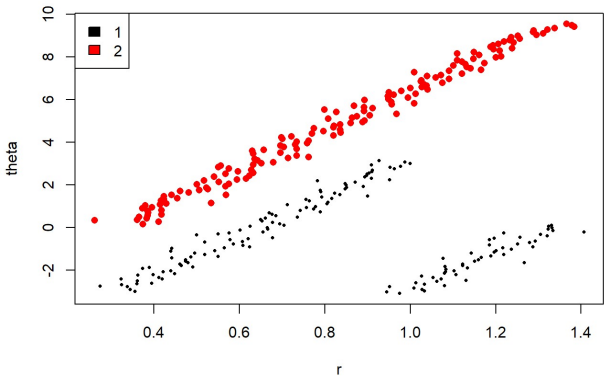


That worked brilliantly. We lifted the two outlying red points up to their mates on the lower red scatter line. Let’s continue working on the red dots and lift the lower scatter line up to their mates in the upper red scatter line.

We already have the slope, which is a characteristic constant of this system’s data-generating function’s Multiphysics remember, so all we need to do is read off a {x0,y0} point for the point-slope form of the linear margin.

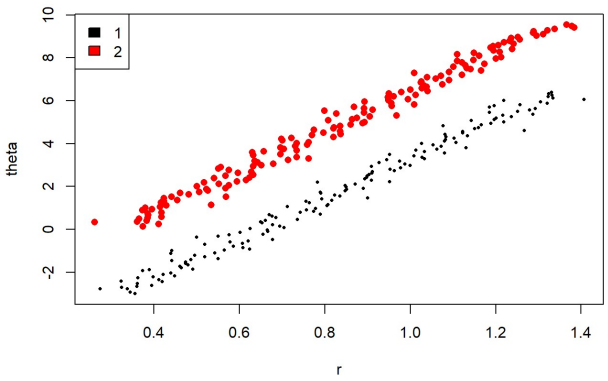
It looks like {0.45, -3} is a good point in the margin, so our line will be:

```
x1 <- 0.45
y1 <- -3.0


df_rot_$theta_ <- ifelse(df_rot_$class_ == 2 & df_rot_$r_ > x1 & df_rot_$theta_ < (m_
* df_rot_$r_  + y1 - m_ * x1),
                         df_rot_$theta_ + 2 * pi,
                         df_rot_$theta_)


plot(y=df_rot_$theta_, x=df_rot_$r_, type='p',col=df_rot_$class_, cex=df_rot_$class_*.
5, pch=filled_circle_, xlab='r', ylab='theta')

legend("topleft",legend=c("1","2"),fill=col_)
```



That worked perfectly. Let's do our work on the black scatter line at {1.0, -2.0}. Just eyeballing the margin suggests that maybe {0.5, -2.0} is clearly in the margin. Let's try it. Again, if this were the solution to an industrial-grade IIoT problem set, we would not guess; we would do the linear regression and do it right, but for this illustrative purpose, given the beautifully wide margins, we can eyeball it.

```
x0_ <- 0.7
y0_ <- -2.0

# Now any Class == 2 that is below the line y = 7x - 5.8 must be shifted up 2 * pi


df_rot_$theta_ <- ifelse(df_rot_$class_ == 1 & df_rot_$r_ > x0_ & df_rot_$theta_ < (m_
* df_rot_$r_ + y0_ - m_ * x0_),
                         df_rot_$theta_ + 2 * pi,
                         df_rot_$theta_)



plot(y=df_rot_$theta_, x=df_rot_$r_, type='p',col=df_rot_$class_, cex=df_rot_$class_*.
5, pch=filled_circle_, xlab='r', ylab='theta')
legend("topleft",legend=c("1","2"),fill=col_)
```



It is a beautiful thing. Now remember, these linear gymnastics notwithstanding, in an industrial solution, we would do a Design of Experiment (DOX) where these angles were captured in the complex plane and the angles would run from [zero, infinity) so that the figure above would be captured straight-away from the sensor telemetry, but for this illustrative purpose, we let the atan2 function do the reporting, so we had this nice educational opportunity to manhandle the scatter lines. Inhale the blessings and exhale the gratitude…

We did this work to illustrate how some elementary analysis, maths, and an old workhorse classifier, such as logistic regression, could perform against the crown jewel of optimization theory, the Support Vector Machine.

Now remember the respect that we have for Vapnik's Support Vector Machine. It really has no equal in non-linear classifiers, and indeed, to do our work here, our first move was to transform the Archimedes spirals to a linear form, so we are not equivacating; however, we want to emphasize the point that a seemingly intractable linear classification problem like Archimedes spiral does not require the crown jewel of optimization theory to draw a decision boundary. Rather, following Archimedes and Decartes, we can transform the dataset to its natural coordinates, exploit its linear character in that reference frame, and apply linear-classification techniques.

Let’s do it.

We have to suppress Warnings since we actually did a little too well and R’s logistic-regression algo will complain that we are overfit, but more on that another day. Here it is.

```
y_ <- df_rot_$class_

y_ <- y_ - 1

X_ <- as.data.frame(cbind('r'=df_rot_$r_, 'theta'=df_rot_$theta_))

glm_ <- suppressWarnings(glm(y_ ~ ., data=X_, family=binomial(link='logit')))

y_hat_ <- suppressWarnings(predict(glm_, newdata=X_, type='response'))

table(class=y_,classifier=y_hat_ >= 0.5)
```

```
##      classifier
## class FALSE TRUE
##     0   150    0
##     1     0  150
```

```
#       classifier
# class FALSE TRUE
#     0   150    0
#     1     0  150
```

Look at that! A perfect classification on the training set. I should hope so given the HUGE linear margin between the two scatter lines in the figure immediately above.

So there you have it. We have solved for a very non-linear classification decision boundary by applying some quantitative analysis and our old workhorse, the logistic regression. State-of-the-art optimizaton theory is tremendously valuable for certain problem sets, and it is an important investment for the aspiring data scientist to make, but the problem sets for which such powerful algorithms are required are often beyond the daily work flows of all but the most gifted scientists in the most cutting-edge research organizations.