TP1
Computing the Shattering Coefficient

CAO Anh Quan
M2 D&K Statistical Learning

December 12, 2018

Contents

1 Region-to-class assignment (RTCA) algorithm **1**
1.1 Introduction 1
1.2 Steps 2
1.3 Comparison the performance with the code in class 3
1.3.1 Single Hyperplane, R^2 3
1.3.2 Single Hyperplane, R^3 4
1.3.3 Two Hyperplanes, Three classes, R^2 4

2 Compute the Shattering Coefficient for the architecture **5**
2.1 Result 8
2.1.1 Hidden layer 1 (R^7, 9 hyperplanes, 2 classes) 8
2.1.2 Hidden layer 2 (R^9, 5 hyperplanes, 2 classes) 9
2.1.3 Output layer 2 (R^5, 3 hyperplanes, 2 classes) 10
2.2 Compute the architecture Shattering Coefficient by taking the sum of all shattering coefficient of the hidden and output layers 10

1 Region-to-class assignment (RTCA) algorithm

1.1 Introduction

In this lab, I improved the algorithm implemented in the lecture. The algorithm in the class has a problem that when it randomly generate the hyperplanes, and it uses the hyperplanes to divide the space into regions, it can only generate one classification (map each region to one class). Even without any algorithm, with two classes, we can always find at least two classifications by reversing the labels. Another problem of the algorithm in class is that it can only work with one hyperplane and binary classification. Therefore, I made a new algorithm which can work with multiple hyperplanes and an arbitrary number of classes by finding every possible classification given a set of regions and named it Region-to-class assignment (RTCA) algorithm.

1.2 Steps

1. Generate the data samples
2. Generate all the hyperplanes randomly
3. The hyperplanes divide the points into many regions. We find all the regions contain the points and assign an index to them.
4. Use the RTCA algorithm to find all possible mapping from regions to classes. For example: If we have regions: 1, 2, 3 and classes: 1, 2. One possible mapping is:
   - region 1 → class 1
   - region 2 → class 2
   - region 3 → class 1
5. Each mapping is one classification. The shattering coefficient is the number of classifications.

The most interesting part is how to find all possible mappings from regions to classes. Thanks to this step, my algorithm will estimate the shattering coefficient more accurate because we can find multiple classifications instead of only one classification from each time we generate the hyperplanes. Here is how I implement it.

```
# generate all possible classifications by finding all mappings from region to class
gen.maps <- function (region.index, num.classes, region.strings) {

    # map each class to new index, so we will not be overlap when replace
    classes = 1:num.classes + num.classes

    if (length(region.index) == 0) {
        # there is no region left
        return (region.strings)
    }

    # select the first region
    x = region.index[1]

    # Generate all possible mapping of this region to all classes
    new.region.strings = NULL
    for (s in region.strings) {
        for (i in classes) {
            new.s = gsub(x, i, s)
            new.region.strings = c(new.region.strings, new.s)
        }
    }

    # recursive call for all regions left
    return (gen.maps(region.index[region.index != x], num.classes, new.region.strings))
}
```

```
27
gen.all.classifications <- function (region.index, num.classes, region.string)
    {
29      return (gen.maps(region.index, num.classes, c(region.string)))
}
```

1.3 Comparison the performance with the code in class

1.3.1 Single Hyperplane, R^2

```
# Compare with the code in the lecture
# 1 hyperplane, R^2
shatterings = estimate.shattering(num.hyperplanes = 1, n.start = 1, n.end =
    15, k = 4, num.classes = 2, R = 2)
in.class.shatterings = read.table(file = 'shattering_R2_single_hyperplane.dat'
    , header = FALSE)
plot(shatterings, xlab = "Number of points", ylab = "Shattering coefficient")
points(in.class.shatterings, col = "red")
```

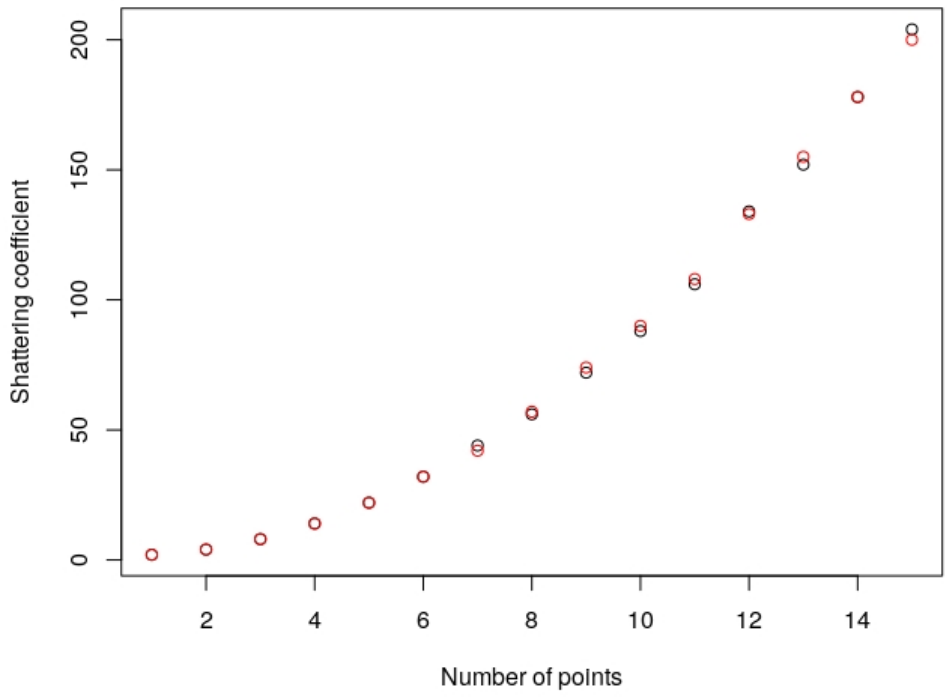

Figure 1: Shattering Coefficient computed by two algorithms.The black points is the result of the RTCA algorithm and the red point is the result of the algorithm in the class.

We can see that the RTCA algorithm seems to be able to find more classifications than the algorithm in the class.

1.3.2 Single Hyperplane, R^3

```
# 1 hyperplane, R^3
shatterings = estimate.shattering(num.hyperplanes = 1, n.start = 1, n.end =
    15, k = 4, num.classes = 2, R = 3)
in.class.shatterings = read.table(file = 'shattering_R3_single_hyperplane.dat'
    , header = FALSE)
plot(shatterings, xlab = "Number of points", ylab = "Shattering coefficient")
points(in.class.shatterings, col = "red")
```

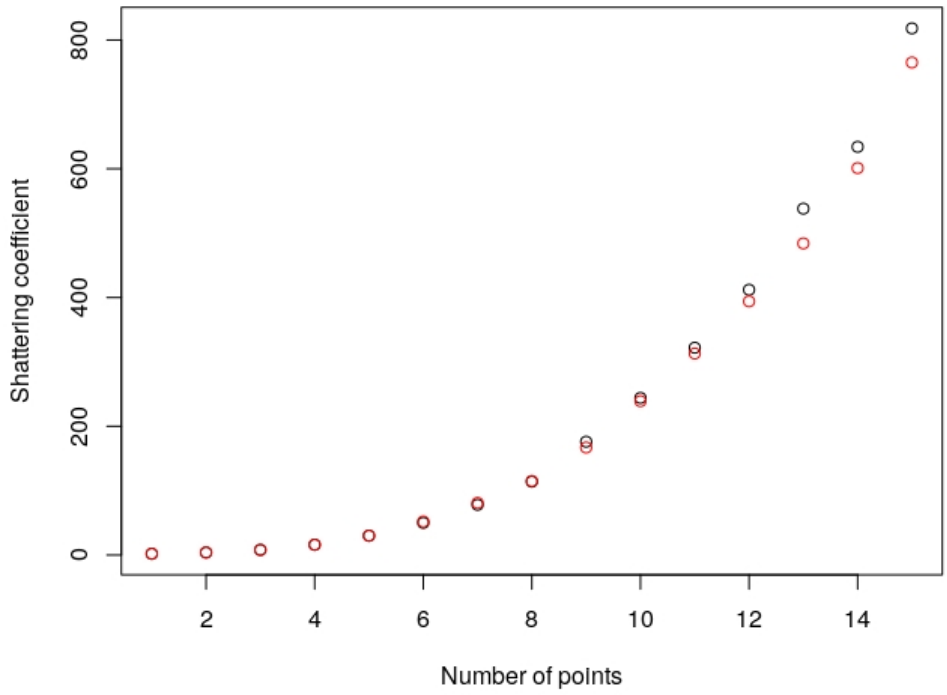

Figure 2: Shattering Coefficient computed by two algorithms.The black points is the result of the RTCA algorithm and the red point is the result of the algorithm in the class.

According to the figure, we can see clearly that the RTCA algorithm finds more classifications than the algorithm in the class with the number of points are more than 10.

1.3.3 Two Hyperplanes, Three classes, R^2

```r
# 2 hyperplanes, 3 classes , R^2
shatterings = estimate.shattering(num.hyperplanes = 2, n.start = 1, n.end =
    15, k = 4, num.classes = 3, R = 2)
plot(shatterings, xlab = "Number of points", ylab = "Shattering coefficient")
```

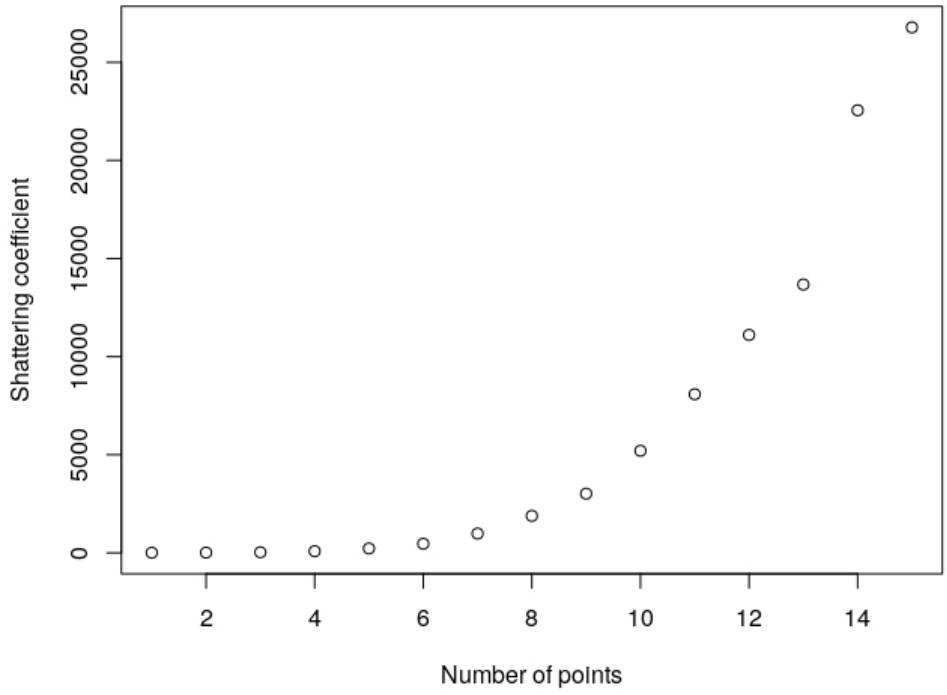

Figure 3: Shattering Coefficient for Two Hyperplanes, Three classes, R^2

2 Compute the Shattering Coefficient for the architecture

To estimate the Shattering Coefficient for the architecture, I follow the steps below:

1. Estimate the shattering coefficient for each layer.
    (a) Estimate the Shattering Coefficient for each neuron (hyperplane) in the layer (1 hyperplane, two classes, the dimension is the number of the neuron of the previous layer).
        i. Estimate the shattering coefficient for the number of observations from 1 to 20.

```
gn = 20
estimate.shattering(num.hyperplanes = 1, R = R, n.start = 1, n.
    end = gn, k = k)

```

ii. Fit a regression model on it.

iii. Use the model to compute the Shattering Coefficient for arbitrary number of observations.

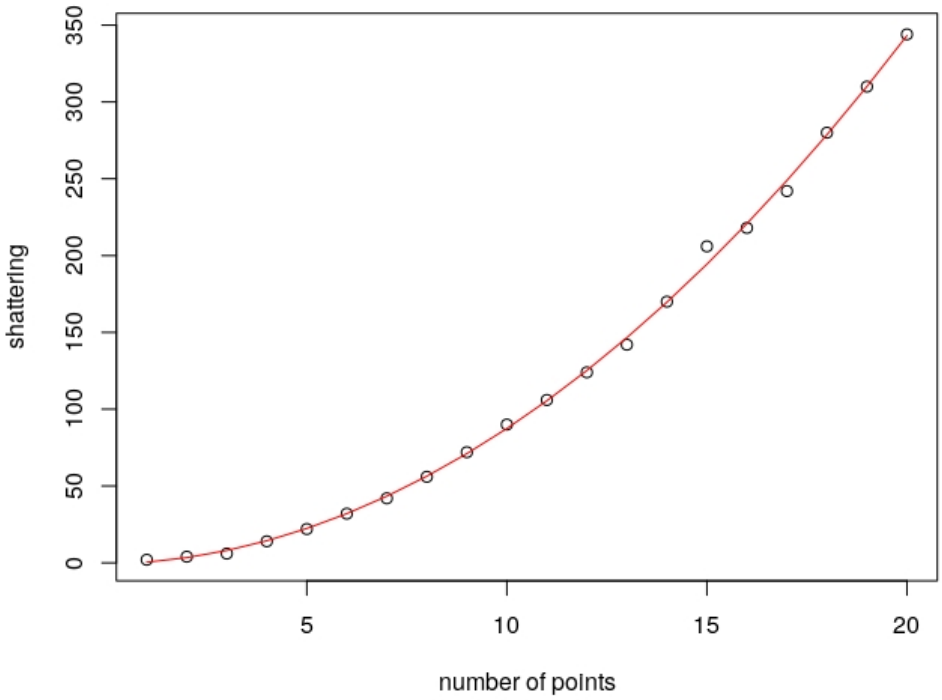

Figure 4: Example fitting a line on Shattering Coefficient

```
fit.regression.one.neuron <- function (gn = 20, R = 2, k = 3) {
    shattering.one.neuron = estimate.shattering(num.hyperplanes = 1,
   R = R, n.start = 1, n.end = gn, k = k)
    x = shattering.one.neuron[, 1]
    y = shattering.one.neuron[, 2]
    model = nls(y~ a*x^2 + b * x +c, start=list(a = 1,b = 1, c = 1))

    plot(x, y, xlab = 'number of points', ylab = "shattering")

    return (model)
}
```

```
# regression model for hidden layer 1
model1 = fit.regression.one.neuron(gn = 20, R = 7, k = 3, a = 12, b =
    0.1)

# regression model for hidden layer 2
model2 = fit.regression.one.neuron(gn = 20, R = 9, k = 3, a = 10, b =
    0.3)

# regression model for output layer 2
model3 = fit.regression.one.neuron(gn = 20, R = 5, k = 3, a = 10, b =
    0.3)

```

(b) Predict the Shattering Coefficient for n observations using the model above. Then, we compute the layer's Shattering Coefficient by taking the product of all Shattering Coefficient of each neuron in the layer.

```
estimate.shattering.nnet <- function(n, num.hyperplanes = 2, k=3, R
    =2, model) {
    gn = 20

    # get the shattering for one neuron
    a = data.frame(x = n)
    shattering = predict(model, a)

    # compute the shattering for all neurons
    shattering.all.neurons = shattering ^ num.hyperplanes

    return (shattering.all.neurons)
}
```

2. Sum all Shattering Coefficient to get the architecture Shattering Coefficient

```
estimate.architecture.shattering <- function (n, model1, model2, model3)
    {
    # compute the shattering coefficient for each layer

    hidden.layer1.shattering = estimate.shattering.nnet(n, num.
    hyperplanes = 9, k=3, R=7, model = model1)
    hidden.layer2.shattering = estimate.shattering.nnet(n, num.
    hyperplanes = 5, k=3, R=9, model = model2)
    output.layer.shattering = estimate.shattering.nnet(n, num.hyperplanes
    = 3, k=3, R=5, model = model3)
    return (hidden.layer1.shattering + hidden.layer2.shattering + output.
    layer.shattering)
}
```

2.1 Result

2.1.1 Hidden layer 1 (R^7, 9 hyperplanes, 2 classes)

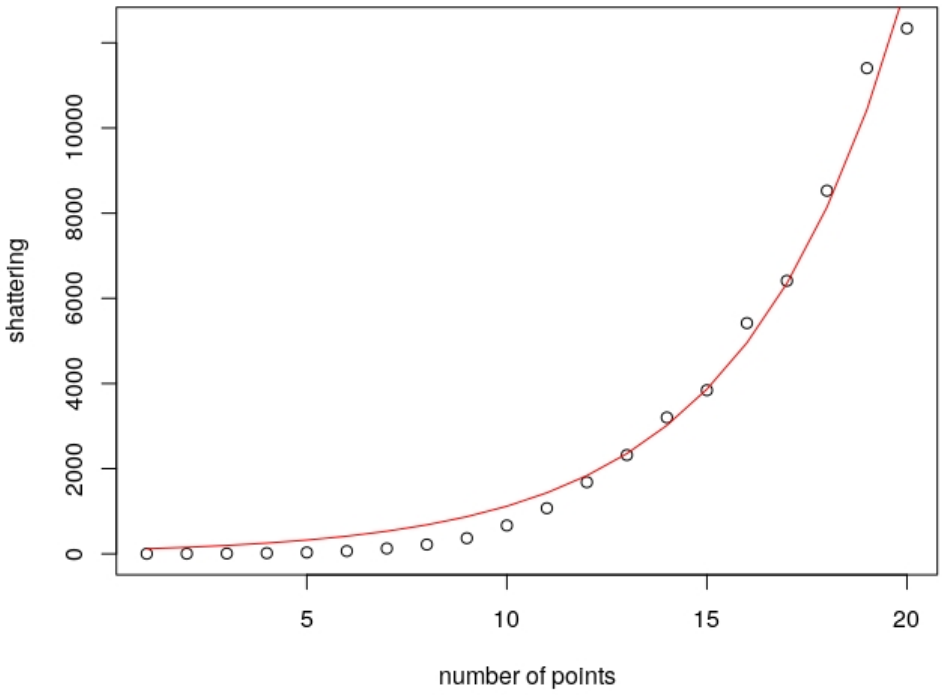

Figure 5: Fitting Regression on the Hidden layer 1

2.1.2 Hidden layer 2 (R^9, 5 hyperplanes, 2 classes)

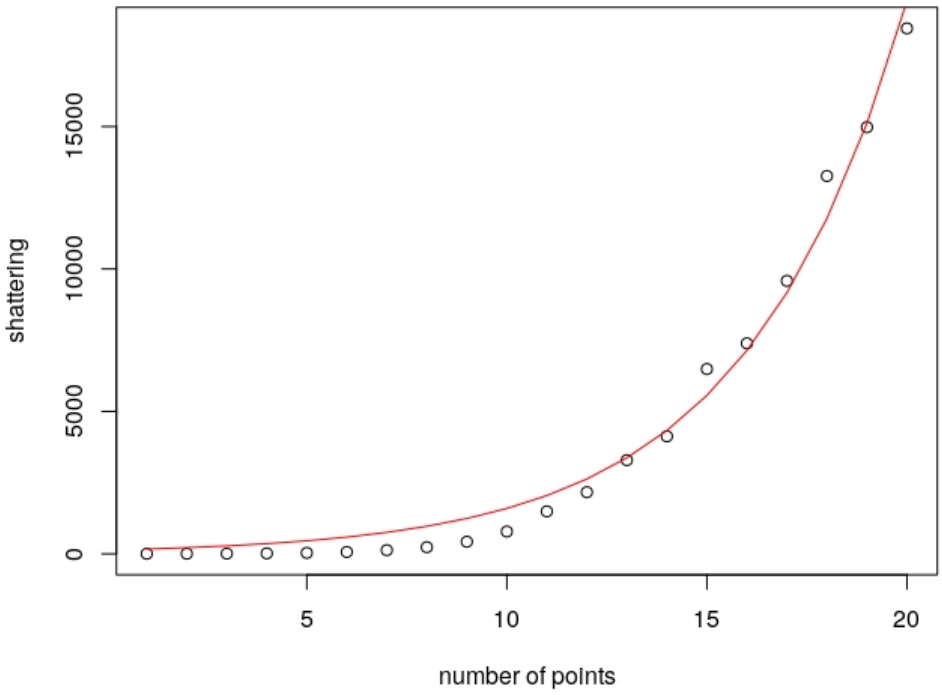

Figure 6: Fitting Regression on the Hidden layer 2

2.1.3 Output layer 2 (R^5, 3 hyperplanes, 2 classes)

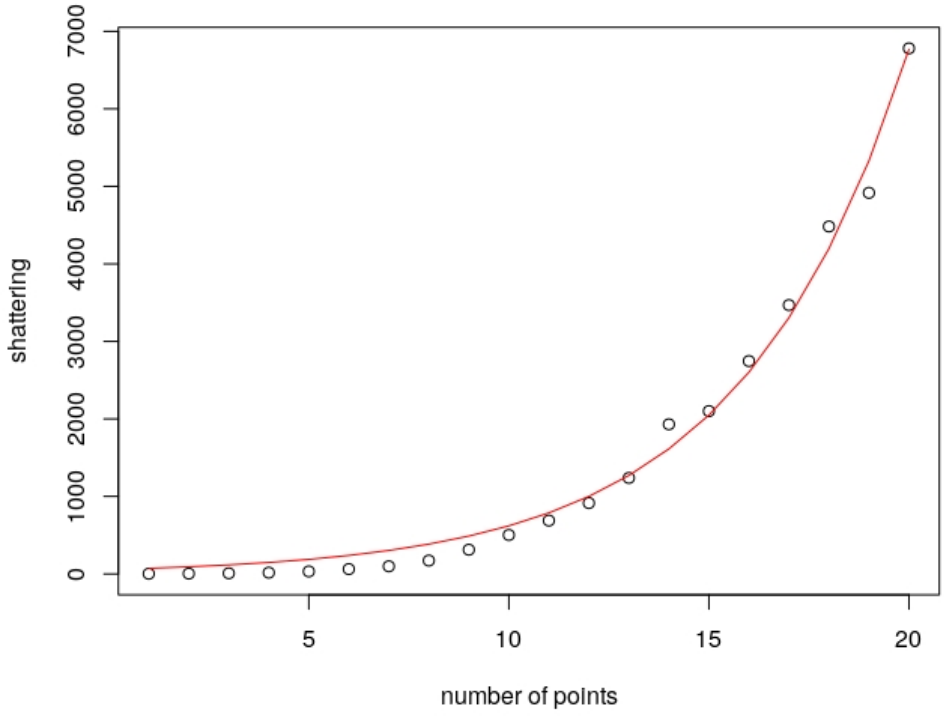

Figure 7: Fitting Regression on the Output layer 2

2.2 Compute the architecture Shattering Coefficient by taking the sum of all shattering coefficient of the hidden and output layers

I use the algorithm to compute the shattering coefficient from 1 to 30 observations. Shattering coefficient for the architecture with number of points from 1 to 30

```
1                [,1]          [,2]
  shattering      1  2.408309e+19
3 shattering      2  2.038419e+20
  shattering      3  1.725340e+21
5 shattering      4  1.460346e+22
  shattering      5  1.236053e+23
7 shattering      6  1.046208e+24
  shattering      7  8.855220e+24
9 shattering      8  7.495153e+25
  shattering      9  6.343979e+26
11 shattering    10  5.369612e+27
  shattering     11  4.544897e+28
13 shattering    12  3.846850e+29
  shattering     13  3.256015e+30
```

```
shattering    14 2.755927e+31
shattering    15 2.332646e+32
shattering    16 1.974377e+33
shattering    17 1.671134e+34
shattering    18 1.414466e+35
shattering    19 1.197219e+36
shattering    20 1.013339e+37
shattering    21 8.577010e+37
shattering    22 7.259674e+38
shattering    23 6.144666e+39
shattering    24 5.200912e+40
shattering    25 4.402108e+41
shattering    26 3.725991e+42
shattering    27 3.153719e+43
shattering    28 2.669342e+44
shattering    29 2.259360e+45
shattering    30 1.912347e+46
```

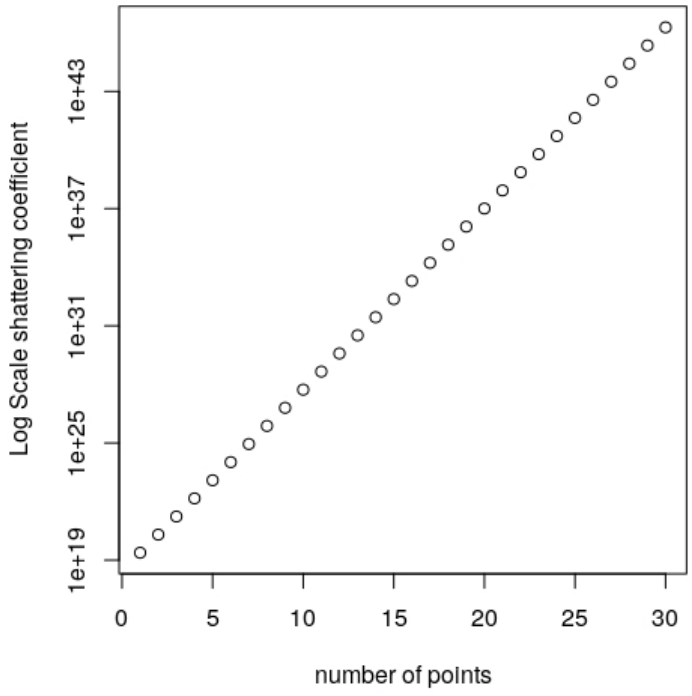

Figure 8: **Log Scale** plot of the shattering coefficient of the architecture. We can see from the figure that, the Shattering Coefficient increases exponential with the number of points.

```
# regression model for hidden layer 1
model1 = fit.regression.one.neuron(gn = 20, R = 7, k = 3, a = 12, b = 0.1)
```

```
3
# regression model for hidden layer 2
5 model2 = fit.regression.one.neuron(gn = 20, R = 9, k = 3, a = 10, b = 0.3)

7 # regression model for output layer 2
model3 = fit.regression.one.neuron(gn = 20, R = 5, k = 3, a = 10, b = 0.3)
9
shatterings = NULL
11 n = 30
for (i in 1:n) {
13     shattering = estimate.architecture.shattering(i, model1, model2, model3)
       shatterings = rbind(shatterings, shattering)
15 }
shatterings = cbind(1:n, shatterings)
17 x = shatterings[, 1]
y = shatterings[, 2]
19 # plot the shatterings with log scale
plot(x, y, xlab = "number of points", ylab = "Log Scale shattering coefficient
    ", log = 'y')
21 print(shatterings)
```