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

Discovery of a Rectangular Decision Boundary by Various Machine-Learning Algorithms

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 25, 2020

Version 2020-01-25-0930-MAI

Abstract

This article compares and contrasts the ability of three machine-learning algorithms to discover a rectangular decision boundary in a supervised classification exercise; Support Vector Machines, Multivariate Logistic Regression with Causal Factors, and Recursive Trees are benchmarked against a hypothetical dataset of rectangular 2-Dimensional geometry.

Discussion

This article extends a previous article (https://4-vector.org/2020/01/22/2020-01-23-1800-mai/) on the analytical construction of a curvilinear classification decision boundary using Multivariate Logistic Regression. During a recent constructive LinkedIn discussion (https://www.linkedin.com/posts/davelanger_analytics-datasciece-machinelearning-activity-6626144928084877312-hgG-) regarding the utility of Recursive Trees and high-dimensional stochastic simulation with Trees, commonly termed “Random Forests,” it was observed that Trees and Forests can find a non-linear decision boundary with which to classify a class. One correspondent observed that this dexterity comes at the expense of multiple adjustable parameters and hyperparameters such that, notwithstanding the purported “interpretability” of Trees and Forests, these powerful classification algorithms returned very little domain knowledge of the problem set’s data-generating function; better to curate the Features of the Analytic Data Set (ADS) and submit the ADS to Regression analysis, which returns coupling constants and the standard error of the central tendency of those constants, which when considered in tandem, hint at the Multiphysics of the problem set.

The discussion then conjectured that classifying two classes that were separated by a hypothetical **rectangular** decision boundary would be difficult by regression but straight-forward by trees or forests.

All parties agreed with that conjecture's intuition.

This article simply documents this experiment and returns it to the LinkedIn conversation for further consideration and discussion.

Let's get started.

Analysis

We manually created a dataset with a rectangular decision boundary and saved it to a CSV file.

Ingest the data set and examine the structure

```
dir_ <- 'c:\\d\\rectangular\\data'
csv_ <- 'data.csv'

file_ <- paste(dir_,csv_,sep="\\")

df_ <- read.csv(file=file_, stringsAsFactors = FALSE)

str(df_)
```

```
## 'data.frame':    450 obs. of  4 variables:
##  $ Item : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ x    : num  -0.174 -1.548 -0.401 -1.859 -0.463 ...
##  $ y    : num  -0.5612 -1.4956 -0.1492 -0.7482 0.0522 ...
##  $ class: int  1 1 1 1 1 1 1 1 1 1 ...
```

Table the Regression ClassID.

```
table(df_$class)
```

```
##
##   1   2
## 150 300
```

There 150 points with Dataset ClassID = 1 and 300 points with Dataset ClassID = 2.

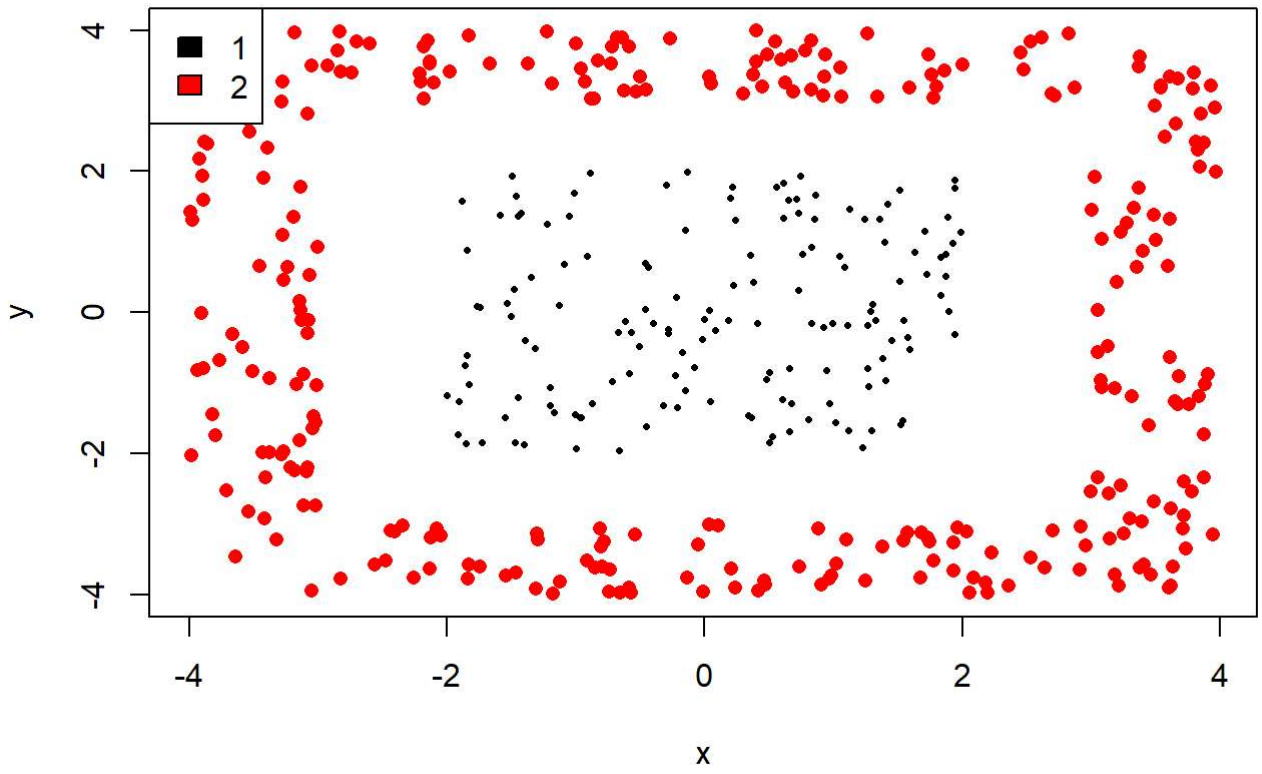
Note that the baseline Accuracy of the data is 300/450; that is, in the absense of any analysis at all, when faced with decision-under-uncertainty, the odds favor Datasaet ClassID = 2 by a 2:1 ratio of 300/150; odds of 2:1 are a probability

p(dataset class id = 2) = 300/(150 + 300) = 0.66667

to within one-tenth of one basis point.

Plot the data for an initial visualization.

```
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', y
lab='y')

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



In a prior article (https://4-vector.org/2020/01/22/2020-01-23-1800-mai/) we applied algebraic techniques to convert a curvilinear decision boundry to a linear one, and it is compelling to wish for similar #2-pencil-powered prestidigitation for this rectangular boundary, but indeed no such elementary transformation is available.

There are several information-theoretical optimization algorithms available.

- **Support Vector Machines (SVM)** employ the renowned kernel trick to lift the 2-Dimensional Cartesian data into a higher N-dimensional space where a [N-1]-dimensional hyperplace can demark a linear decision boundary for clasification
- **Recursive Trees (Trees)** apply an information-gain metric to learn a series of business rules that are applied programmatically to parse space in an arbitrarily-large number of rectangular regions
- **Random Forests (Forests)** use a set of parameters to tune a stochastic simulation of Recursive Trees to calculate an ensemble average of classification or scoring outcomes, which is interpreted as the optimal polling result
- **Logistic Regression (Logit)** computes a best multivariate linear regression line, then calculates a sigmoid function over the line to render a classification on the linear decision boundary

The purpose of this paper is two fold:

1. Confirm that SVM and Trees do indeed efficiently classify the class over the rectangular decision boundary, and
2. Investigete under what embellishments of the Analytic Data Set (ADS) the workhorse Logit can apply a linear decision boundary to classify the class over the rectangular decision boundary.

The Basline: Logistic Regression ***without*** Causal Factors or Synthetic Features

As a baseline metric with which to calculate the ROI of our analytic investment, perform a zeroth-order Logit as a classifier with no synthetic features in the ADS **X**. Load the class into the dependent variable and the Cartesian abscissa and ordinate into the ADS. In the R statistical language, Logit is invoked via the Generalized Linear Model that is linked to the binomial logit sigmoid. First, see how well Logit memorizes the training data by calculating the multivariate coupling constants, then immediately predicting directly back on the training data itself. As an accounting matter, the Dataset ClassID lables are reduced by integer 1 to facilitate tabling the Regression Class and classification to observe and/or compute Accuracy. In this text, we will refer to these values as the Regression Class 0 and 1 to eliminate confusion with the Dataset ClassIDs 1 and 2.

```
y_ <- df_$class

y_ <- y_ - 1

X_ <- as.data.frame(cbind('x'=df_$x, 'y'=df_$y))

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 TRUE
##     0  150
##     1  300
```

The Logit completely failed to classify Regression Class = 0 (= Dataset ClassID = 1). This is a complete fail to find a decision boundary.

It is instructive to examine the prediction, y_hat_, for this calculation. It is verbose, but let's look:

```
y_hat_
```

```
##         1         2         3         4         5         6         7         8
## 0.6671400 0.6706157 0.6679165 0.6717399 0.6681612 0.6619200 0.6683355 0.6626110
##         9        10        11        12        13        14        15        16
## 0.6658017 0.6624982 0.6686231 0.6680858 0.6689796 0.6717202 0.6702717 0.6668794
##        17        18        19        20        21        22        23        24
## 0.6716936 0.6627567 0.6631425 0.6665222 0.6706972 0.6647325 0.6703351 0.6653789
##        25        26        27        28        29        30        31        32
## 0.6667626 0.6631165 0.6656546 0.6700256 0.6627767 0.6656326 0.6653196 0.6668850
##        33        34        35        36        37        38        39        40
## 0.6653208 0.6683443 0.6615812 0.6702535 0.6636931 0.6638693 0.6634601 0.6632668
##        41        42        43        44        45        46        47        48
## 0.6649638 0.6690086 0.6668013 0.6627871 0.6651506 0.6624955 0.6715489 0.6632660
##        49        50        51        52        53        54        55        56
## 0.6722724 0.6666481 0.6663764 0.6629259 0.6632668 0.6636802 0.6707693 0.6697184
##        57        58        59        60        61        62        63        64
## 0.6649492 0.6661591 0.6685250 0.6650525 0.6675377 0.6656071 0.6698006 0.6620053
##        65        66        67        68        69        70        71        72
## 0.6668324 0.6662745 0.6644213 0.6681631 0.6719709 0.6647829 0.6632572 0.6620780
##        73        74        75        76        77        78        79        80
## 0.6675299 0.6633248 0.6688131 0.6648757 0.6713407 0.6618248 0.6618211 0.6623433
##        81        82        83        84        85        86        87        88
## 0.6659326 0.6713190 0.6641975 0.6715092 0.6671733 0.6625917 0.6650299 0.6642416
##        89        90        91        92        93        94        95        96
## 0.6668689 0.6645155 0.6647528 0.6639201 0.6716879 0.6644817 0.6620673 0.6712995
##        97        98        99       100       101       102       103       104
## 0.6717503 0.6659637 0.6715346 0.6654784 0.6695821 0.6691288 0.6679456 0.6685005
##       105       106       107       108       109       110       111       112
## 0.6663018 0.6622198 0.6682984 0.6696630 0.6675366 0.6640230 0.6704312 0.6634498
##       113       114       115       116       117       118       119       120
## 0.6618441 0.6717506 0.6642169 0.6622324 0.6651699 0.6705735 0.6617214 0.6701238
##       121       122       123       124       125       126       127       128
## 0.6620452 0.6654356 0.6709984 0.6626307 0.6702762 0.6620162 0.6669702 0.6710624
##       129       130       131       132       133       134       135       136
## 0.6667584 0.6630400 0.6689442 0.6621094 0.6679899 0.6613184 0.6640638 0.6626581
##       137       138       139       140       141       142       143       144
## 0.6713453 0.6640645 0.6647414 0.6709994 0.6672774 0.6726364 0.6683834 0.6717854
##       145       146       147       148       149       150       151       152
## 0.6676834 0.6700783 0.6675625 0.6711462 0.6635367 0.6700133 0.6546383 0.6565028
##       153       154       155       156       157       158       159       160
## 0.6562043 0.6573693 0.6583174 0.6569912 0.6580447 0.6560837 0.6590832 0.6553034
##       161       162       163       164       165       166       167       168
## 0.6564526 0.6576926 0.6555580 0.6575720 0.6568610 0.6553687 0.6558436 0.6568845
##       169       170       171       172       173       174       175       176
## 0.6551732 0.6566061 0.6578330 0.6589586 0.6571382 0.6560153 0.6572057 0.6558069
##       177       178       179       180       181       182       183       184
## 0.6574157 0.6586689 0.6553383 0.6575645 0.6569224 0.6568277 0.6580800 0.6581151
##       185       186       187       188       189       190       191       192
## 0.6562664 0.6581289 0.6577504 0.6561757 0.6579022 0.6574749 0.6566100 0.6571434
##       193       194       195       196       197       198       199       200
## 0.6564459 0.6570284 0.6569517 0.6565474 0.6565942 0.6555913 0.6586986 0.6567887
##       201       202       203       204       205       206       207       208
## 0.6574340 0.6551400 0.6553823 0.6579121 0.6575371 0.6580608 0.6576374 0.6585987
##       209       210       211       212       213       214       215       216
```

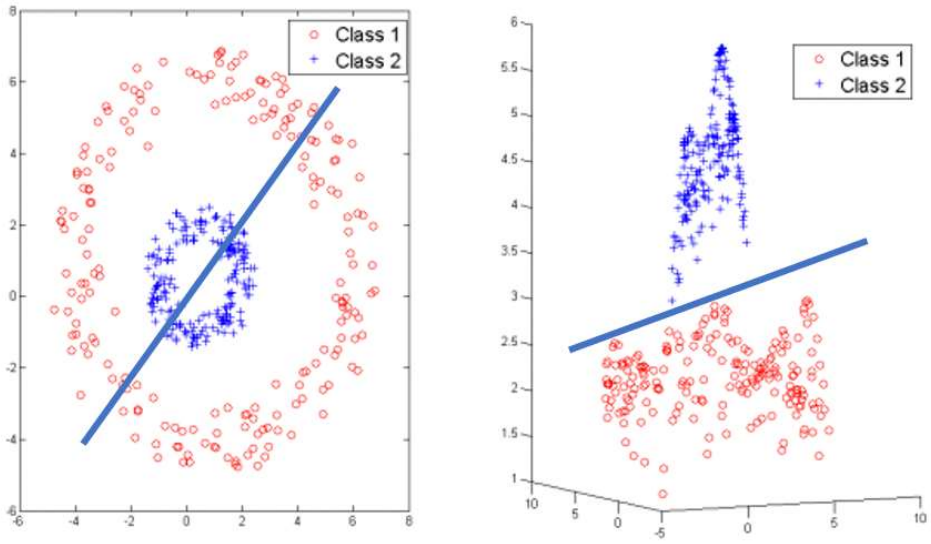
```
## 0.6580456 0.6560849 0.6582245 0.6553360 0.6555772 0.6580918 0.6579318 0.6561038
##       217       218       219       220       221       222       223       224
## 0.6570638 0.6557064 0.6553303 0.6577324 0.6575541 0.6553522 0.6570210 0.6556601
##       225       226       227       228       229       230       231       232
## 0.6581113 0.6649715 0.6704079 0.6692881 0.6706159 0.6724206 0.6670146 0.6726793
##       233       234       235       236       237       238       239       240
## 0.6672481 0.6610383 0.6627777 0.6581631 0.6658808 0.6600569 0.6680176 0.6575896
##       241       242       243       244       245       246       247       248
## 0.6695950 0.6706480 0.6601301 0.6642084 0.6713994 0.6613413 0.6608008 0.6585283
##       249       250       251       252       253       254       255       256
## 0.6679300 0.6618043 0.6716134 0.6627336 0.6647902 0.6632807 0.6642711 0.6626384
##       257       258       259       260       261       262       263       264
## 0.6654695 0.6733347 0.6654569 0.6698153 0.6715184 0.6656844 0.6686190 0.6593864
##       265       266       267       268       269       270       271       272
## 0.6573827 0.6619948 0.6672354 0.6641527 0.6722601 0.6635226 0.6678680 0.6658060
##       273       274       275       276       277       278       279       280
## 0.6678964 0.6596658 0.6677875 0.6724435 0.6593968 0.6725038 0.6592856 0.6573858
##       281       282       283       284       285       286       287       288
## 0.6670948 0.6686928 0.6717782 0.6681293 0.6690993 0.6596736 0.6602884 0.6714779
##       289       290       291       292       293       294       295       296
## 0.6629309 0.6581694 0.6675871 0.6613814 0.6608681 0.6588502 0.6606325 0.6693360
##       297       298       299       300       301       302       303       304
## 0.6602653 0.6675065 0.6627951 0.6608099 0.6753078 0.6755862 0.6758491 0.6766257
##       305       306       307       308       309       310       311       312
## 0.6775898 0.6781510 0.6780602 0.6744969 0.6770489 0.6752506 0.6752139 0.6770127
##       313       314       315       316       317       318       319       320
## 0.6784328 0.6784105 0.6777385 0.6769782 0.6782538 0.6784196 0.6787723 0.6754573
##       321       322       323       324       325       326       327       328
## 0.6745902 0.6773108 0.6760236 0.6781123 0.6748582 0.6762823 0.6748920 0.6754879
##       329       330       331       332       333       334       335       336
## 0.6751794 0.6765030 0.6755138 0.6756221 0.6776142 0.6779275 0.6760452 0.6751319
##       337       338       339       340       341       342       343       344
## 0.6783364 0.6762520 0.6746824 0.6761455 0.6755231 0.6782741 0.6745855 0.6771338
##       345       346       347       348       349       350       351       352
## 0.6782634 0.6738871 0.6752380 0.6762013 0.6783605 0.6763451 0.6754329 0.6768622
##       353       354       355       356       357       358       359       360
## 0.6770387 0.6742281 0.6754744 0.6756831 0.6749621 0.6775801 0.6748173 0.6746359
##       361       362       363       364       365       366       367       368
## 0.6747205 0.6779370 0.6756297 0.6765895 0.6752550 0.6775069 0.6784547 0.6770353
##       369       370       371       372       373       374       375       376
## 0.6748872 0.6774157 0.6769648 0.6780111 0.6761374 0.6767116 0.6768873 0.6692415
##       377       378       379       380       381       382       383       384
## 0.6735320 0.6604289 0.6667434 0.6741600 0.6611590 0.6662642 0.6666952 0.6733143
##       385       386       387       388       389       390       391       392
## 0.6671219 0.6654500 0.6653764 0.6761805 0.6642092 0.6670106 0.6599577 0.6629776
##       393       394       395       396       397       398       399       400
## 0.6660545 0.6703082 0.6625186 0.6655483 0.6760268 0.6701311 0.6741150 0.6753751
##       401       402       403       404       405       406       407       408
## 0.6679387 0.6662754 0.6694380 0.6633103 0.6712899 0.6647513 0.6664740 0.6756277
##       409       410       411       412       413       414       415       416
## 0.6629903 0.6704132 0.6740216 0.6689689 0.6609386 0.6635518 0.6701764 0.6742149
##       417       418       419       420       421       422       423       424
## 0.6698121 0.6603629 0.6756803 0.6739462 0.6740121 0.6613328 0.6716664 0.6670061
##       425       426       427       428       429       430       431       432
```

```
## 0.6701344 0.6738141 0.6707418 0.6628883 0.6758728 0.6741647 0.6700116 0.6678736
##       433       434       435       436       437       438       439       440
## 0.6649892 0.6659191 0.6656653 0.6719115 0.6696989 0.6703803 0.6694235 0.6659995
##       441       442       443       444       445       446       447       448
## 0.6668036 0.6705955 0.6671618 0.6651563 0.6709762 0.6603795 0.6760938 0.6631381
##       449       450
## 0.6611807 0.6727090
```

All 450 data points have a probability p(Regression Class = 1) = p(Dataset ClassID = 2) of ~ 0.67, which is the default baseline probability. The Logit simply could not get a hook into the dataset's Data Generating Function (DGF), and could not locate the decision boundary. When the data was tabled, all points were classified as TRUE, which was certainly correct for the 300 Regression Class = 1 (Dataset ClassID = 2) points, but represented a degradation of Accuracy, Precision, and Recall for the 150 Regression Class = 0 (Dataset ClassID = 1) points.

Support Vector Machine

We should experience no such trouble with the Support Vector Machine (SVM). The SVM is the crown jewel of optimization theory. We have discussed the algorithm here (https://4-vector.org/2020/01/22/2020-01-23-1800-mai/), so we will not repeat ourselves, save to remind the reader that the SVM's kernel trick serves to lift the data into an N-Dimensional space where an [N-1]-Dimensional hyperplane can effortlessly find a hyper-linear decision boundary with which to classify the class.



The SVM can be invoked in Base R via the **ksvm** function. We will employ the Gaussian Radial Basis Function kernel, which is an exponential, the Taylor expansion of which contains all powers of the abscissa and ordinate and which converges for all values of the same. At run time, the SVM creates all powers of x and y as synthetic features in the Analytic Data Set. The algorithm will certainly be able to draw the decision boundry's hyperplane to classify the class.

First calculate the SVM, then immediately predict the training data to observe how well the classifier memorizes the class.

```
y_ <- df_$class

y_ <- y_ - 1

X_ <- as.data.frame(cbind(df_$x, df_$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   150    0
##     1     0  300
```

As expected, The powerful SVM, with the Radial Basis kernel, perfectly memorized the class on the rectangular decision boundary. Query whether it can classify unseen data.

First we assign a random double-precision number to the ADS, then partition a 65/35 training/testing split. Then we calculate the ksvm on the training set and predict on the testing set before tabling the data and observing Accuracy, Precision, and Recall, which are functions of the False Positive and False Negative rates. We seed the pseudo-random number generator with the integer 137, which is the reciprocal of the quantum-mechanical fine-structure constant. It was also the author's minor-league baseball batting average.

```
n_ <- nrow(df_)

set.seed(137)

runif_ <- runif(n_, 0,1)

which_train_ <- which(runif_ <= 0.65)
which_test_ <- which(runif_ > 0.65)

y_ <- df_[which_train_,'class']

y_ <- y_ - 1

X_ <- as.data.frame(cbind(df_[which_train_,'x'], df_[which_train_,'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   100    0
##     1     0  192
```

Again, the SVM memorized the training data perfectly. Now let's see how well it can predict on the testing data.

```
y_ <- df_[which_test_,'class']

y_ <- y_ - 1


X_ <- as.data.frame(cbind(df_[which_test_,'x'], df_[which_test_,'y']))

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

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

```
##      classifier
## class FALSE TRUE
##     0    50    0
##     1     0  108
```

Well, there you have it! The powerful Support Vector Machine, with its famed Gaussian Radial Basis Function kernel, was able to perfectly classify the class for unseen, withheld test data over a complex rectangular decision boundary. Very, very impressive, indeed! It is important to note that the SVM has no domain knowledge whtasoever of the data-generating function of the problem set, which is impressive, indeed, but contrariwise, it can teach us nothing about the Multiphysics of the problem. If there is cause upstream from the effect, we have no hope of finding it with the Support Vector Machine. The blessing is the curse.

Recursive Trees

Let's move on to the Recursive Trees algo, which partitions rectangular space via an information-theoretic metric to score or classify.

In the R statistical language, Recursive Trees are exposed as optimized C++ classes in the recursive-partitioning **rpart** package. Rpart calculates the information gain at each node and scores or classifies the leaves. Rendering the decision tree itself to the R video layer requires the Recursive Partitioning Plot **rpart.plot** library.

The prediction resultant differes slightly from other classification algos that return the classifier's estimation that the probability of the class is 1; that is, to say that the Logit, for example, is 0.71125 is to say that the Logit sigmoid evaluates to p(Regression Class = 1) = p(Dataset ClassID = 2) = 71.125%. However, the CART recursive-partitioning algo returns the not only the proability that the Regression Class = 1, but also the probability that the Regression Class = 0. To evaluate the prediction and make a classification for our confusion matrix, we then must ask the prediction matrix which value is larger, and classify the Regression Class based on this business rule.

Because of the way that recursive trees work to partition rectangular space, we expect that they will fit a rectangular decision boundary perfectly, and indeed, we expect that it will have a perfect showing in this problem set, so as a gauge metric, let's take a quick detour and try to make the Trees fail. This way, we will have confidence that a strong classification via recursive trees on the rectangular problem set is a valid machine-learning effect and not some overfitting anomaly.

To fool Recursive Trees, let's introduce a curvy rather than rectilinear dataset.

Trees on Curvilinear Data

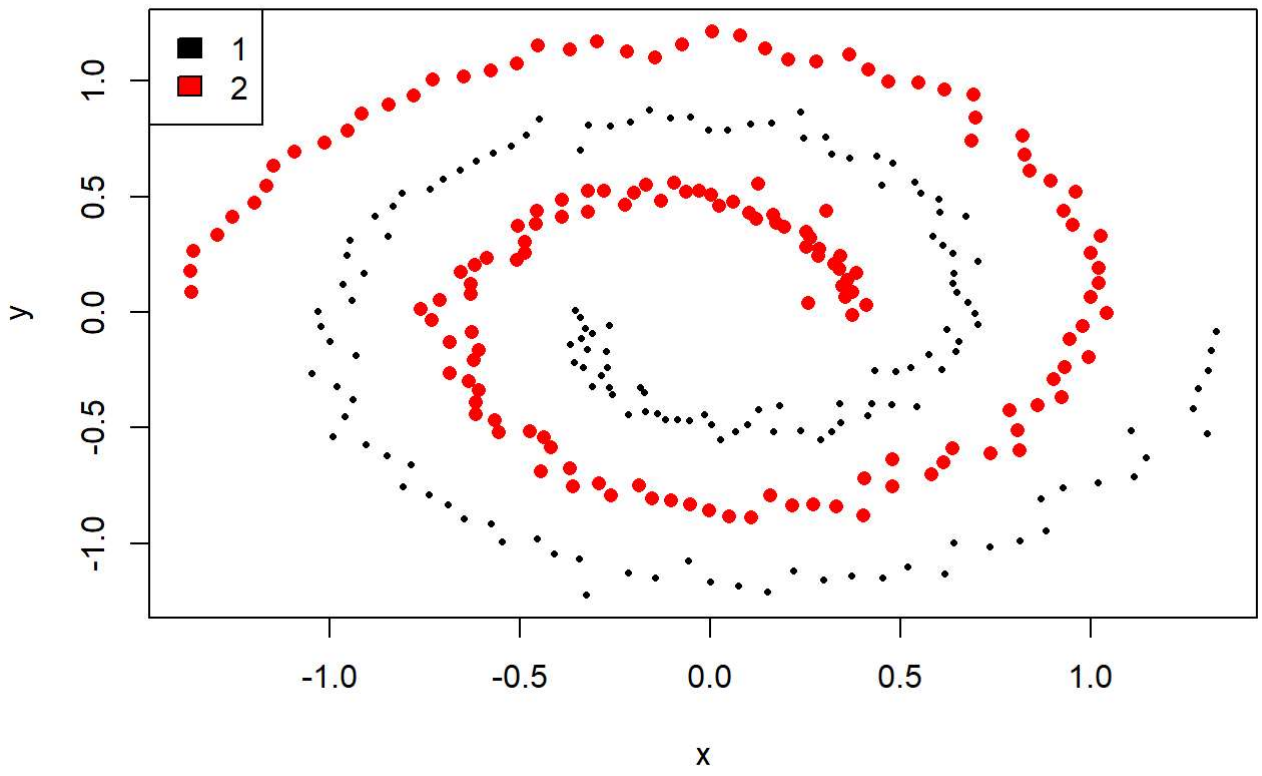
The kernlab **spirals** data set (https://cran.r-project.org/web/packages/kernlab/index.html) is curvilinear in nature, and our expectation is that spirals will completely frustrate Trees and Forests alike. Let's load spirals and levelset.

```
dir_spirals_ <- "c:\\d\\spirals\\write.csv"
csv_spirals_ <- "df_spirals.csv"
file_ <- paste(dir_spirals_, csv_spirals_,sep="\\")

df_spirals_ <- read.csv(file=file_,stringsAsFactors = FALSE)

str(df_spirals_)
```

```
## 'data.frame':    300 obs. of  4 variables:
##  $ item : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ 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: int  1 1 2 2 1 2 2 2 1 2 ...
```

We discussed the spirals dataframe at length in a previous writing (https://4-vector.org/2020/01/22/2020-01-23-1800-mai/), referenced *supra*. It comprises two interlaced Archimedes spirals, which are linear in polar coordinates and within a simple transformation of being linear in Cartesian space.

```
filled_circle_ = 16

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

plot(x=df_spirals_$x, y=df_spirals_$y, type='p',col=df_spirals_$class, cex=df_spirals_$class*.5
 ,pch=filled_circle_, xlab='x', ylab='y')

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



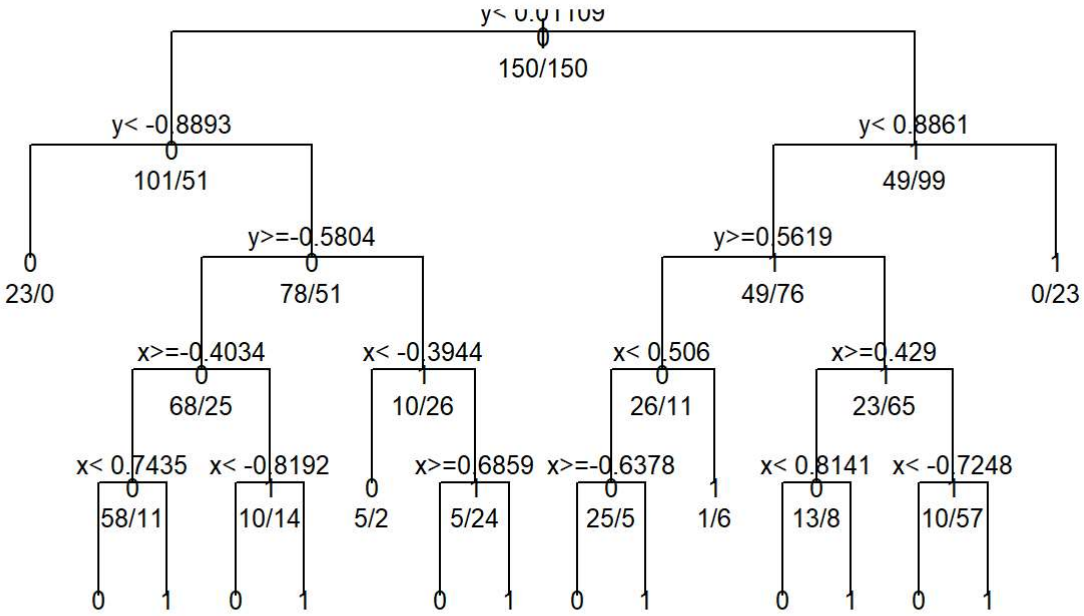
Our expectation is that Trees will have a Devil of a time finding this curvilinear decision boundary, which should be characterized by a flood of False Negative and False Positive classifications of the class. Remember that in our coding standards and Agile work program and road maps, we always begin by oberving how well the supervised method memorizes the dataset; that is, treating the dataset itself as a training set and predicting right back on itself, we observe how well the algo memorizes the labeled data. Let's see how Recursive Trees do in this regard.

```
# library(caret)
# library(rpart)
# library(rpart.plot)

y_ <- df_spirals_$class

y_ <- y_ - 1

X_ <- as.data.frame(cbind('x'=df_spirals_$x, 'y'=df_spirals_$y))

rpart_ <- rpart(y_ ~ .,data = X_, method = 'class')

predict_ <- predict(rpart_,newdata=X_)

n_ <- length(y_)

y_hat_ <- vector("numeric",n_)

y_hat_ <- apply(predict_,1,FUN=function(x){

  if(x[1] >= x[2]){

    0

  } else {

    1
  }

})

table(y_,y_hat_)
```

```
##    y_hat_
## y_    0   1
##   0 140  10
##   1  13 137
```

Frankly, this is absolutely astonishing! By parsing 2-Space, Trees correctly identified all but 23 of the memorized training data. This is especially impressive since the decision boundary is clearly curvilinear and Trees must parse space rectangularly. We can observe just how hard Trees had to work by plotting the Decision Tree itself.

```
plot(rpart_, uniform=TRUE)
text(rpart_, use.n=TRUE, all=TRUE, cex=.8)
```



This Decision Tree is certainly much more petite than it might have been, but careful study of it and the associated business rules would show that Trees is working very hard. Many proponents like trees for their interpretability, and that is true as far as it goes, but query whether you would like to make a decision under uncertainty with these fifty-five rules as your heuristics?

```
summary(rpart_)
```

```
## Call:
## rpart(formula = y_ ~ ., data = X_, method = "class")
##   n= 300
##
##           CP nsplit rel error    xerror       xstd
## 1 0.33333333      0 1.0000000 1.0933333 0.05748301
## 2 0.05333333      1 0.6666667 0.7266667 0.05553644
## 3 0.05000000      3 0.5600000 0.6733333 0.05456766
## 4 0.04666667      5 0.4600000 0.6733333 0.05456766
## 5 0.04333333      7 0.3666667 0.6466667 0.05401097
## 6 0.03333333      9 0.2800000 0.5266667 0.05085783
## 7 0.02000000     10 0.2466667 0.4600000 0.04859355
## 8 0.01333333     14 0.1666667 0.4666667 0.04883836
## 9 0.01000000     15 0.1533333 0.4533333 0.04834444
##
## Variable importance
##  x  y
## 56 44
##
## Node number 1: 300 observations,    complexity param=0.3333333
##   predicted class=0  expected loss=0.5  P(node) =1
##     class counts:   150   150
##    probabilities: 0.500 0.500
##   left son=2 (152 obs) right son=3 (148 obs)
##   Primary splits:
##       y < 0.01108812  to the left,  improve=16.669630, (0 missing)
##       x < -1.070137   to the right, improve= 4.639175, (0 missing)
##   Surrogate splits:
##       x < -0.4482403  to the right, agree=0.563, adj=0.115, (0 split)
##
## Node number 2: 152 observations,    complexity param=0.05333333
##   predicted class=0  expected loss=0.3355263  P(node) =0.5066667
##     class counts:   101    51
##    probabilities: 0.664 0.336
##   left son=4 (23 obs) right son=5 (129 obs)
##   Primary splits:
##       y < -0.889302   to the left,  improve=6.101897, (0 missing)
##       x < -0.7359095  to the left,  improve=3.471968, (0 missing)
##
## Node number 3: 148 observations,    complexity param=0.05
##   predicted class=1  expected loss=0.3310811  P(node) =0.4933333
##     class counts:    49    99
##    probabilities: 0.331 0.669
##   left son=6 (125 obs) right son=7 (23 obs)
##   Primary splits:
##       y < 0.8860759   to the left,  improve=5.970054, (0 missing)
##       x < 0.7604205   to the left,  improve=2.862878, (0 missing)
##
## Node number 4: 23 observations
##   predicted class=0  expected loss=0  P(node) =0.07666667
##     class counts:    23     0
##    probabilities: 1.000 0.000
##
```

```
## Node number 5: 129 observations,    complexity param=0.05333333
##   predicted class=0  expected loss=0.3953488  P(node) =0.43
##     class counts:    78    51
##    probabilities: 0.605 0.395
##   left son=10 (93 obs) right son=11 (36 obs)
##   Primary splits:
##       y < -0.5803962  to the right, improve=10.670830, (0 missing)
##       x < -0.7359095  to the left,  improve= 4.909201, (0 missing)
##
## Node number 6: 125 observations,    complexity param=0.05
##   predicted class=1  expected loss=0.392  P(node) =0.4166667
##     class counts:    49    76
##    probabilities: 0.392 0.608
##   left son=12 (37 obs) right son=13 (88 obs)
##   Primary splits:
##       y < 0.5618536   to the right, improve=10.147270, (0 missing)
##       x < 0.7604205   to the left,  improve= 4.079575, (0 missing)
##
## Node number 7: 23 observations
##   predicted class=1  expected loss=0  P(node) =0.07666667
##     class counts:     0    23
##    probabilities: 0.000 1.000
##
## Node number 10: 93 observations,    complexity param=0.04666667
##   predicted class=0  expected loss=0.2688172  P(node) =0.31
##     class counts:    68    25
##    probabilities: 0.731 0.269
##   left son=20 (69 obs) right son=21 (24 obs)
##   Primary splits:
##       x < -0.4034077  to the right, improve=6.3997190, (0 missing)
##       y < -0.2380596  to the left,  improve=0.4258065, (0 missing)
##   Surrogate splits:
##       y < -0.5318457  to the right, agree=0.753, adj=0.042, (0 split)
##
## Node number 11: 36 observations,    complexity param=0.02
##   predicted class=1  expected loss=0.2777778  P(node) =0.12
##     class counts:    10    26
##    probabilities: 0.278 0.722
##   left son=22 (7 obs) right son=23 (29 obs)
##   Primary splits:
##       x < -0.3943665  to the left,  improve=3.3114400, (0 missing)
##       y < -0.8105643  to the right, improve=0.8752137, (0 missing)
##
## Node number 12: 37 observations,    complexity param=0.03333333
##   predicted class=0  expected loss=0.2972973  P(node) =0.1233333
##     class counts:    26    11
##    probabilities: 0.703 0.297
##   left son=24 (30 obs) right son=25 (7 obs)
##   Primary splits:
##       x < 0.505958    to the left,  improve=5.411840, (0 missing)
##       y < 0.7877383   to the right, improve=1.074366, (0 missing)
##   Surrogate splits:
##       y < 0.5732131   to the right, agree=0.865, adj=0.286, (0 split)
##
```

```
## Node number 13: 88 observations,    complexity param=0.04333333
##   predicted class=1  expected loss=0.2613636  P(node) =0.2933333
##     class counts:    23    65
##    probabilities: 0.261 0.739
##   left son=26 (21 obs) right son=27 (67 obs)
##   Primary splits:
##       x < 0.4290291   to the right, improve=7.0575850, (0 missing)
##       y < 0.3323563   to the left,  improve=0.3561547, (0 missing)
##
## Node number 20: 69 observations,    complexity param=0.02
##   predicted class=0  expected loss=0.1594203  P(node) =0.23
##     class counts:    58    11
##    probabilities: 0.841 0.159
##   left son=40 (52 obs) right son=41 (17 obs)
##   Primary splits:
##       x < 0.7435409   to the left,  improve=8.2959210, (0 missing)
##       y < -0.06395763 to the left,  improve=0.8411143, (0 missing)
##
## Node number 21: 24 observations,    complexity param=0.04666667
##   predicted class=1  expected loss=0.4166667  P(node) =0.08
##     class counts:    10    14
##    probabilities: 0.417 0.583
##   left son=42 (10 obs) right son=43 (14 obs)
##   Primary splits:
##       x < -0.81921    to the left,  improve=11.6666700, (0 missing)
##       y < -0.2648876  to the right, improve= 0.2380952, (0 missing)
##   Surrogate splits:
##       y < -0.5286711  to the left,  agree=0.625, adj=0.1, (0 split)
##
## Node number 22: 7 observations
##   predicted class=0  expected loss=0.2857143  P(node) =0.02333333
##     class counts:     5     2
##    probabilities: 0.714 0.286
##
## Node number 23: 29 observations,    complexity param=0.02
##   predicted class=1  expected loss=0.1724138  P(node) =0.09666667
##     class counts:     5    24
##    probabilities: 0.172 0.828
##   left son=46 (7 obs) right son=47 (22 obs)
##   Primary splits:
##       x < 0.6858614   to the right, improve=5.4187190, (0 missing)
##       y < -0.8105643  to the right, improve=0.7758621, (0 missing)
##   Surrogate splits:
##       y < -0.6348078  to the right, agree=0.828, adj=0.286, (0 split)
##
## Node number 24: 30 observations,    complexity param=0.02
##   predicted class=0  expected loss=0.1666667  P(node) =0.1
##     class counts:    25     5
##    probabilities: 0.833 0.167
##   left son=48 (23 obs) right son=49 (7 obs)
##   Primary splits:
##       x < -0.6378051  to the right, improve=5.4761900, (0 missing)
##       y < 0.7877383   to the right, improve=0.3695324, (0 missing)
##   Surrogate splits:
```

```
##       y < 0.6408703   to the right, agree=0.867, adj=0.429, (0 split)
##
## Node number 25: 7 observations
##   predicted class=1  expected loss=0.1428571  P(node) =0.02333333
##     class counts:     1     6
##    probabilities: 0.143 0.857
##
## Node number 26: 21 observations,    complexity param=0.04333333
##   predicted class=0  expected loss=0.3809524  P(node) =0.07
##     class counts:    13     8
##    probabilities: 0.619 0.381
##   left son=52 (13 obs) right son=53 (8 obs)
##   Primary splits:
##       x < 0.8140997   to the left,  improve=9.9047620, (0 missing)
##       y < 0.3968352   to the right, improve=0.1904762, (0 missing)
##
## Node number 27: 67 observations,    complexity param=0.01333333
##   predicted class=1  expected loss=0.1492537  P(node) =0.2233333
##     class counts:    10    57
##    probabilities: 0.149 0.851
##   left son=54 (18 obs) right son=55 (49 obs)
##   Primary splits:
##       x < -0.7248075  to the left,  improve=8.1260360, (0 missing)
##       y < 0.1680251   to the left,  improve=0.1496963, (0 missing)
##
## Node number 40: 52 observations
##   predicted class=0  expected loss=0.01923077  P(node) =0.1733333
##     class counts:    51     1
##    probabilities: 0.981 0.019
##
## Node number 41: 17 observations
##   predicted class=1  expected loss=0.4117647  P(node) =0.05666667
##     class counts:     7    10
##    probabilities: 0.412 0.588
##
## Node number 42: 10 observations
##   predicted class=0  expected loss=0  P(node) =0.03333333
##     class counts:    10     0
##    probabilities: 1.000 0.000
##
## Node number 43: 14 observations
##   predicted class=1  expected loss=0  P(node) =0.04666667
##     class counts:     0    14
##    probabilities: 0.000 1.000
##
## Node number 46: 7 observations
##   predicted class=0  expected loss=0.2857143  P(node) =0.02333333
##     class counts:     5     2
##    probabilities: 0.714 0.286
##
## Node number 47: 22 observations
##   predicted class=1  expected loss=0  P(node) =0.07333333
##     class counts:     0    22
##    probabilities: 0.000 1.000
```

```
##
## Node number 48: 23 observations
##   predicted class=0  expected loss=0  P(node) =0.07666667
##     class counts:    23     0
##    probabilities: 1.000 0.000
##
## Node number 49: 7 observations
##   predicted class=1  expected loss=0.2857143  P(node) =0.02333333
##     class counts:     2     5
##    probabilities: 0.286 0.714
##
## Node number 52: 13 observations
##   predicted class=0  expected loss=0  P(node) =0.04333333
##     class counts:    13     0
##    probabilities: 1.000 0.000
##
## Node number 53: 8 observations
##   predicted class=1  expected loss=0  P(node) =0.02666667
##     class counts:     0     8
##    probabilities: 0.000 1.000
##
## Node number 54: 18 observations
##   predicted class=0  expected loss=0.4444444  P(node) =0.06
##     class counts:    10     8
##    probabilities: 0.556 0.444
##
## Node number 55: 49 observations
##   predicted class=1  expected loss=0  P(node) =0.1633333
##     class counts:     0    49
##    probabilities: 0.000 1.000
```

If that prospect makes you happy, then I am happy for you. Better you than me.

Notwithstanding the complexity of the decision heuristics, the ability of Trees to classify a curvilinear decision boundary with rectangular parsing is indeed impressive, at least when memorizing under the supervision of a labeled data set.

Let’s see whether this remarkable success continues when we ask Trees to predict heretofore unseen, withheld, testing data. Following our road map, parse the data into training and testing datasets on a 65/35 split and repeat our benchmark by predicting on the memorized training data once again.

```
  n_ <- nrow(df_spirals_)

  set.seed(137)

  runif_ <- runif(n_, 0,1)

  which_train_ <- which(runif_ <= 0.65)
  which_test_ <- which(runif_ > 0.65)

  y_ <- df_spirals_[which_train_,'class']

  y_ <- y_ - 1

  X_ <- as.data.frame(cbind(df_spirals_[which_train_,'x'], df_spirals_[which_train_,'y']))

  rpart_ <- rpart(y_ ~ .,data = X_, method = 'class')

  predict_ <- predict(rpart_,newdata=X_)

  n_ <- length(y_)

  y_hat_ <- vector("numeric",n_)

  y_hat_ <- apply(predict_,1,FUN=function(x){

    if(x[1] >= x[2]){

      0

    } else {

      1
    }

  })

  table_ <- table(y_,y_hat_)

  table_
```

```
##    y_hat_
## y_    0   1
##   0 102   3
##   1  22  71
```

```
  accuracy_ <- sum(diag(table_))/sum(table_)

  print(noquote(" "))
```

```
## [1]
```

```
  print(paste("The accuracy on the training set is:",accuracy_,sep=" "))
```

```
## [1] "The accuracy on the training set is: 0.873737373737374"
```

Trees is about 87% accurate on the memorized training data. Now predict on the unseen, withheld, testing data. If the Accuracy degrades, then these data are overfit and we must reconsider Trees as a robust classifier of this curvilinear data.

```
  y_ <- df_spirals_[which_test_,'class']

  y_ <- y_ - 1

  X_ <- as.data.frame(cbind(df_spirals_[which_test_,'x'], df_spirals_[which_test_,'y']))


  predict_ <- predict(rpart_,newdata=X_)

  n_ <- length(y_)

  y_hat_ <- vector("numeric",n_)

  y_hat_ <- apply(predict_,1,FUN=function(x){

    if(x[1] >= x[2]){

      0

    } else {

      1
    }

  })

  table_ <- table(y_,y_hat_)

  table_
```

```
##    y_hat_
## y_   0  1
##   0 40  5
##   1 20 37
```

```
  accuracy_ <- sum(diag(table_))/sum(table_)

  print(noquote(" "))
```

```
## [1]
```

```
print(paste("The accuracy on the testing set is:",accuracy_,sep=" "))
```

```
## [1] "The accuracy on the testing set is: 0.754901960784314"
```

The Accuracy did indeed degrade, but it is not a terribly drastic degradation. However, the reduction from 87% to 75% Accuracy does indicate that the Trees algorithm is struggling to make the classification. So Trees does not stand on all fours as a robust classifier over this curvilinear decision boundary, however, it is rather impressive that the algo does this well. I would not hesitate to include Trees in a stacked and boosted solution set; when paired with other classifiers, I should think that the stacked and boosted solution would benefit from Trees' vote in the ensemble polling.

But Trees' Accuracy did significantly degraded on the unseen, withheld, testing data, which suggests that its performance on the training data is overfit. Trees has a number of tunable parameters that can mitigate overfitting, but the tuning would move toward the testing Accuracy of 75% rather than toward the training Accuracy of 87%; because of way that Trees parses space, it has a very hard time honing in on the business rules around a curvilinear data generating function, and remember that those business rules are fifty-five in number, and do not necessarily scale to the general case.

HOWEVER, Trees' clumsiness with curvilinear data does not impugn its dexterity and robustness with **rectilinear** data! Remember, we hazed Trees immediately above exactly so that we would know Trees' success when we saw it. Let's let Trees shine now by pivoting back to our rectangular data set and let's watch Trees show off for us.

Trees on Rectangular Data

Ok, with this gauge metric in mind, let's let Trees shine. If it is true that Trees is challenged with a curvilinear decision boundary because it parses space rectangularly, perhaps it follows that it would excel with a rectangular decision boundary, and this indeed was the sentiment of the LinkedIn colleagues (https://www.linkedin.com/posts/davelanger_analytics-datasciece-machinelearning-activity-6626144928084877312-hgG-). Let's see whether Trees is up to the challenge.

Let's continue with our work plan to see how well the CART algo memorizes the class. Back to our rectangular data frame, **df_**. Following our work program, let's watch Trees memorize the data set in its entirety as training data.

```
y_ <- df_$class

y_ <- y_ - 1

X_ <- as.data.frame(cbind('x'=df_$x, 'y'=df_$y))

rpart_ <- rpart(y_ ~ .,data = X_, method = 'class')

predict_ <- predict(rpart_,newdata=X_)

n_ <- length(y_)

y_hat_ <- vector("numeric",n_)

y_hat_ <- apply(predict_,1,FUN=function(x){

  if(x[1] >= x[2]){

    0

  } else {

    1
  }

})

table(y_,y_hat_)
```

```
##    y_hat_
## y_    0   1
##   0 150   0
##   1   0 300
```

Wow! Now Trees is on its home turf! It memorized the rectangular data perfectly and drew a perfect rectangular decision boundary!

Following the work plan, let's now parse the data and see how Trees does over the rectangular boundary when predicting on heretofore unseen, withheld, testing data. We will parse training and testing data and watch Trees memorize the training data, then challenge it to predict the testing data. Here we go:

```
n_ <- nrow(df_)

set.seed(137)

runif_ <- runif(n_, 0,1)

which_train_ <- which(runif_ <= 0.65)
which_test_ <- which(runif_ > 0.65)


y_ <- df_[which_train_,'class']

y_ <- y_ - 1

X_ <- as.data.frame(cbind('x'=df_[which_train_,'x'], 'y'=df_[which_train_,'y']))

rpart_ <- rpart(y_ ~ .,data = X_, method = 'class')

predict_ <- predict(rpart_,newdata=X_)

n_ <- length(y_)

y_hat_ <- vector("numeric",n_)

y_hat_ <- apply(predict_,1,FUN=function(x){

  if(x[1] >= x[2]){

    0

  } else {

    1
  }

})

table_ <- table(y_,y_hat_)

table_
```

```
##    y_hat_
## y_    0   1
##   0 100   0
##   1   0 192
```

```
accuracy_ <- sum(diag(table_))/sum(table_)

print(noquote(" "))
```

```
## [1]
```

```
print(paste("The accuracy on the training set is:",accuracy_,sep=" "))
```

```
## [1] "The accuracy on the training set is: 1"
```

One again, Trees perfectly memorizes the training data, this time on the bifurcated training data subset.

Note that there are **292 observations in the training data.** We will see that number parsed piecemeal leaf-by-leaf in the business rules **infra**.

Let's predict on the heretofore, unseen, withheld, testing data.

```
y_ <- df_[which_test_,'class']

y_ <- y_ - 1

X_ <- as.data.frame(cbind('x'=df_[which_test_,'x'], 'y'=df_[which_test_,'y']))


predict_ <- predict(rpart_,newdata=X_)

n_ <- length(y_)

y_hat_ <- vector("numeric",n_)

y_hat_ <- apply(predict_,1,FUN=function(x){

  if(x[1] >= x[2]){

    0

  } else {

    1
  }

})

table_ <- table(y_,y_hat_)

table_
```

```
##    y_hat_
## y_    0   1
##   0  50   0
##   1   0 108
```

```
accuracy_ <- sum(diag(table_))/sum(table_)

print(noquote(" "))
```

```
## [1]
```

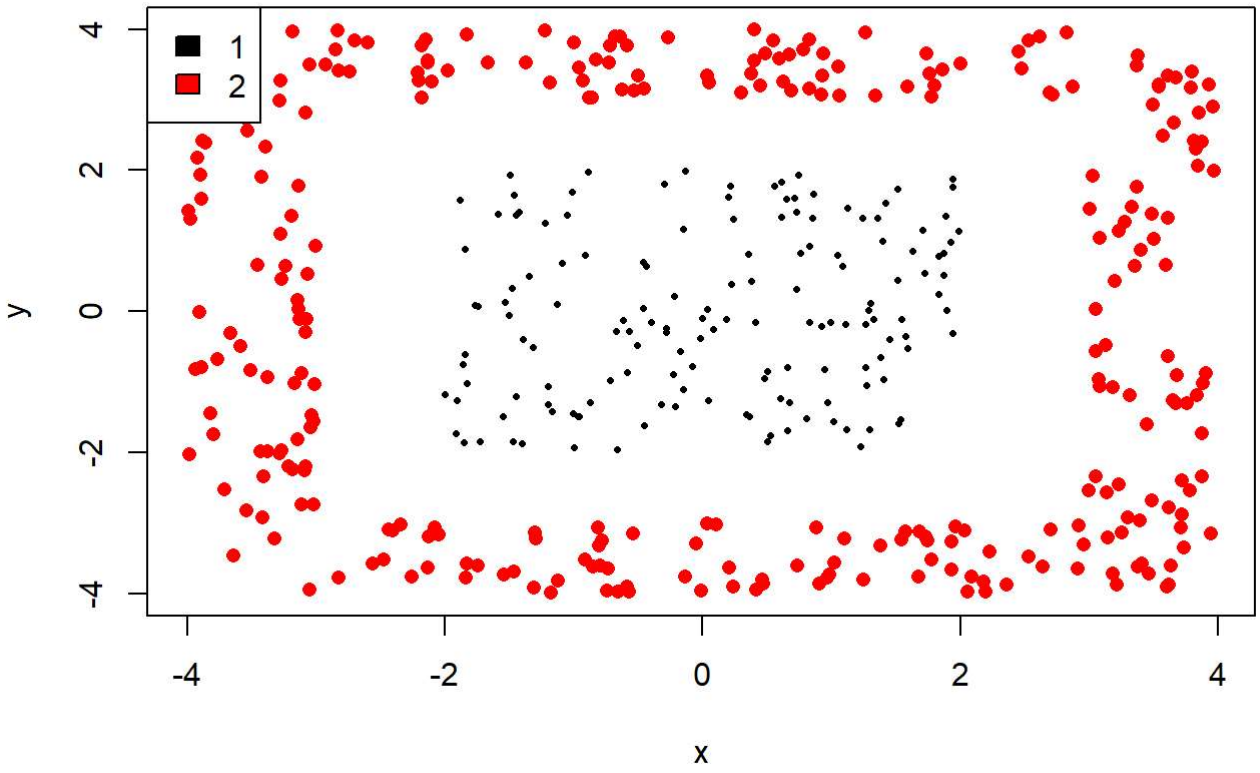
```
  print(paste("The accuracy on the training set is:",accuracy_,sep=" "))
```

```
## [1] "The accuracy on the training set is: 1"
```

Wow! Trees had perfect performance on the testing data! Clearly Recursive Trees is a powerful algorithm for discerning a rectangular decision boundary in a binary classification problem set.

Query to what degree Trees are interpretable on such a decision boundary. Remember, the business rules were challenging above when Trees had to rectangularly parse curvilinear data, but Trees is on its home turf now with a rectangular boundary. Let's view a summary of the rpart object. First, let's plot the data again right here so that we can reference it as we marvel at Trees dexterity with this problem set.

```
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', y
lab='y')

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



Here is the summary of the rpart recursive-partitioning object. Look at this report as the business rules that Trees has learned as it calculated the information gain at each node. Note that Node number 1 begins with the 292 observations that we observed in the training data on which the rpart object was calculated.

```
summary(rpart_)
```

```
## Call:
## rpart(formula = y_ ~ ., data = X_, method = "class")
##   n= 292
##
##      CP nsplit rel error xerror        xstd
## 1 0.285      0      1.00   1.00 0.081088485
## 2 0.240      2      0.43   0.45 0.061696895
## 3 0.190      3      0.19   0.21 0.044147170
## 4 0.010      4      0.00   0.01 0.009982862
##
## Variable importance
##  y  x
## 53 47
##
## Node number 1: 292 observations,    complexity param=0.285
##   predicted class=1  expected loss=0.3424658  P(node) =1
##     class counts:   100   192
##    probabilities: 0.342 0.658
##   left son=2 (216 obs) right son=3 (76 obs)
##   Primary splits:
##       y < -1.988228  to the right, improve=24.09944, (0 missing)
##       x < -1.911714  to the right, improve=19.22615, (0 missing)
##
## Node number 2: 216 observations,    complexity param=0.285
##   predicted class=1  expected loss=0.462963  P(node) =0.739726
##     class counts:   100   116
##    probabilities: 0.463 0.537
##   left son=4 (143 obs) right son=5 (73 obs)
##   Primary splits:
##       y < 1.999106   to the left,  improve=47.26755, (0 missing)
##       x < -1.911714  to the right, improve=25.05447, (0 missing)
##   Surrogate splits:
##       x < 3.345415   to the left,  agree=0.685, adj=0.068, (0 split)
##
## Node number 3: 76 observations
##   predicted class=1  expected loss=0  P(node) =0.260274
##     class counts:     0    76
##    probabilities: 0.000 1.000
##
## Node number 4: 143 observations,    complexity param=0.24
##   predicted class=0  expected loss=0.3006993  P(node) =0.489726
##     class counts:   100    43
##    probabilities: 0.699 0.301
##   left son=8 (119 obs) right son=9 (24 obs)
##   Primary splits:
##       x < -2.424606  to the right, improve=28.207090, (0 missing)
##       y < -0.2684122 to the right, improve= 1.678322, (0 missing)
##
## Node number 5: 73 observations
##   predicted class=1  expected loss=0  P(node) =0.25
##     class counts:     0    73
##    probabilities: 0.000 1.000
##
```

```
## Node number 8: 119 observations,    complexity param=0.19
##   predicted class=0  expected loss=0.1596639  P(node) =0.4075342
##     class counts:   100    19
##    probabilities: 0.840 0.160
##   left son=16 (100 obs) right son=17 (19 obs)
##   Primary splits:
##       x < 2.475763   to the left,  improve=31.9327700, (0 missing)
##       y < -0.4372953 to the right, improve= 0.8594871, (0 missing)
##
## Node number 9: 24 observations
##   predicted class=1  expected loss=0  P(node) =0.08219178
##     class counts:     0    24
##    probabilities: 0.000 1.000
##
## Node number 16: 100 observations
##   predicted class=0  expected loss=0  P(node) =0.3424658
##     class counts:   100     0
##    probabilities: 1.000 0.000
##
## Node number 17: 19 observations
##   predicted class=1  expected loss=0  P(node) =0.06506849
##     class counts:     0    19
##    probabilities: 0.000 1.000
```

Note 17 rules this time for the rectangular data in contrast to the 55 rules for the spiral data, above. The business rules are involved, but not intractable. We can get a good overall impression of rules if we examine the Tree itself. The top node recites the running total of the unassigned Regression classes (100 Regression Class 0's and 192 Regression Class 1's at the outset, as can be observed in the training data's confusion matrix) and asks the conditional, "Is the training class ordinate greater than or equal to -1.988?" If **not** then assign the point to Regression Class 1 for now. In the first leaf, 76 points that are below the line y = -1.988 are classified as red Regression Class 1 (= Dataset ClassID 2. We subtracted 1, remember, for computational ease within the algos.)

In the second node, Trees, using information gain on the condition y < 1.999, classifies a further 73 points that are above the line as red Regression Class 1 (= Dataset ClassID 2.), for now. Consult the plot above to convince yourself that is correct.

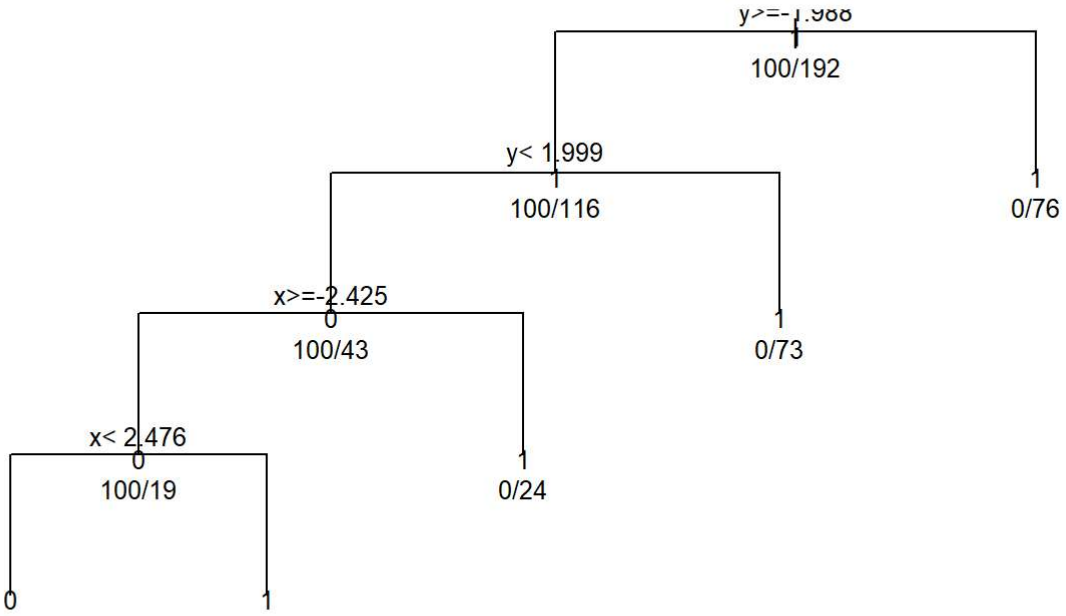
In the third node, Trees make a decision on the rule x >= -2.425. If no, then the points are correcly classified as red Regression Class 1 (= Dataset ClassID 2). So Trees is saying, "of the points that are above y = -1.988 and below y = 1.999, those that are left of x = 2.425 are red," and indeed that is correct.

Finally, Trees interrogates the data at x < 2.476. Now, remember, it has already classified almost all of the red dots. This last node correctly classifies the final 19 red Regression Class 1 dots, which leaves the 100 black Regression Class 0 dots.

The remaining 100 dots that have survived these buisness rules are correctly classified as black Regression Class 0 (= Dataaset ClassID 1).

Trees has performed beautifully on this rectangular decision boundary.

```
plot(rpart_, uniform=TRUE)
text(rpart_, use.n=TRUE, all=TRUE, cex=.8)
```



There is a lot going on in this object; the business rules are absolutely straight-forward, and to be sure, they are involved, but if you read the business rules and study the tree while referencing the plot of of the dataset, above, you will convince yourself that Trees was able to apply information gain perfectly to discover the rectangular decision boundary and classify the class.

Random Forests

Given the success of Recursive Trees on this rectangular decision boundary, we will not needlessly torture the matter with Random Forests, which are effectively a hyperparametric-tuned, bootstrapped, cross-validated, stochastic simulation of many Recursive Trees to calculate an ensemble average of Node and Leaf assignments over many iterations of the Trees algo, all while applying tunable parameters for cross validation, information-gain metric, bootstrapping, and voting.

Random Forest directly applies the wisdom of Galton's crowds, whereby Galton observed that the measure of central tendency of votes from a large crowd will be more accurate than the vote of any given member. By repeatedly recalculating the Recursive Tree with cross-validated random draws of rows and bootlegged random selections of features, and averaging the classifications or scores, the algo will average outcomes via a voting or polling methodology (many use the term "amalamalgamate" the outcomes), which will be more stable and accurate than any given Tree's prediction.

Many analysts prefer Random Forrests and see their role as business- and quantitative-analysts as parameter and hyperparameter tuning to extract the best performance from the Forests. Others see this exercise as anything but fundimental to the Multiphysics of the probelem set and do not consider hyperparameter tuning as valuable to the

science of data. No number of hyperparamaters amounts to even a modicum of understanding of the maths or physics of the problem set's data-generating function. And while machine-learning technicians debate this religious issue, no scientist is confused on this matter.

And for good reason. The great mathmatician, Freeman Dyson, discusses his conversation with Enrico Fermi about adjustable parameters in the absense of strong axiomatic mathematics or a compelling physical theory. Dyson's model of the strong interaction had four adjustable parameters with which to reproduce the experimental findings of early quantum electrodynamic effects. In an instant, Fermi mooted Dyson's entire research program for the pseudo-scalar theory of pions which purported to explain the strong interaction, and indeed, Fermi was correct; Gelmann's *ab initio* two-quark theory eventually explained the affect without the need for parametric prestidigitation.

So while we respect base Trees and Forests for their preternatural ability to parse rectangular space, we are very distrustful of tunable parameters. Whenever you find yourself tuning parameters and hyperparameters, you are nothing but a puppet master who is manipulating von Neumann's elephant, and the laughter you hear is Fermi's ghost as your audience.

So what to do? If the SVM can learn any decision boundary but gives no business insight in the process, if Trees & Forests return business rules but may require parametric and hyperparametric tuning, where can we turn if we wish to fit a difficult decision boundary **and**, receive statistical advice on our classsification, and have the opportunity to learn something of our problem set's data-generating function?

The secret lies in the nature of the SVM's Gaussian Radial Basis kernel and using this powerful idea to craft synthetic features and apply business analysis to identify causal factors of the Data Generating Function for the problem set.

Recall that the Radial Basis Function kernel is an exponential function; it is a Gaussian, after all.

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

Recall from your studies of the fluxions and fluents that the series expansion of an exponential has all powers of the coordinate and converges for all values of the coordinate.

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

So when the Radial Basis kernel does its work of lifting the data into a high-dimension space, it is going all the way! As it turns out, for many well-behaved decision boundaries, such an infinite expansion is hardly necessary. If that is indeed the case, then we have a way forward; we can engineer synthetic features in the Analytic Data Set (ADS), and in the process learn something about the Multiphysics of the data-generating function for the problem set.

Multivariate Logistic Regression with Causal Factors

Let's use this idea to see whether we can embellish our ADS to help the Logit make a good, interpretable classification of our class on our rectangular decision boundary. Recall at the top of this article that the Logit failed miserably on the data itself. Following Taylor, let's add the quadratic terms of each Cartesian x-y pair, including the cross term.

It turns out that this humble exercise is well-grounded in functional kernel theory. The quadratic terms, with cross terms, comprise the Homogeneous Polynomial kernel with parameter d = 2. See Nonlinear Classification here (https://en.wikipedia.org/wiki/Support-vector_machine).

R has a function to to this work, but let's do it manually to watch it work.

```
df_$x2 <- df_$x * df_$x

df_$y2 <- df_$y * df_$y

df_$xy <- df_$x * df_$y

str(df_)
```

```
## 'data.frame':    450 obs. of  7 variables:
##  $ Item : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ x    : num  -0.174 -1.548 -0.401 -1.859 -0.463 ...
##  $ y    : num  -0.5612 -1.4956 -0.1492 -0.7482 0.0522 ...
##  $ class: int  1 1 1 1 1 1 1 1 1 1 ...
##  $ x2   : num  0.0304 2.3962 0.1608 3.4558 0.2147 ...
##  $ y2   : num  0.31498 2.23696 0.02226 0.55978 0.00272 ...
##  $ xy   : num  0.0979 2.3152 0.0598 1.3909 -0.0242 ...
```

So we now have the linear terms and the second-order terms for each datum. Let's see whether the Logit can find a decision boundary on for this dataset where it was unable to do so before, **supra**. Folowing our work program, let's see how well the Logit memorizes the data under the supervision of a label.

```
y_ <- df_$class

y_ <- y_ - 1

X_ <- as.data.frame(cbind('x'=df_$x, 'y'=df_$y, 'x2'=df_$x2, 'y2'=df_$y2, 'xy'=df_$xy))

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  300
```

Wow! There you have it. We did not need all powers of x and y via the Radial Basis kernel to lift the data and slip a hyperplane between the classes. Simple quadratic terms from the Homogeneous Polynomial kernel of Degree 2 did the trick, at least on the memorized dataset itself. Let's follow our work program to see whether we are overfit. Remember, we get a hint of that by observing the degradation of Accuracy when predicting on heretofore unseen, withheld test data. We will partition an 65/35 split and predict first on the memorized training set followed by the testing set.

```
set.seed(137)

n_ <- nrow(df_)

set.seed(137)

runif_ <- runif(n_, 0,1)

which_train_ <- which(runif_ <= 0.65)
which_test_ <- which(runif_ > 0.65)


y_ <- df_[which_train_,'class']

y_ <- y_ - 1

X_ <- as.data.frame(cbind('x'=df_[which_train_,'x'],
                          'y'=df_[which_train_,'y'],
                          'x2'=df_[which_train_,'x2'],
                          'y2'=df_[which_train_,'y2'],
                          'xy'=df_[which_train_,'xy']))

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   100    0
##     1     0  192
```

The Logit memorized the training data, as expected, but are we overfit? Now for the ultimate test. Do the quadratic features provide sufficient dimensionality that the Logit can predict on heretofore unseen, withheld, testing data?

```
y_ <- df_[which_test_,'class']

y_ <- y_ - 1

X_ <- as.data.frame(cbind('x'=df_[which_test_,'x'],
                          'y'=df_[which_test_,'y'],
                          'x2'=df_[which_test_,'x2'],
                          'y2'=df_[which_test_,'y2'],
                          'xy'=df_[which_test_,'xy']))


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

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

```
##      classifier
## class FALSE TRUE
##     0    50    0
##     1     0  108
```

Fantastic! The quadratic terms provide all the dimensionality we need for Logit to discover this rectangular decision boundary. But have we learned anything about the Multiphysics? Unfortunately not.

```
summary(glm_)
```

```
##
## Call:
## glm(formula = y_ ~ ., family = binomial(link = "logit"), data = X_)
##
## Deviance Residuals:
##        Min          1Q      Median          3Q         Max
## -6.936e-05  -2.100e-08   2.100e-08   2.100e-08   5.990e-05
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -93.9670 30817.0754  -0.003    0.998
## x              -0.4995  4465.0423   0.000    1.000
## y              -0.4904  4057.3086   0.000    1.000
## x2             12.4307  3735.4655   0.003    0.997
## y2             12.4081  3750.3879   0.003    0.997
## xy             -4.0563  3665.4883  -0.001    0.999
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 3.7531e+02  on 291  degrees of freedom
## Residual deviance: 2.1135e-08  on 286  degrees of freedom
## AIC: 12
##
## Number of Fisher Scoring iterations: 25
```

None of the odds are statistically significant and the standard error of the estimate completely subsumes the estimate itself. Logit found the boundary, so the machine-learning technician's work is done here, but the data **scientist's** work is only beginning.

The scientific question is, "What data-generating function could/would beget the dataset? What process in nature draws a rectangular decision boundary?" Remember Fermi, we must have a compelling economic or physical theory or a strong axiomatic mathematical foundation before our work is done, so while the machine-learning technician is celebrating the confusion matrix that shows the perfect classification, the data scientist is still at work.

Closing Thoughts

Of course, in this hypothetical instance, there is not a DGF to be found exactly because this data was manufactured for this hypothetical. In that sense, Trees' business rules are indeed the correct insight. Nevertheless, Regression's contribution is absolutely integral to the scientific prosecution of the hypothesis; to

suspect that something is true based on superior classifications of SVM, Trees, or Forests is very, very different than **knowing** that something is true based on the negative result of Regression that is supported by statistical advice.