DETERMINATION OF STABILITY OF POLY DISPERSE EMULSIONS USING MACHINE LEARNING

A thesis submitted in fulfillment of the requirements

for the degree of

BACHELOR OF TECHNOLOGY

MASTER OF TECHNOLOGY

by

Pavithra Sivakumar

EE13B128

Guide: Prof. Raghunathan Rengaswamy

Co-guide: Prof. Kaushik Mitra

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNLOGY MADRAS

MAY 2018

CERTIFICATE

This is to certify that the thesis titled “DETERMINATION OF STABILITY OF POLYDISPERSE SYSTEMS USING MACHINE LEARNING” submitted by Pavithra Sivakumar to the Indian Institute of Technology Madras, Chennai for the award of the degree of Dual Degree, is a bonafide record of research work done by her under my supervision. The contents of this thesis, in full or a part has not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Raghunathan Rengaswamy

Project Guide

Professor

Department of Chemical Engineering

Indian Institute of Technology Madras

Place: Chennai

Date: 15/May/2018

Acknowledgments

Firstly, I would like to convey my warm regards and deepest gratitude to my guide, Professor, Dr. Raghunathan Rengaswamy, for giving me an opportunity to work under him. It is his endless patience, immense knowledge, constant guidance and timely advice that helped me and motivated me during the two semesters of Dual Degree project.

I would also like to thank Dr. Danny Raj, for his valuable guidance and suggestions throughout the course of the work done for this project. I would extend my sincere thanks to my parents for their constant support and encouragement to pursue my interests. I would also like to thank my friends and seniors for their support and guidance.

Abstract

Concentrated emulsions can be synthesized in a 2D micro channel with ease. Stability of these emulsions are critical from the point of view of design of micro channels for applications like incubation. These emulsions can undergo spontaneous destabilization through a series of coalescence avalanches. In poly-disperse emulsions, droplets can be arranged in different configurations, which changes the number of neighbors for every droplet, the orientation of droplet pairs etc. The local configuration of the droplets affects the propagation of coalescence events in the system which either augments or diminishes the propagation of avalanches. Hence, it would be important to identify those configurations that are stable and those that are not.

An interesting question would be: Can one comment on the stability of poly- disperse emulsions by simply observing an assembly of droplets? Say, by analyzing the snapshot of droplets in an experiment. To answer this question, we introduce a machine learning algorithm where we build what is called a classifier.

The idea here is to use the droplet configuration as input and classify the given configuration as stable or unstable. To do so, we create a pool of possible configurations as a training set and identify a functional relation that can relate the measurable quantities of a droplet assembly- like number of neighbors, local orientation, packing density, size ratio, number ratio etc.- to its stability.

Contents

List of figures

List of tables

Listings

Abbreviations and notations

PAv – Probability of avalanche

N – Total number of droplets in the system

SR – Size ratio

NR – Number ratio

Alpha – scaling factor in probability function

CHAPTER 1: INTRODUCTION

1.1 Coalescence in poly-disperse systems

High concentration of drops in a micro-channel can form closely packed arrangements. On creating disturbance in such a system, consisting of particles of varied sizes in a dispersed phase, can collapse the system completely making it unstable. Movement of droplets in a microchannel is a dynamic phenomenon as the arrangement of the droplets changes continuously. Therefore a perturbation created can propagate and grow to any size. This is possible because when two drops are pulled apart, the low pressure between the drops deforms and pulls both the interfaces together facilitating contact, which results in coalescence. Efforts have been made to understand the mechanism of coalescence.[1-3] In a concentrated emulsion, when two drops coalesce they move towards each other. In the process they get pulled away from every other drop in the neighborhood. This retraction results in a low pressure region around the coalescing pair, which avalanches into a cascade of coalescence events.[4] This can sometime lead to phase inversion, a phenomenon where continuous phase becomes dispersed and vice versa. An avalanche of huge size disturbs the equilibrium of the system.

An illustration for the propagation of perturbation is shown as below:

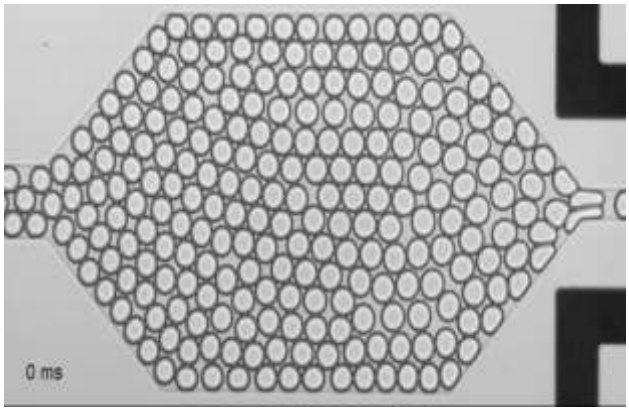


Fig 1.1.a Initial stage

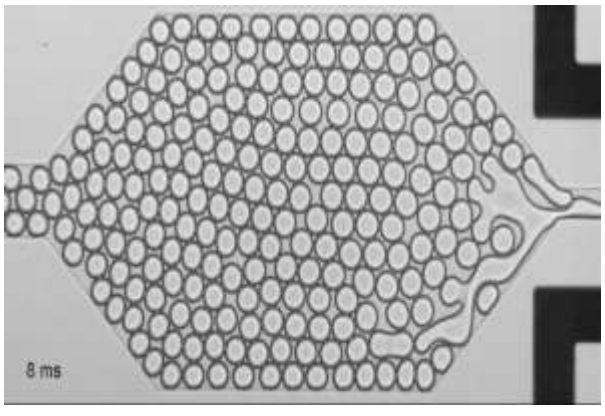


Fig 1.1.b Propagation starts

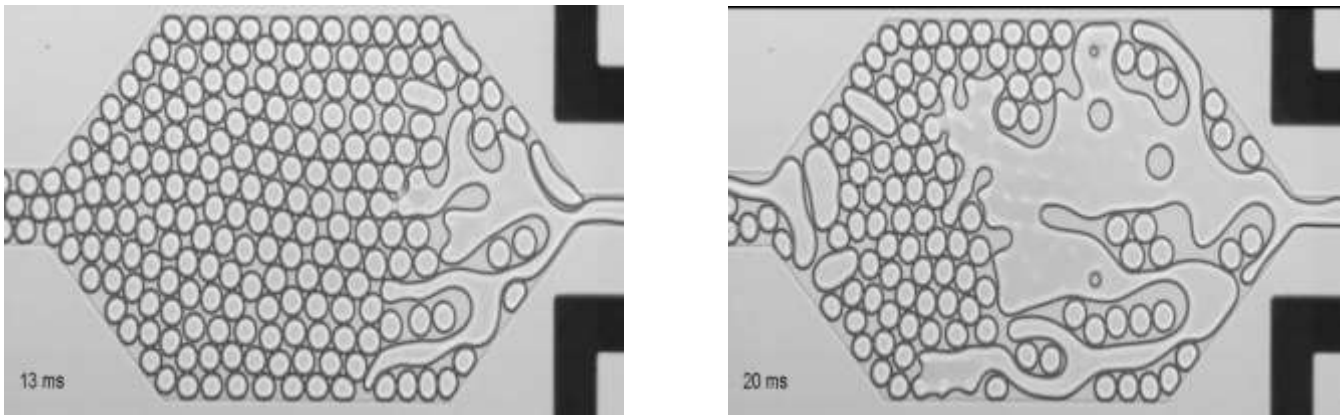


Fig 1.1.c & d Collusion of droplets

Fig 1.1: Propagation of coalescence in a system

1.2 Stochastic model for coalescence

Quantifying coalescence is difficult because of its dependency on several factors. Bremond et al. conducted experiments to find out that this dependency by coalescence made the process probabilistic. The stability of the emulsion depends on its immunity to coalescence events. If local coalescence events propagate through the assembly of drops, stability is compromised. Hence it is important to understand the collective behavior of drops in 2D micro channels.[5]

Coalescence depends on how the initiated perturbation propagates through the system. Hence not every coalescence event results in an avalanche. Depending on the angle made by the droplet and its neighbors, the next droplet is pulled into the coalescence. Even the angle between two coalescing droplets and their neighbors plays a role in choosing the next droplet. To understand this behavior, Bremond et al. considered a combination of three drops and experimentally measured the probability $P(\theta)$ of drop 3 to coalesce with a coalescing pair (1 and 2) as a function of the orientation of the three drops (defined by θ).[4,5] Though Bremond et al. fit a fourth order polynomial for the probability data from physics of the system, a cosine formulation is chosen for further calculations and computations.

Following are two figures showing different cases of coalescence:

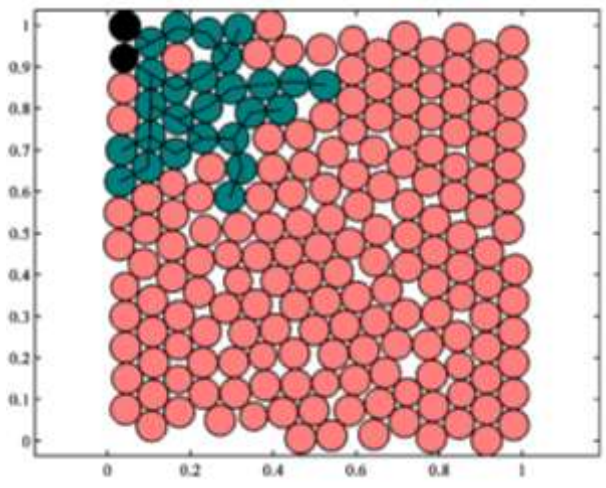


Fig 1.2.a: Avalanche not caused

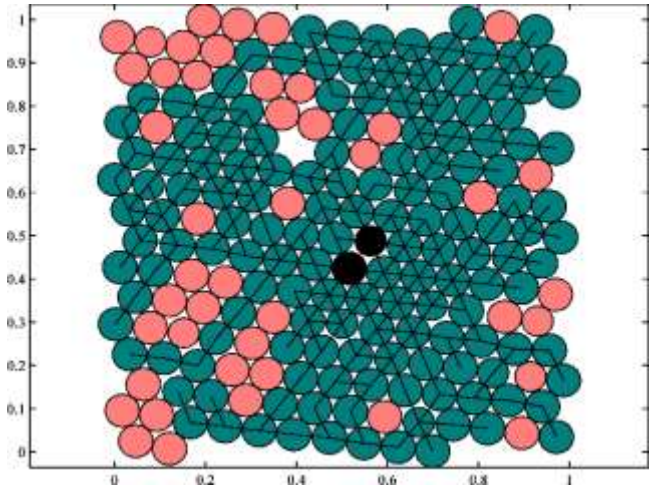


Fig 1.2.b: Resulted in avalanche

Fig 1.2: Results of coalescence

Following shows the probability data obtained after Bremond's experiments:

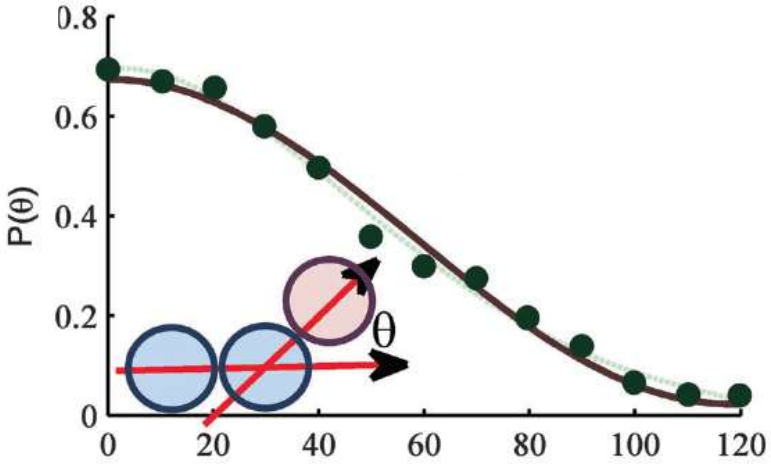


Fig 1.3 Probability of coalescence as a function of theta (data digitized from ref. 4), fit-polynomial (ref. 4) and P(θ)

Hence using this probability function can we quantify the stability of a system? We try to classify a system as stable or unstable by determining certain factors that define the system and maintain its unique nature. The usage of probability function comes into picture in two places. One during the computation of factors and other during determination of stability of system.

QUESTION OF THOUGHT: Given a snapshot of arrangement of the droplets can we say anything about the stability of the system?

CHAPTER 2: ALGORITHM

We try to classify the system as stable or not depending on certain factors obtained from the droplet arrangements. Hence we need dataset sufficient enough to train the model. We generate several configuration of droplets and run a Monte Carlo simulation on each configuration to determine the stability of the same configuration system. Then a machine learning algorithm is learnt on the dataset generated.

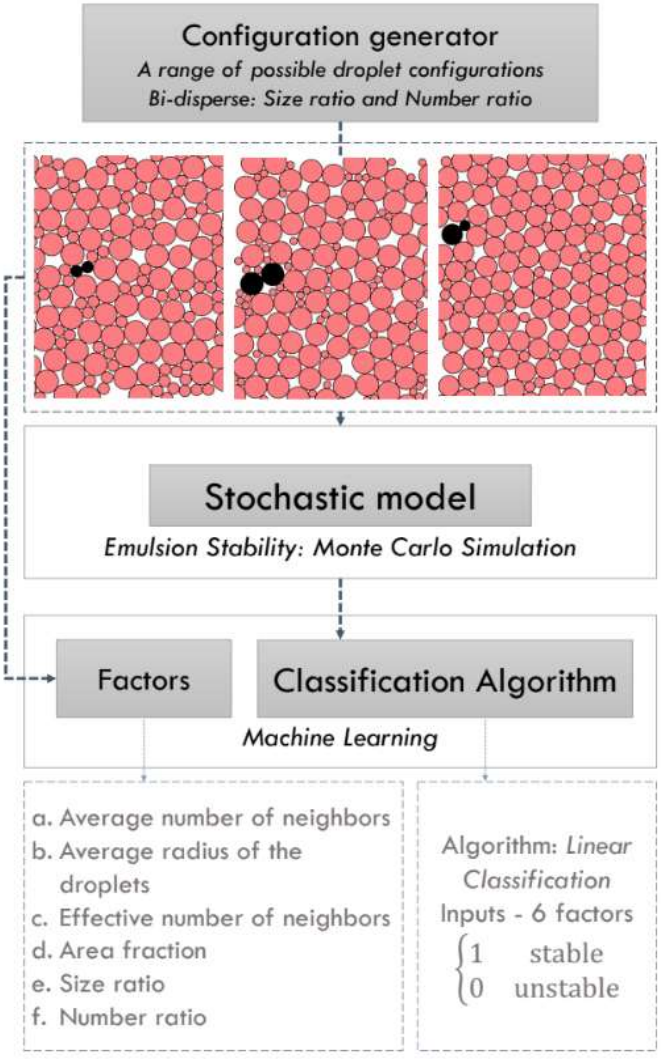


Fig 2.1: Flow chart describing the algorithm

2.1 Configuration generation:

In real life, the droplets packed in a confined area will be configured randomly with gaps between them. The droplets will not be of same size. The complex case being poly-disperse, we start by looking at bi-disperse system, two different sizes of droplets. Size ratio is the ratio of two different radii and number ratio is the ratio of total number of two different droplets. Imitating the real-life system of droplets, we create a numerous number of configurations using the algorithm as stated in - Random close packing of disks and spheres in confined geometries K.S.Desmond.[6]

In this paper, the author proposes a method to create compactly packed droplets system abiding by few input parameters. The algorithm takes N, the number of total particles, the size ratio and number ratio as inputs. It runs infinitely till a breaking condition is met. The algorithm first sets a confining boundary based on N. Next, N particles are placed randomly inside the boundary of zero size (points). Then they are expanded such that no particles overlap after any expansion. If there is an overlap, the droplets are moved or contracted until no particles overlap (within the allowed tolerance). Again the droplets are expanded but this time at a rate that is half of previous rate. Each time we switch from contraction to expansion, the rate of expansion is halved. Finally, when the rate of expansion is less than a certain specified number (breaking condition) and no particles are overlapping, the algorithm terminates. It is made sure that the droplets are expanded or contracted with respect to the size ratio and number ratio given to us beforehand. Since the radii as such are not fixed, the radii of the droplets vary among configurations slightly. But this is at the least importance to us, as absolute sizes do not matter. In this way, we created configurations for several N, size ratio and number ratio.

2.2 Factors:

From understanding of physics, the following is the list of factors that could possibly affect the propagation of the coalescence:

1. Size ratio:

   Ratio of radii of two different droplets in the system

2. Number ratio:

   Ratio of number of two different sized droplets in the system

3. Average area:

   Average area is the ratio of area occupied by the droplets and the total number of droplets in the packed structure. It is a measure of how closely the droplets are packed. The more closely they are packed, more the chances of big avalanche.

4. Average neighbors:

   Average number of neighbors for every droplet in the assembly is counted. Higher the value, higher the number of directions the coalescence can propagate, hence higher the chance of big avalanche.

5. Effective neighbors:

   The number of neighbors weighted by the probability for propagation defines this factor. Effective neighbors is calculated for all neighbor pairs in the assembly. It affects the system in a similar way as average neighbors.

6. Average radius:

   Average radius gives the average of radius of all the droplets in the configuration. Average radius in monodisperse case is the radius of droplet itself. Average radius in bi-disperse case lies between the two radii.

7. N:

   Number of droplets in the system

Determination of factors: (input of the classification algorithm)

Configuration files generated in section 2.1 is used for the calculation of the factors.

1. Effective neighbors:
    - Consider every pair of adjacent droplets. Let total number of pairs be - NC. Let the pair be droplet A and droplet B
    - Looping over all the neighbors of A and B, we calculate the angle between line joining A & B and line joining the neighbor and one of A or B. Let the angle be theta (θ)
    - 0.3552*cosd(1.565*θ)+0.3183+0.05 - is calculated and summed over all such pairs
    - Dividing the above obtained quantity with total number of pairs we get effective neighbors of that configuration
2. Average area: Total area occupied by the drops/ N
3. Average neighbors: Sum of neighbors of each droplet/ N
4. Average radius: Sum of all the radii/ N
5. Potential: Sum of all the distances between droplets/ N

Number ratio, Size ratio and N: Fixed in the configuration generation code

```
for configN=1:nSR
    cn=int2str(configN);
    filename=['FinalConfig',cn];
    F=load (filename);
    x0=(F(:,1));
    y0=(F(:,2));
    rad_list=importfile(['system', cn], 6,8);
    %%Factor_4 - average radius && Factor_1 - Average area
    NR=rad_list(1);
    R1=rad_list(2);
    R2=rad_list(3);
    ar1 = pi*R1*R1;
    ar2 = pi*R2*R2;
    if R1>R2
        avg_rad(1,configN,sr) = (NR*R2 + (1-NR)*R1)/N;
        area_fr(1,configN,sr) = (N*NR*ar2 + N*(1-NR)*ar1)/N;
    else
        avg_rad(1,configN,sr) = (NR*R2 + (1-NR)*R1)/N;
        area_fr(1,configN,sr) = (N*NR*ar1 + N*(1-NR)*ar2)/N;
    end
    %%Factor_2 - Average no: of neighbours
    [nc, n1]=neighbour_find(configN);
    avg_neigh(1,configN,sr) = mean(nc);
```

```
%%Factor_3 - Effective no: of neighbours
eff_pairwise = zeros(1500,nSR);
 for j_d1 = 1:N
     p = nc(j_d1);
     l = nl(j_d1,:);
     for j = 1:p
         j_d2 = l(j);
         for kk=1:p
             if(l(kk) == j_d2)
                 continue;
             end
             a1=[(x0(l(kk),1)-x0(j_d1,1)); (y0(l(kk),1)-y0(j_d1,1))];
             b1=[(x0(j_d2,1)-x0(j_d1,1)); (y0(j_d2,1)-y0(j_d1,1))];
             theta=real(acosd(dot(a1,b1)/(norm(a1)*norm(b1))));
             eff_pairwise(k(1,configN),configN) = eff_pairwise(k(1,configN),configN) + 0.3552*cosd(1.565*theta)+0.3183+0.05;
             eff_neigh(1,configN,sr)= eff_neigh(1,configN,sr) + 0.3552*cosd(1.565*theta)+0.3183+0.05;
         end
         k(1,configN) = k(1,configN)+1;
     end
 end
 eff_neigh(1,configN,sr) = 2*eff_neigh(1,configN,sr)/sum(nc);
end
```

Listing 2.1: Computation of factors

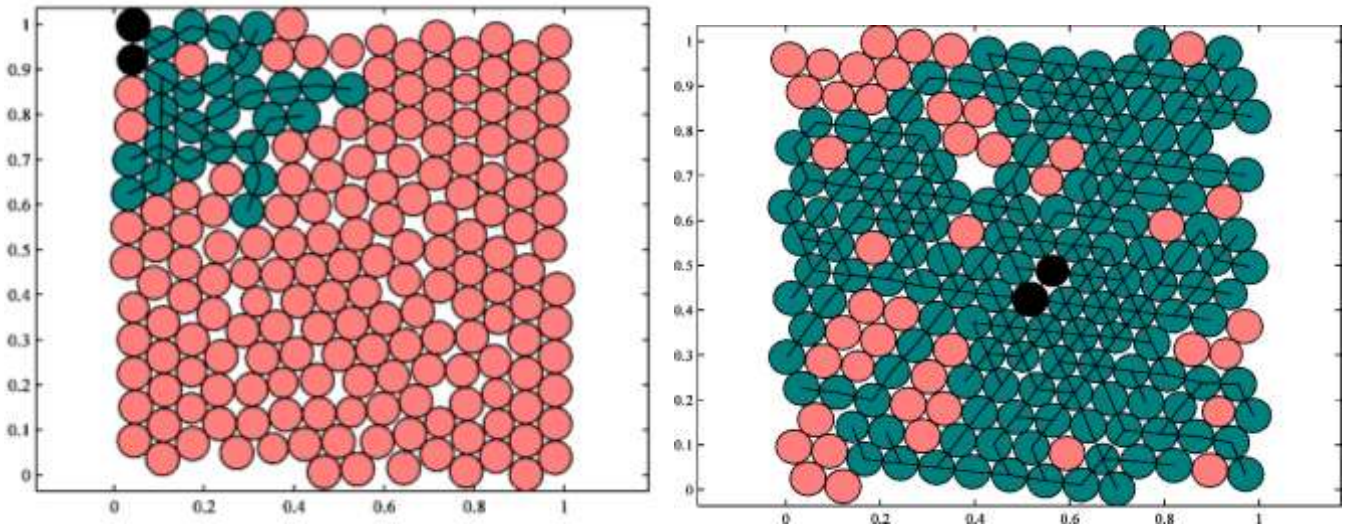
2.3 Stability:

Configuration files generated in section 2.1 is sent into a Monte Carlo simulation to obtain output files consisting of an array which has the sizes of avalanche resulted when a disturbance was triggered randomly in the system. The simulation is run for a large number of times (Here 1e5 times). In this simulation the propagation of coalescence uses the aforementioned probability model. Plotting the histogram of the array of avalanche sizes gives us a curve which is similar to the probability curve of size of avalanche. If this curve is monotonic we call the system as stable and unstable otherwise (non-monotonic). Intuitively this distinction makes sense because a stable system will have high probability of resulting in small sized avalanche and low probability of resulting in big sized avalanche.

Monotonicity determination:

A polynomial, Y is fit over the histogram. By plotting numerous curves, it can be seen that there will be either two zero derivative points or no such points. Let the points be A and B. Points A and B are determined by differentiating the polynomial curve and equating it to zero. If A and B are imaginary then the curve is monotonic. If not then it is actually non-monotonic. But here

we give some threshold for monotonicity calculation. We calculate Y(A) - Y(B). If this difference is greater than the threshold, it is marked as non-monotonic. Here we are using a threshold of 0.06. Therefore if the peak and dip are very far apart, we consider the system to be unstable.



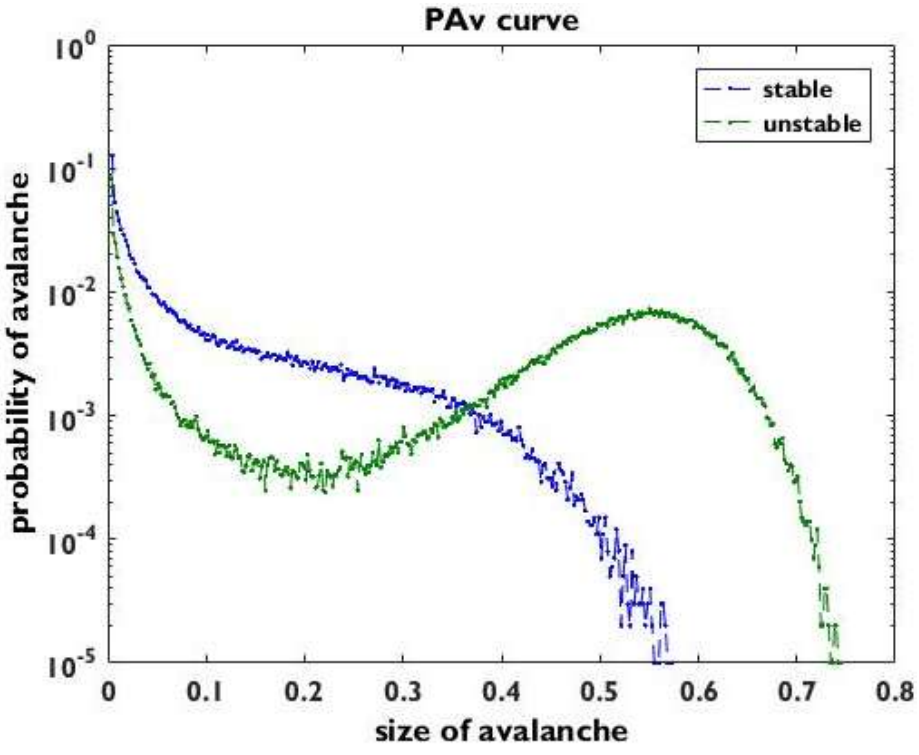


Fig 2.2: PAv curve of stable and unstable system

```
for r=1:nSR
    cn=int2str(r);
    name=['Output_nr_0_sr_',k_r,'_',cn];
    load(name);
    bins=2:1:1e5;
    nn=grids;
    counts=hist(aval_grid,bins);
    X=bins;
    Y=counts/sum(counts);
    Y=Y(find(Y));
    X=X(find(Y));
    semilogy(X,Y,'--.')
    hold all
    disp(r);
    %% Cubic fit and stability determination
    p = polyfit(X,log10(Y),3);
    y_fit = polyval(p,X);
    root = roots([3*p(1,1) 2*p(1,2) p(1,3)]);
    diff_y(r) = polyval(p,root(1,1)) - polyval(p,root(2,1));
    if(isreal(diff_y(r))==1)
        if(diff_y(r)<0.06)
            monotonic(1,r,sr) = 1;
        end
    else monotonic(1,r,sr) = 1;
    end
    %%Average avalanche size for various NR and SR
    avg_aval(1,r,sr) = mean(aval_grid);
end
```

Listing 2.2: Determining stability of a system

2.4 Classification:

2.4.1 Data set for classifications:

The dataset generated covers the following cases:

Parameters	Values
N	144, 196, 225
SR	1, 1.2, 1.4, 1.6, 1.8, 2, 2.2, 2.4 2.6, 2.8, 3
NR	0, 0.1, 0.2, 0.3, 0.4, 0.5
Alpha	0.9, 1, 1.1

Table 2.1: Parameters and values

Since the process of generating configurations is randomized, in each case we generate 35 configurations.

Hence the total number of observations = 3*6*11*3*35 = 20,790

Before classification we need two datasets - training and testing. And both the dataset should include all the possible NR, SR, N and alpha. This preprocessing of the whole data is written as a code. Given the percentage (x%) of data to be considered for training dataset, the code outputs training and test dataset. It takes x% of the 35 configuration from each possible case and adds it to the training data and remaining to test dataset.

```
function [X_train,X_test,Y_train,Y_test]=preprocess(division,nSR,N_R,SR)
    index = int16(nSR*division);
    X_train = zeros(1,8)
    X_test = zeros(1,8)
    Y_train = zeros(1,1)
    Y_test = zeros(1,1)
    number_ratio = [0,0.1,0.2,0.3,0.4,0.5];
    size_ratio = [1,1.2,1.4,1.6,1.8,2,2.2,2.4,2.5,2.6,2.8,3];
    for alpha = [0.9]
        cd(['data_alpha_',int2str(alpha)]);
        for N = [225]
            factors = load(['factors_',int2str(N),'.mat']);
            stability = load(['training_data_',int2str(N),'.mat']);
            avg_rad = factors.avg_rad;
            avg_neigh = factors.avg_neigh;
            eff_neigh = factors.eff_neigh;
            area_frac = factors.area_fr;
            monotonic = stability.monotonic;
            for i = 1:SR
                for j = 1:N_R
                    temp = area_frac(j,1:index,i)';
                    temp = [temp,avg_neigh(j,1:index,i)'];
                    temp = [temp,eff_neigh(j,1:index,i)'];
                    temp = [temp,avg_rad(j,1:index,i)'];
                    temp = [temp,number_ratio(1,j)*ones(index,1)];
                    temp = [temp,size_ratio(1,i)*ones(index,1)];
                    temp = [temp,alpha*ones(index,1)];
                    temp = [temp,N*ones(index,1)];
                    X_train = [X_train;temp];
                    Y_train = [Y_train;monotonic(j,1:index,i)'];
                    temp = area_frac(j,index+1:nSR,i)';
                    temp = [temp,avg_neigh(j,index+1:nSR,i)'];
                    temp = [temp,eff_neigh(j,index+1:nSR,i)'];
                    temp = [temp,avg_rad(j,index+1:nSR,i)'];
                    temp = [temp,number_ratio(1,j)*ones(nSR-index,1)];
```

```
                    temp = [temp,size_ratio(l,i)*ones(nSR-index,l)];
                    temp = [temp,alpha*ones(nSR-index,l)];
                    temp = [temp,N*ones(nSR-index,l)];
                    X_test = [X_test;temp];
                    Y_test = [Y_test;monotonic(j,index+1:nSR,i)'];
                end
            end
        end
        cd ..
    end
    X_train(1,:) = [];
    X_test(1,:) = [];
    Y_train(:,2) = 0;
    Y_test(:,2) = 0;
    Y_train(1,:) = [];
    Y_test(1,:) = [];
    idx2 = find(Y_train(:,1)); Y_train(idx2,2) = 1; idx1 = find(not(Y_train(:,1)));
    Y_train(idx1,1) = 1; Y_train(idx2,1) = 0;
    idx2 = find(Y_test(:,1)); Y_test(idx2,2) = 1; idx1 = find(not(Y_test(:,1)));
    Y_test(idx1,1) = 1; Y_test(idx2,1) = 0;
    in = Y_train(:,2);
    Y_train(:,2) = Y_train(:,1);
    Y_train(:,1) = in;
    in = Y_test(:,2);
    Y_test(:,2) = Y_test(:,1);
    Y_test(:,1) = in;
end
```

Listing 2.3: Pre-processing of data

2.4.2 Logistic regression:

Using the above datasets, coefficients of the dividing line are obtained from training dataset and the error is calculated from the test dataset.

Classification, in the field of machine learning, is a problem where we try to identify, to which given set of categories, a new observation of our variable of interest belongs to. Here we are trying to identify whether a given system is stable or not. Hence, our variable of interest is stability and since it can take only two values - true or false, we have two categories. This is

termed as binary classification. There are several algorithms, out of which logistic regression is used here.

The logistic regression outputs a line which divides the stability variable into two regions 1 (true) and 0 (false), the two categories. This line will be a linear combination of the predetermined factors. The dividing line will be a1x1 + a2x2 + … = 0.5 where x1, x2… - factors. Before passing the dataset into the logistic regression function the dataset is scaled appropriately. Similarly the test dataset is scaled too, before computing the error.

A1, a2 … are equivalent to weightage given to each factor. That is, how well a factor is able to divide the output region. From the plots we can see whether the factors play any role in determining the stability of the system. The process of determining these weights is known as learning or training the classifier. We train our classifier on certain data not as training data and measure the performance using test data.

We define the error as the number of points that have been classified wrongly on test data that is, stable system being classified as unstable and vice versa. We also compute the confusion matrix which counts the number of true positive, true negative, false positive and false negative.

```
function [err]=class_all(division)
[X_train,X_test,Y_train,Y_test] = preprocess(division,35,6,11);
err = 0;
%%complete data
nonzero_indices = find(Y_train(:,1));
zero_indices = find(not(Y_train(:,1)));
size2 = size(nonzero_indices);
size1 = size(zero_indices);
Y = cell(size1(1,1)+size2(1,1),1);
for i = 1:size2(1,1)
    Y{nonzero_indices(i,1),1} = 'stable';
end
for i = 1:size1(1,1)
    Y{zero_indices(i,1),1} = 'unstable';
end
sp = categorical(Y);
X = X_train;
%X(:,8) = sqrt(X(:,8));
X(:,7) = [];
X(:,4) = [];
X = X - ones(size1(1,1)+size2(1,1),1)*mean(X,1);
stan_dev = std(X,1);
for i = 1:6
    X(:,i) = X(:,i)/stan_dev(1,i);
end
B_logistic = mnrfit(X,sp);

X = X_test;
size_test = size(Y_test);
%X(:,8) = sqrt(X(:,8));
X(:,7) = [];
X(:,4) = [];
X = X - ones(size_test(1,1),1)*mean(X,1);
stan_dev = std(X,1);
for i = 1:6
    X(:,i) = X(:,i)/stan_dev(1,i);
end
Y_test_pred = mnrval(B_logistic,X);
Y_pred = Y_test(:,1);
for i = 1:size_test(1,1)
    if(Y_test_pred(i,1)>Y_test_pred(i,2))
        Y_pred(i,1) = 1;
    else Y_pred(i,1) = 0;
    end
end
%%NORMAL ERROR CALCULATION
for i = 1:size_test(1,1)
    if(Y_test(i,1) ~= Y_pred(i,1))
        err = err + 1;
    end
end
err = err/size_test(1,1)
end
```

Listing 2.4: Logistic regression and error calculation

2.4.3 Plots:

Few plots to see the importance of each factors:

Single Factor:

Now the impact of each factor towards stability of the system is visualized. Following are the 1D plots with X axis as factor. Blue dots correspond to stable system whereas red dots correspond to unstable system.

These correspond to a specific SR = 2, N = 196 and alpha = 1. The separating line is obtained by using logistic regression. That is output (stability) is regressed with only one factor and it's corresponding separating line is obtained.

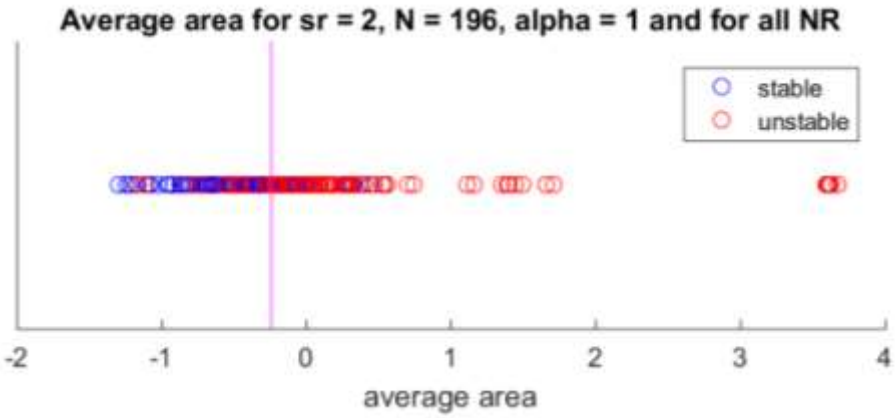


Fig 2.3.a: Average area

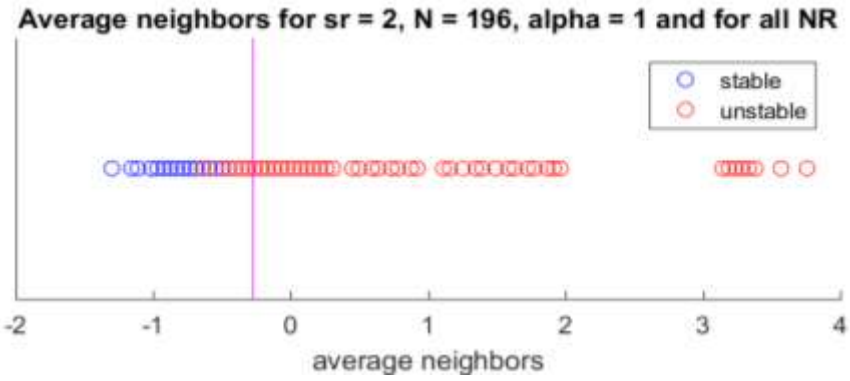


Fig 2.3.b: Average neighbors

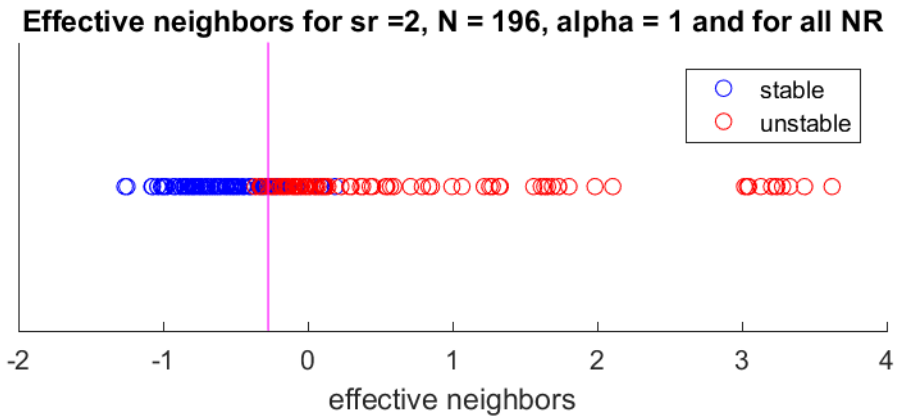


Fig 2.3.c: Effective neighbors

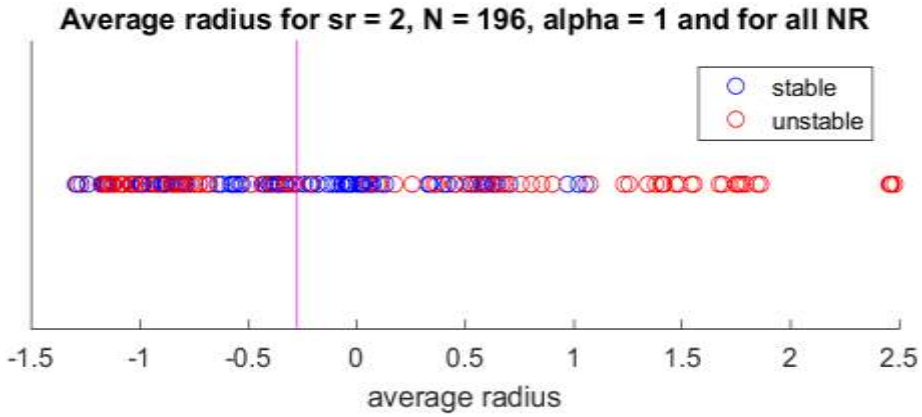


Fig 2.3.d: Average radius

Two factors:

Visualizing how combination of factors affect the stability of the system. Following are the 2D plots with X axis and Y axis as factors. Blue dots correspond to stable system whereas red dots correspond to unstable system.

These again correspond to the same set, SR = 2, N = 196 and alpha = 1. Here the output (stability) is regressed with two factors and their corresponding separating line is obtained.

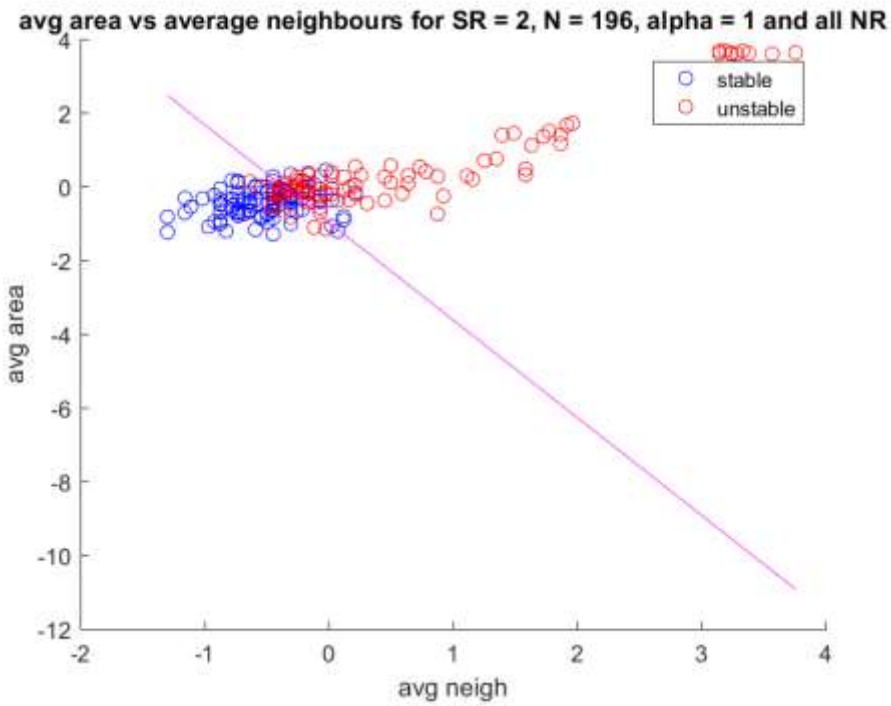


Fig 2.4.a: Average area vs Average neighbors

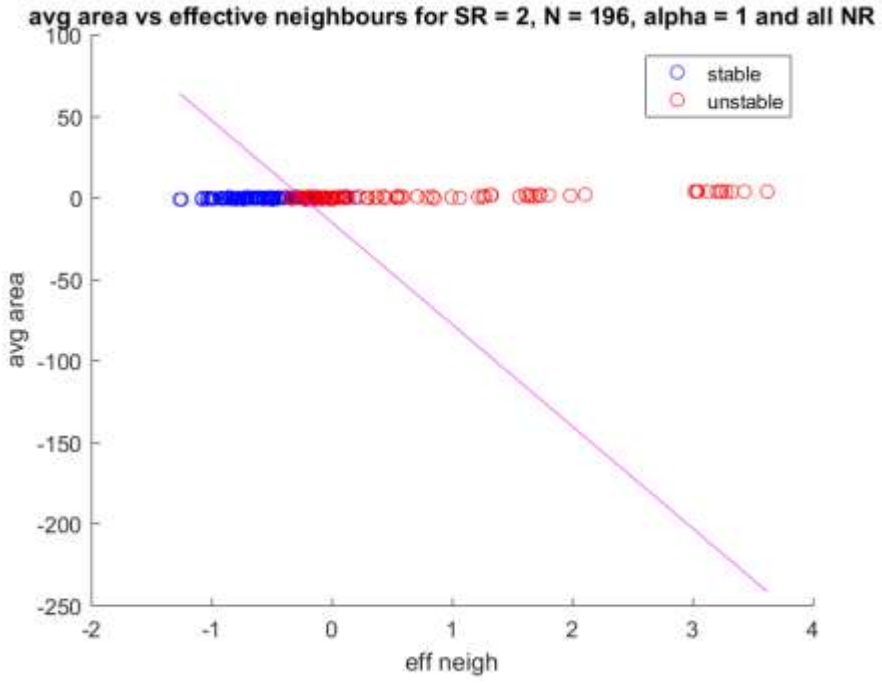


Fig 2.4.b: Average area vs Effective neighbors

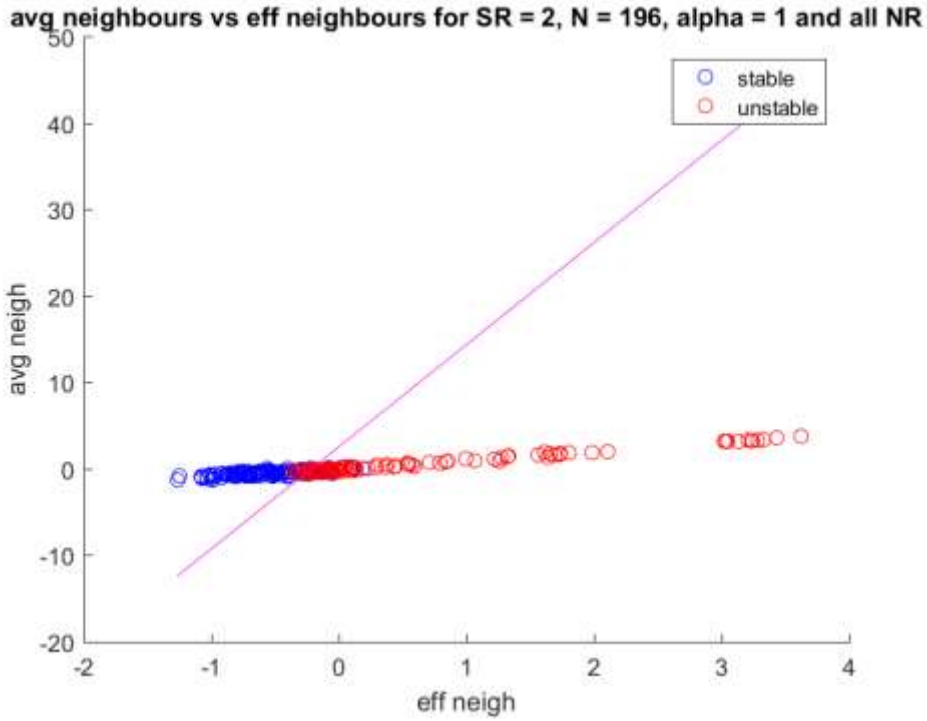


Fig 2.4.c: Average neighbors vs Effective neighbors

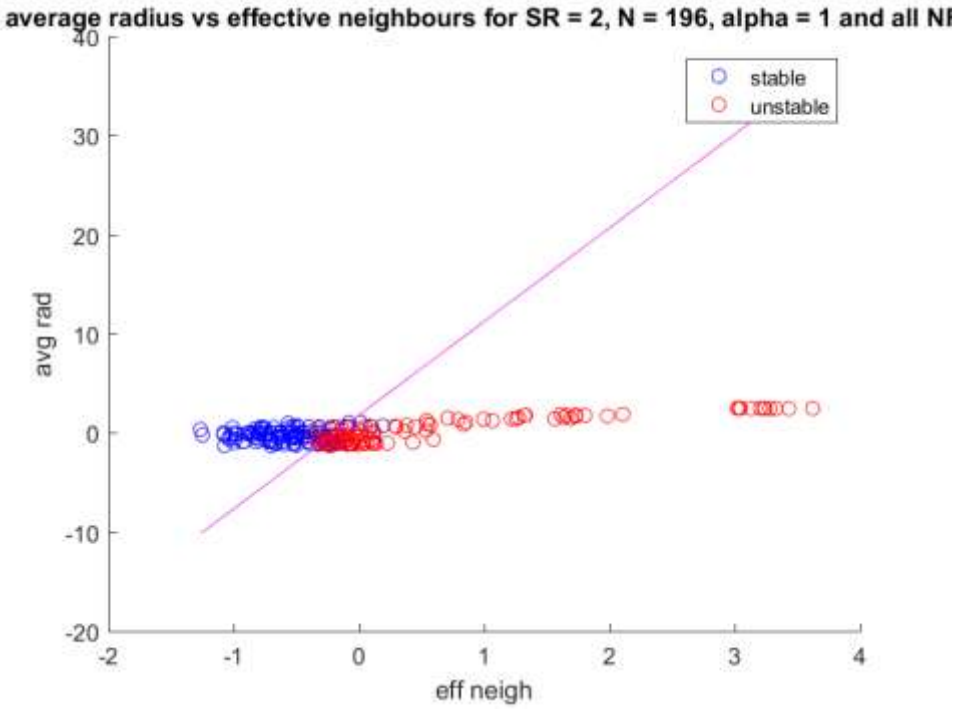


Fig 2.4.d: Average radius vs Effective neighbors

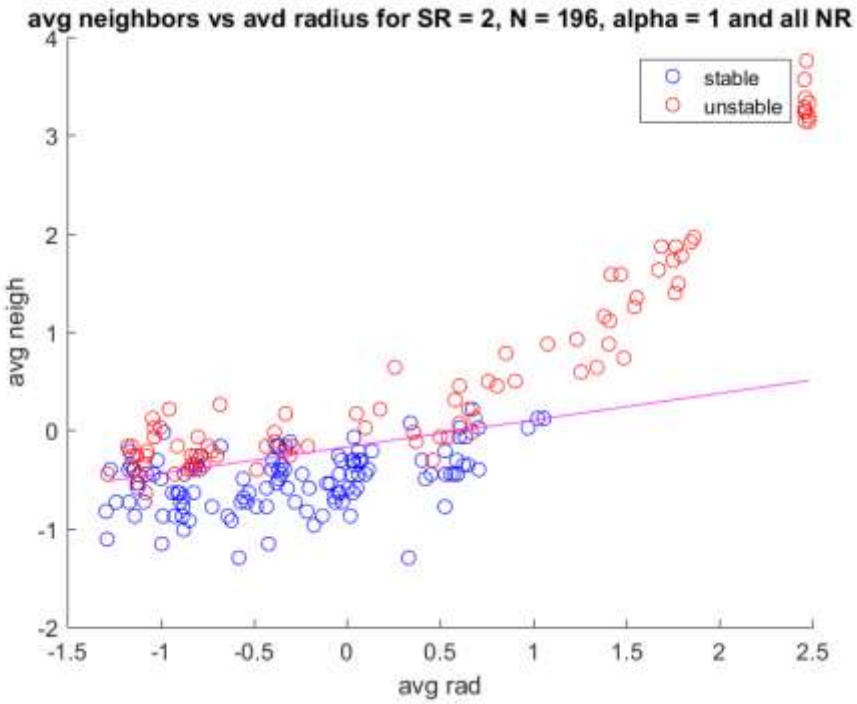


Fig 2.4.e: Average neighbors vs Average radius

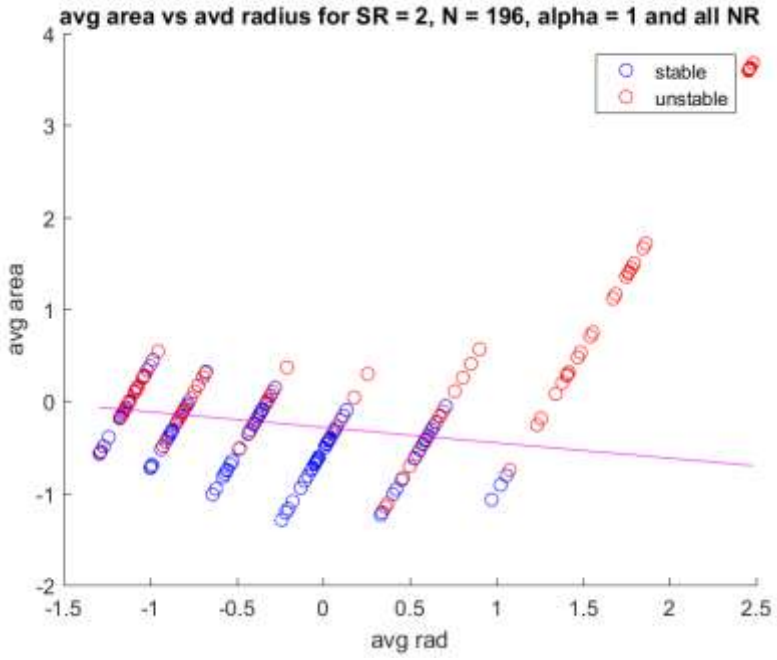


Fig 2.4.f: Average area vs Average radius

We can see visually from the graphs that when two variables are used in regression, we get a better classifier. This motivates us to proceed with considering all the factors. Before proceeding, we need to check for factor importance and correlation. From the average radius vs average area plot, it is clear that they are highly correlated. For further clarity we look into correlation matrix.

```
[X_train,X_test,Y_train,Y_test] = preprocess(1,35,6,11);
X_train = [X_train,Y_train(:,1),Y_train(:,2)];
X = X_train
nonzero_indices = find(X(:,9));
zero_indices = find(not(X(:,9)));
size2 = size(nonzero_indices)
size1 = size(zero_indices)
Y = cell(size1(1,1)+size2(1,1),1);
for i = 1:size2
    Y{nonzero_indices(i,1),1} = 'stable';
end
for i = 1:size1
    Y{zero_indices(i,1),1} = 'unstable';
end
sp = categorical(Y);
X = X - ones(size1(1,1)+size2(1,1),1)*mean(X,1);
stan_dev = std(X,1);
for i = 1:4
    X(:,i) = X(:,i)/stan_dev(1,i);
end

var = X(:,1)
B_logistic = mnrfit(X(:,1),sp)
figure()
hold all;
plot(var(nonzero_indices),0.1*ones(1,length(nonzero_indices)),'bo');
plot(var(zero_indices),0.1*ones(1,length(zero_indices)),'ro');
plot(((0.5-B_logistic(1,1))/B_logistic(2,1))*[1,1],[1,0],'m');
```

Listing 2.5.a: Plot generation – single factor

```
[X_train,X_test,Y_train,Y_test] = preprocess(1,35,6,11);
X_train = [X_train,Y_train(:,1),Y_train(:,2)];
X = X_train
nonzero_indices = find(X(:,9));
zero_indices = find(not(X(:,9)));
size2 = size(nonzero_indices)
size1 = size(zero_indices)
Y = cell(size1(1,1)+size2(1,1),1);
for i = 1:size2
    Y{nonzero_indices(i,1),1} = 'stable';
end
for i = 1:size1
    Y{zero_indices(i,1),1} = 'unstable';
end
sp = categorical(Y);
X = X - ones(size1(1,1)+size2(1,1),1)*mean(X,1);
stan_dev = std(X,1);
for i = 1:4
    X(:,i) = X(:,i)/stan_dev(1,i);
end
X_all(:,1) = X(:,1)
X_all(:,2) = X(:,2);
var1 = X_all(:,1)
var2 = X_all(:,2)
B_logistic = mnrfit(X_all,sp)
%%
figure()
hold all;
plot(var2(nonzero_indices),var1(nonzero_indices),'bo');
plot(var2(zero_indices),var1(zero_indices),'ro');
plot(X_all(:,2),(-(B_logistic(3,1))*X_all(:,2)+(0.5-B_logistic(1,1)))/(B_logistic(2,1)),'m');
```

Listing 2.5.b: Plot generation – two factor

CHAPTER 3: FACTOR SELECTION

3.1 Correlation

The correlation matrix for all the considered factors is as shown as below.

	Average area	Average neighbours	Effective neighbours	Average radius	NR	SR	N
Average area	1	-0.1281	-0.0426	0.9944	-0.0289	-0.0096	-0.9909
Average neighbours	-0.1281	1	0.8948	-0.1190	-0.5245	-0.3131	0.2024
Effective neighbours	-0.0426	0.8948	1	-0.0710	-0.4211	-0.0276	0.1193
Average radius	0.9944	-0.1190	-0.0710	1	-0.0711	-0.0673	-0.9853
NR	-0.0289	-0.5245	-0.4211	-0.0711	1	0	0
SR	-0.0096	-0.3131	-0.0276	-0.0673	0	1	0
N	-0.9909	0.2024	0.1193	-0.9853	0	0	1

Table 3.1: Correlation matrix

From the table it can be seen that there is high correlation between:

1) Average area and average radius
2) Average area and N
3) Average radius and N
4) Average neighbors and effective neighbors

Now each case has to be taken separately and analyzed. Depending on whether average area is included or average radius one of the case 2 and case 3 will be redundant.

3.1.1 Correlation between average radius and average area

From the plots correlation between average radius and average area is very high. Since area and radius are related, so will their averages be. Hence one among them can be considered for final regression.

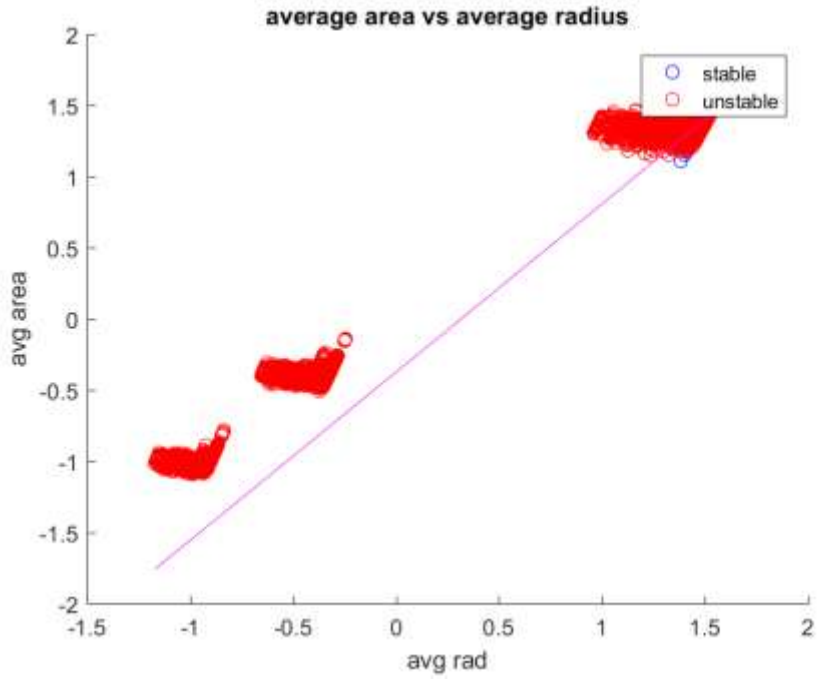


Fig 3.1: Average radius and Average area

We plot the error for various size of training dataset and test dataset to obtain the error plot. There will be dip in the plot and on comparing the dip we can get the factor with minimum error. The factors considered in two cases are:

Case 1: NR, SR, average area, average neighbors, effective neighbors

Case 2: NR, SR, average radius, average neighbors, effective neighbors

Currently the error plotted is misclassification error (%), that is, percentage of test data misclassified.

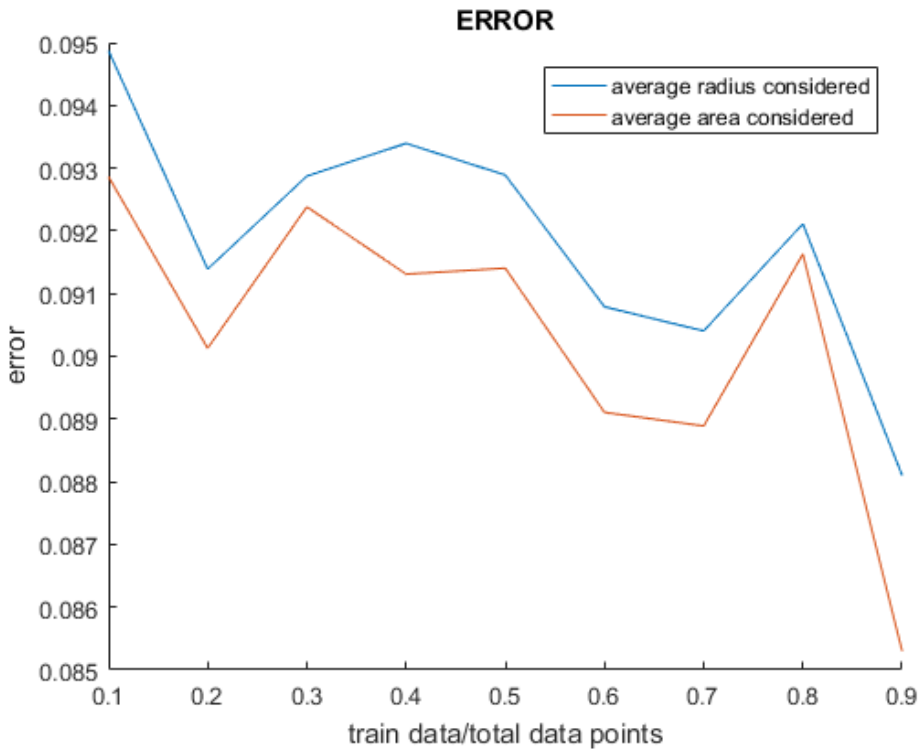


Fig 3.2: Error plot – average radius & average area

Training data/ Total data	Error (Average radius considered)	Error (Average area considered)
10%	0.0949	0.0929
20%	0.0914	0.0901
30%	0.0929	0.0924
40%	0.0934	0.0913
50%	0.0929	0.0914
60%	0.0908	0.0891
70%	0.0904	0.0889
80%	0.0921	0.0916
90%	0.0881	0.0853

Table 3.2: Comparison of error – average radius vs average area

From the table, both the dips occur when 90% of the data is used for training and 10% is used as test data. Clearly average area gives better results, hence only average area is considered.

3.1.2 Correlation between N and average area

Similarly the error plot is visualized. The factors considered in two cases are:

Case 1: NR, SR, average area, average neighbors, effective neighbors

Case 2: NR, SR, N, average neighbors, effective neighbors

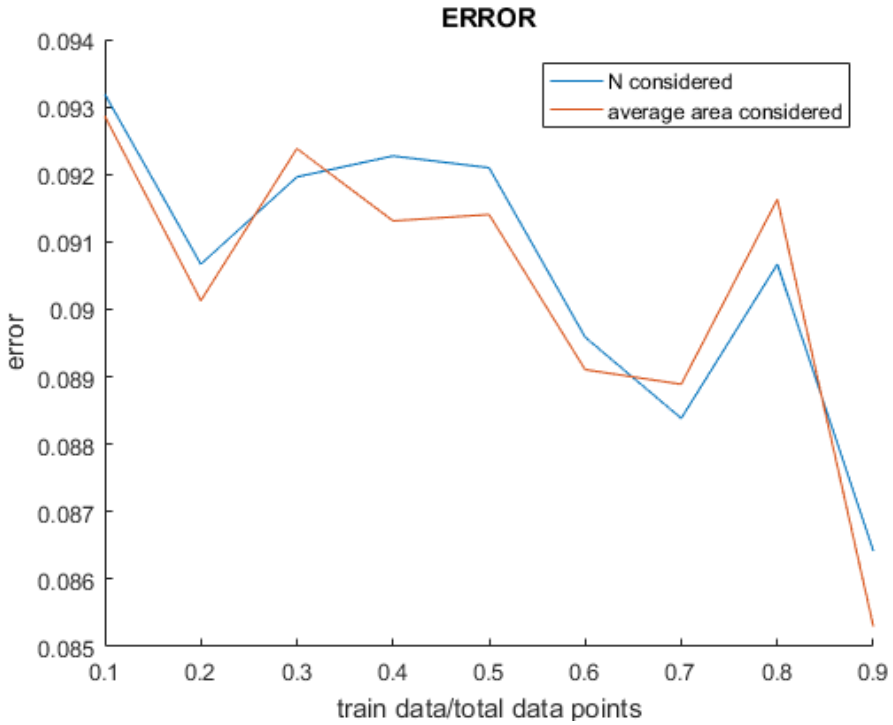


Fig 3.3: Error plot – N & average area

Training data/ Total data	Error (N considered)	Error (Average area considered)
10%	0.0932	0.0929
20%	0.0907	0.0901
30%	0.0920	0.0924
40%	0.0923	0.0913
50%	0.0921	0.0914
60%	0.0896	0.0891
70%	0.0884	0.0889
80%	0.0907	0.0916
90%	0.0864	0.0853

Table 3.3: Comparison of error – N vs average area

From the table, both the dips occur when 90% of the data is used for training and 10% is used as test data. The error difference is very low, hence any one of the both factors can be considered. Since the overall minimum error occurs when average area is considered, we drop out N.

3.1.3 Correlation between effective neighbors and average neighbors

Similarly the error plot is visualized. The factors considered in two cases are:

Case 1: NR, SR, average area, average neighbors

Case 2: NR, SR, average area, effective neighbors

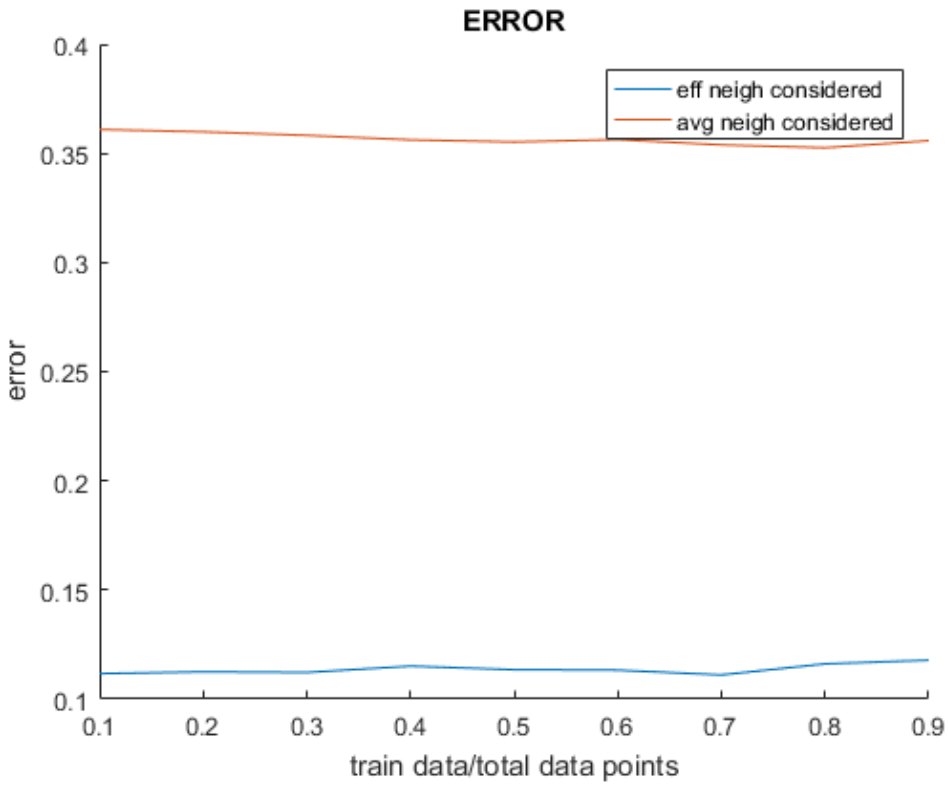


Fig 3.4: Error plot – average neighbors & effective neighbors

From the graph, effective neighbors is the most important factor. But the minimum error 11.78% is greater than the error when both the factors are included. Hence no factor is dropped.

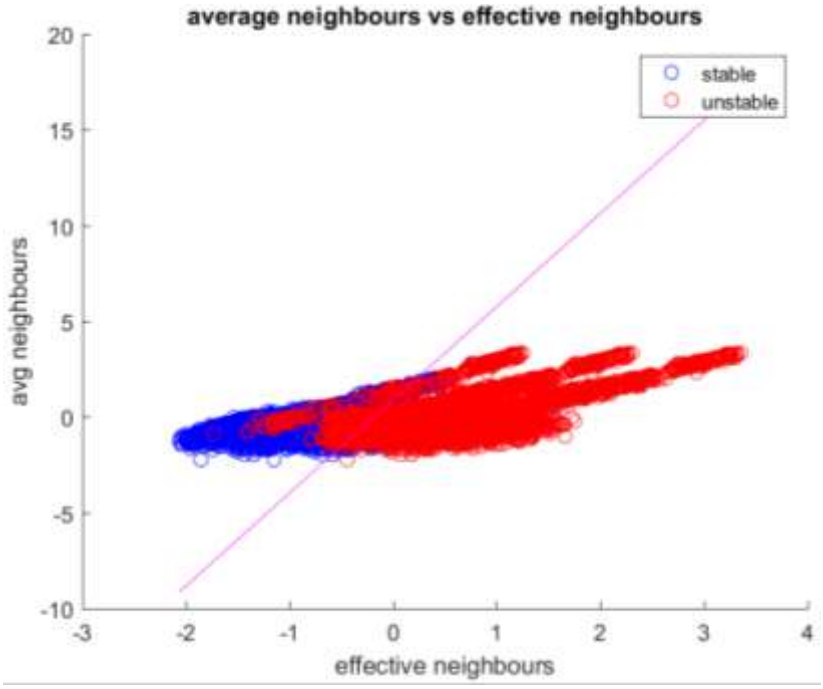


Fig 3.5: Average neighbors and effective neighbors

The above plot also supports the decision of not dropping any factor as from visualization we can see that classifier performs well.

3.2 Factor importance

Minimum error occurs when 90% of the total data is considered for training but this can also be because of very small test data set. Hence the next minimum at 70% (training data/ total data) is also considered. 100% train data is also considered for reference. Logistic regression coefficients for both the cases are tabulated to see the factors importance.

Train data/ Total data	Constant	Average area	Average neighbours	Effective neighbours	NR	SR
100%	-0.2768	0.7103	5.3425	-23.1086	0.8745	1.1535
90%	-0.2735	0.7129	5.3656	-23.0875	0.8466	1.1451
70%	-0.2648	0.7076	5.5119	-23.0511	1.0648	1.1857

Table 3.4: Regression coefficients for various cases

All the three cases follow the same order of factor importance. That is,

1) Effective neighbors
2) Average neighbors
3) SR
4) NR
5) Average area

3.3 Confusion matrix

Looking at the confusion matrix for two different cases:

Case 1: 70% of the total data is training data

Case 2: 90% of the total data is training data

Train data/ total data = 70%	Predicted = stable	Predicted = unstable
Actual = stable	True positive = 2682	False negative = 241
Actual = unstable	False positive = 287	True negative = 2730

Table 3.5.a: Confusion matrix for case 1

Train data/ total data = 90%	Predicted = stable	Predicted = unstable
Actual = stable	True positive = 809	False negative = 68
Actual = unstable	False positive = 84	True negative = 821

Table 3.5.b: Confusion matrix for case 2

Among the misclassifications, stable system classified as unstable has less cost compared to unstable system classified as stable. Precision accounts for this and it is a measure of number of unstable system classified as stable. Hence higher the precision, better it is.

Precision = true positive/(true positive + false positive)

(Case 1 - 70% of the total data is training data) Precision: 90.33%

(Case 2 - 90% of the total data is training data) Precision: 90.6%

```
%%MATRIX ERROR CALCULATION
matrix = [0,0;0,0];
false_positive = [];
false_negative = [];
true_positive = [];
true_negative = [];
for i = 1:size_test(1,1)
    if(Y_test(i,1) ~= Y_pred(i,1))
        if(Y_test(i,1) == 1)
            matrix(1,2) = matrix(1,2)+1;
            false_negative = [false_negative,i];
        else
            matrix(2,1) = matrix(2,1)+1;
            false_positive = [false_positive,i];
        end
    end
    if(Y_test(i,1) == Y_pred(i,1))
        if(Y_test(i,1) == 1)
            matrix(1,1) = matrix(1,1)+1;
            true_positive = [true_positive,i];
        else
            matrix(2,2) = matrix(2,2)+1;
            true_negative = [true_negative,i];
        end
    end
end
```

Listing 3.1: Confusion matrix

CHAPTER 4: RESULTS

4.1 Single factor plots

The plots are plotted for all SR and NR for the following nine cases:

Alpha = 0.9, 1, 1.1 and N = 144, 196, 225

4.1.1 Average area

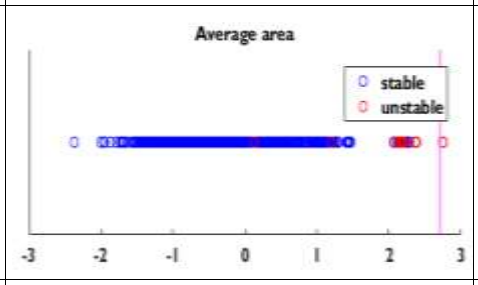
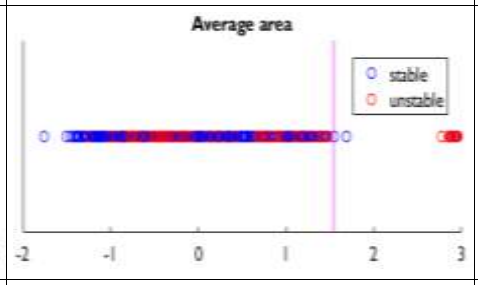
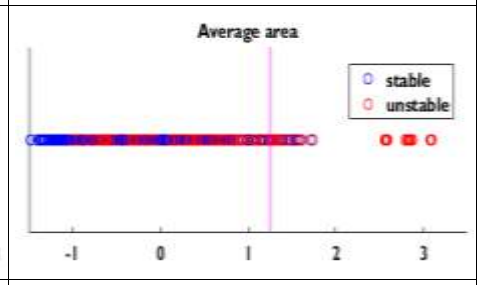
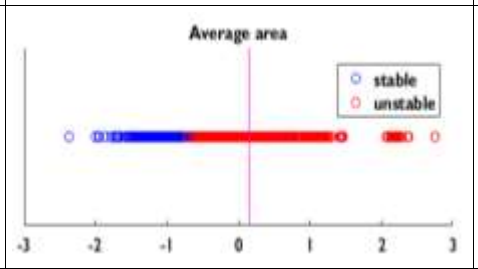
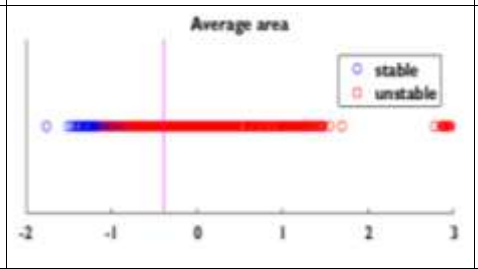
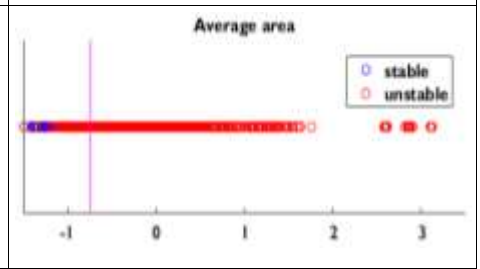
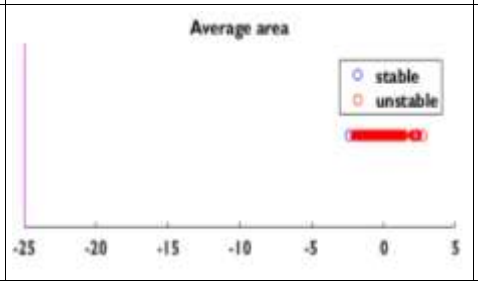
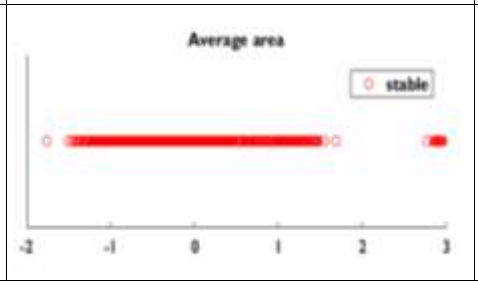
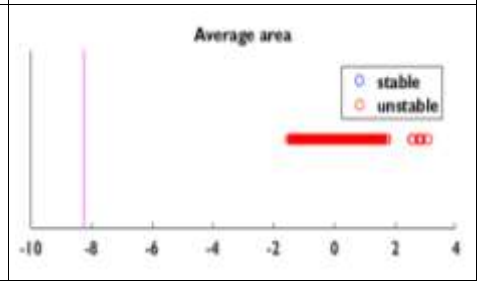
	144	196	225
0.9			
1			
1.1			

Table 4.1: Single factor plots – Average area

4.1.2 Average neighbors

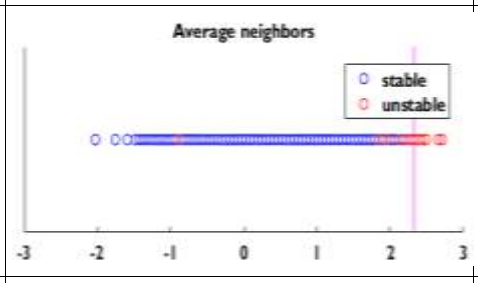
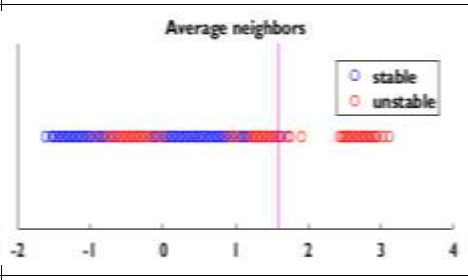
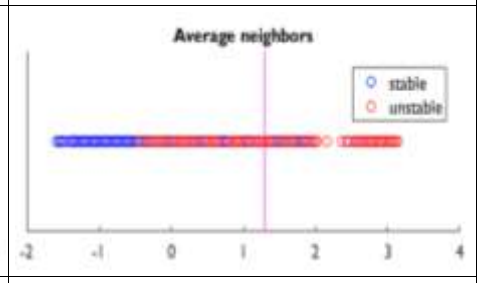
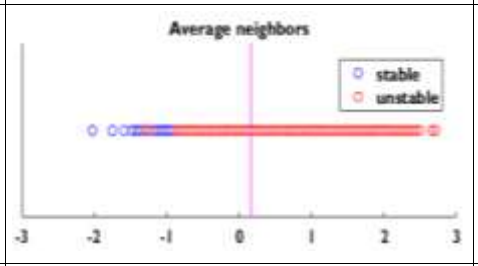
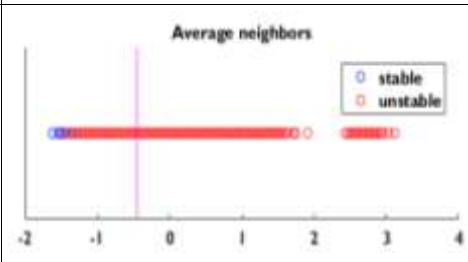
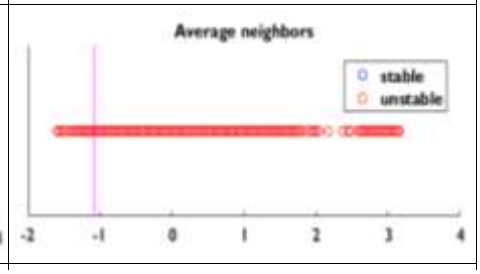
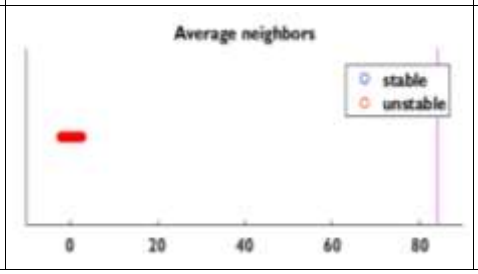
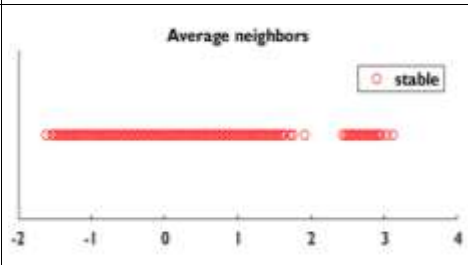
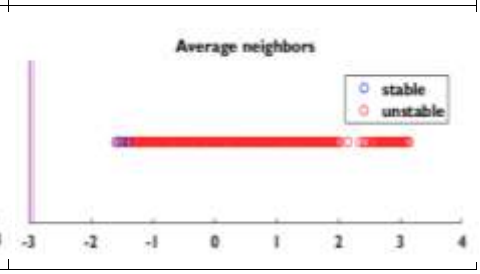
	144	196	225
0.9			
1			
1.1			

Table 4.2: Single factor plots – Average neighbors

We can see that distribution of stable and unstable cases depends on both N and alpha. Low alpha and low N results in more stable cases and high alpha and high N results in more unstable cases. This also implies for fixed alpha as N increases instability in the system increases and similarly for fixed N, as alpha increases instability in the system increases.

4.1.3 Effective neighbors:

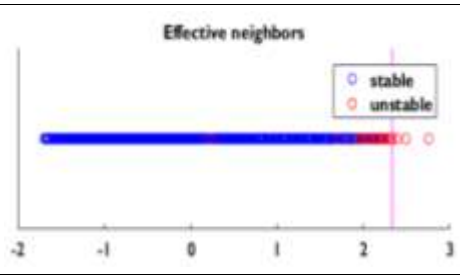
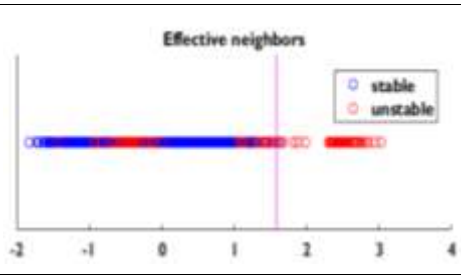
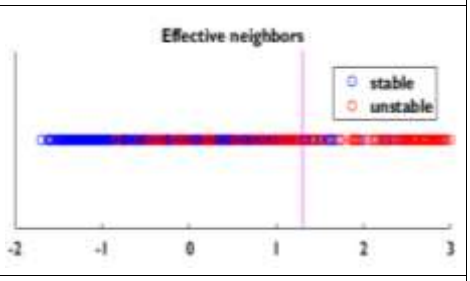
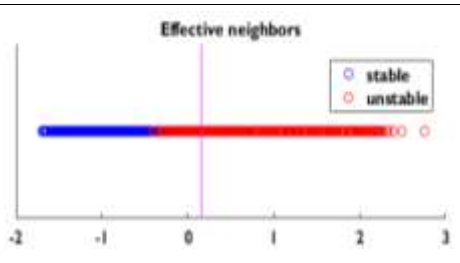
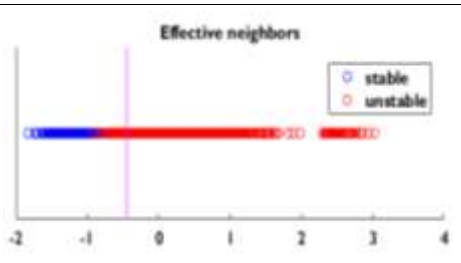
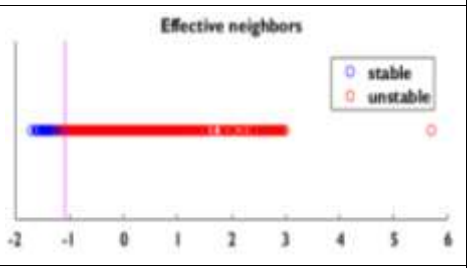
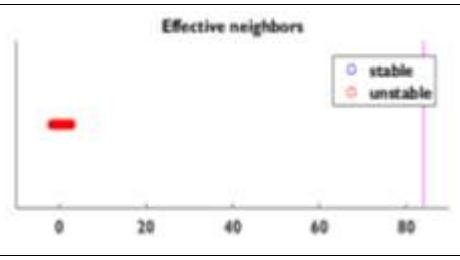
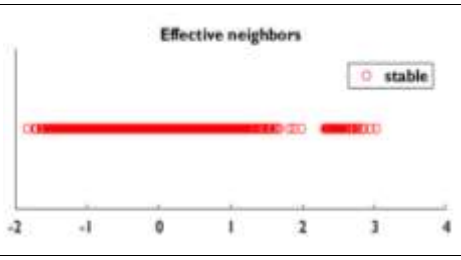
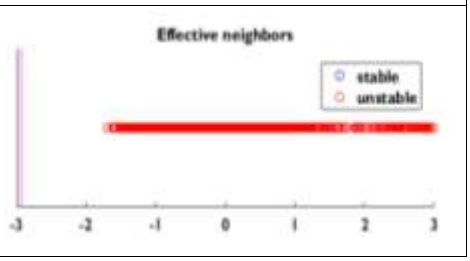
	144	196	225
0.9			
1			
1.1			

Table 4.3: Single factor plots – Effective neighbors

There are also cases where all the possible outcomes are unstable or stable, in such cases regression cannot be run since only one class prevails. But a point to note is that if the data is generated again by running Monte Carlo simulation we might get different representation of two classes. This variation comes because of the randomness in choosing the initial point of trigger in each configuration of droplets.

4.2 Two factor plots

Similarly two factor plots are plotted for all SR and NR for the following nine cases:

Alpha = 0.9, 1, 1.1 and N = 144, 196, 225

4.2.1 Average area vs Average neighbors:

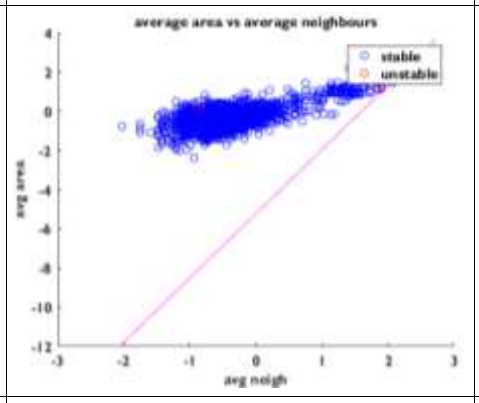
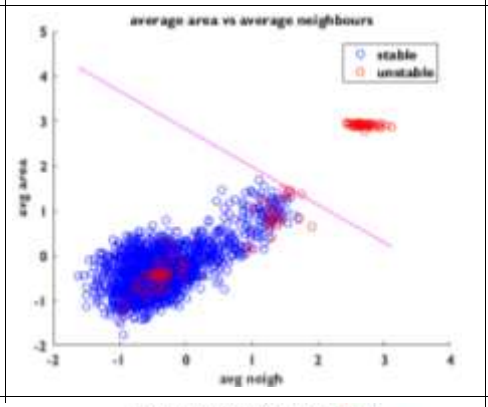
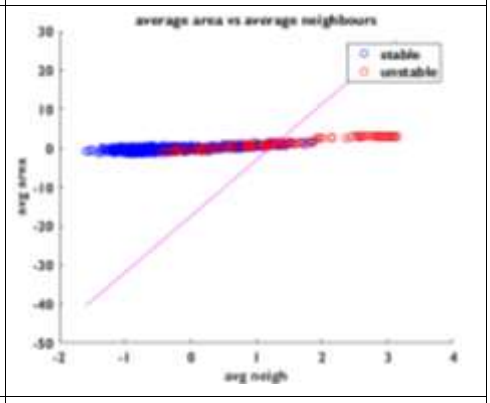
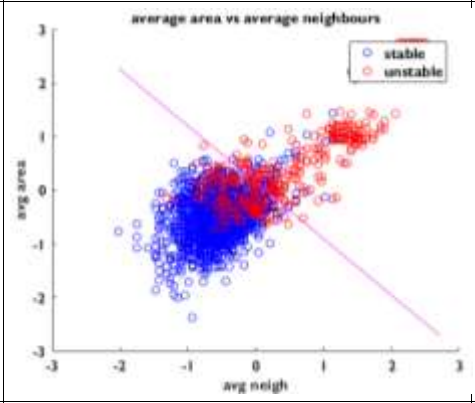
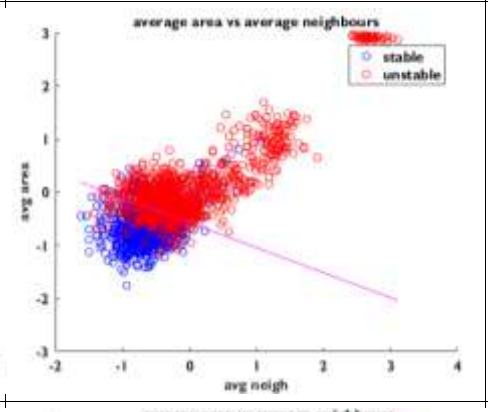
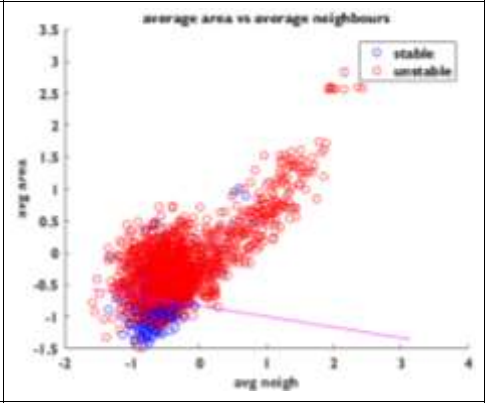
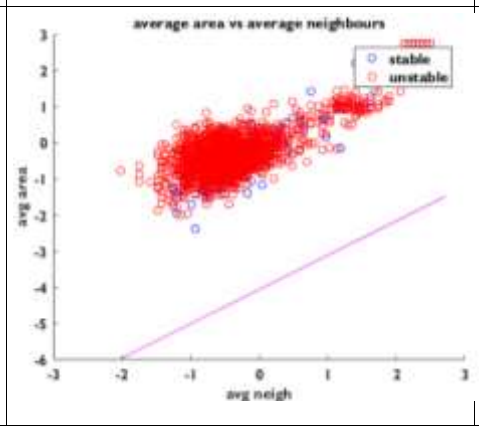
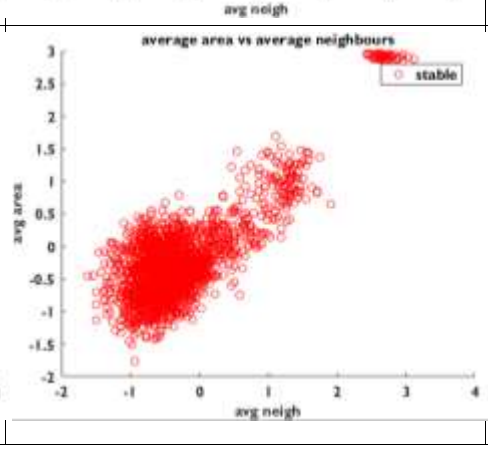
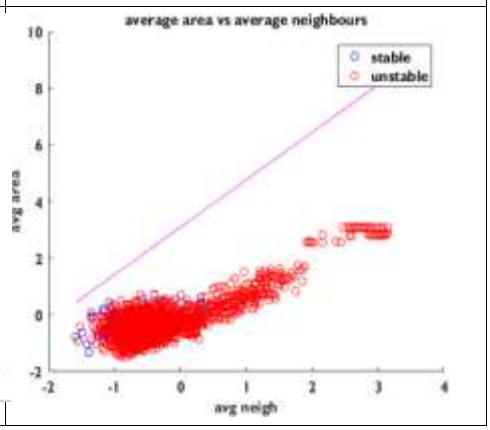
	144	196	225
0.9			
1			
1.1			

Table 4.4: Two factor plots – Average area vs average neighbors

4.2.2 Average area vs Effective neighbors:

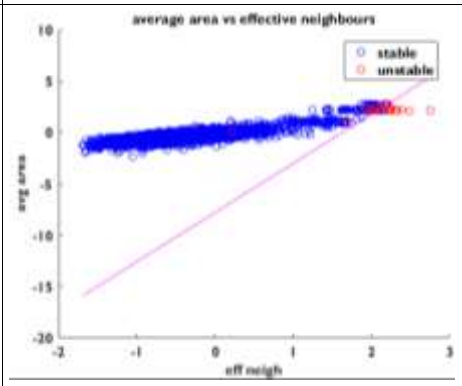
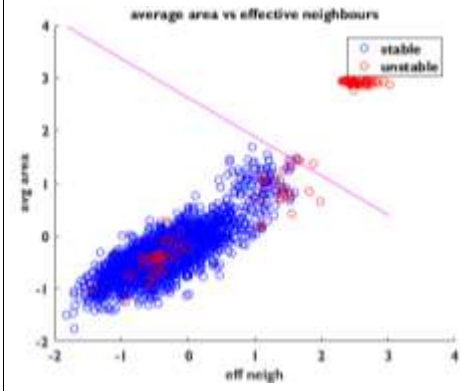
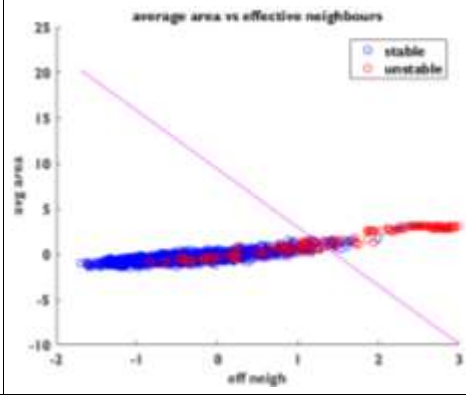
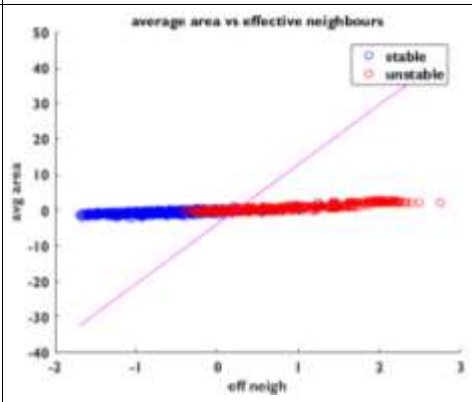
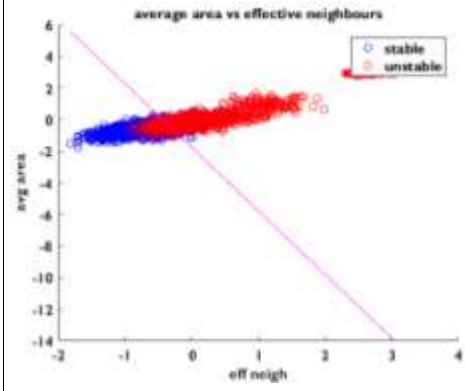
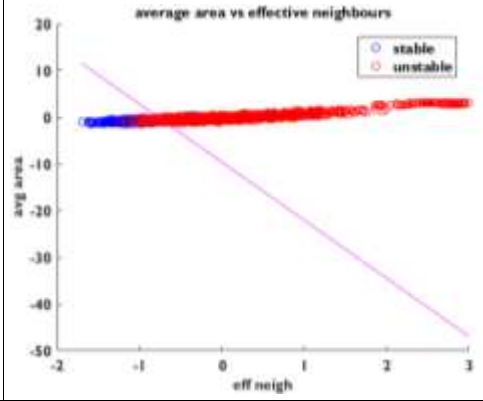
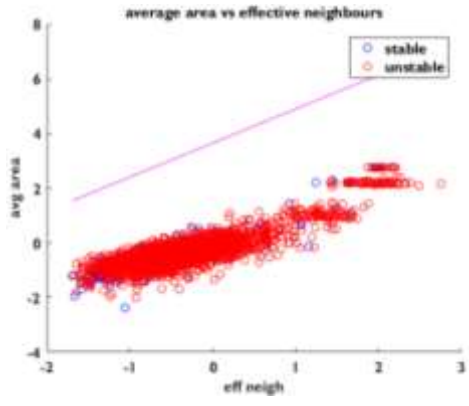
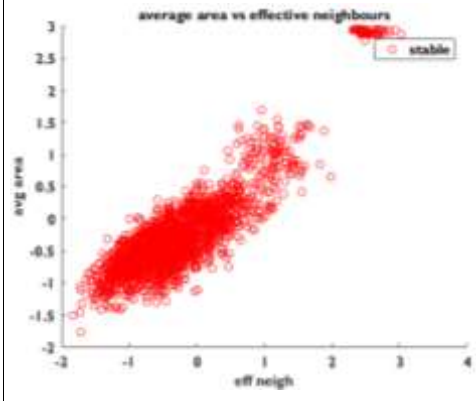
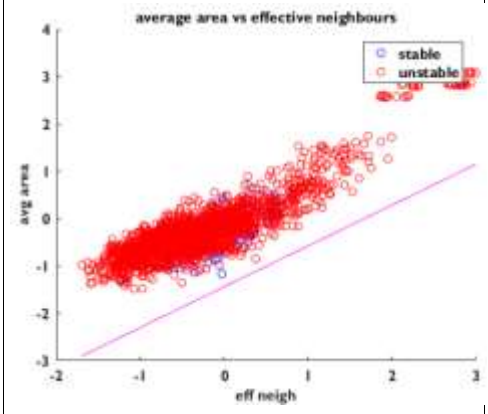
	144	196	225
0.9			
1			
1.1			

Table 4.5: Two factor plots – Average area vs effective neighbors

4.2.3 Effective neighbors vs Average neighbors:

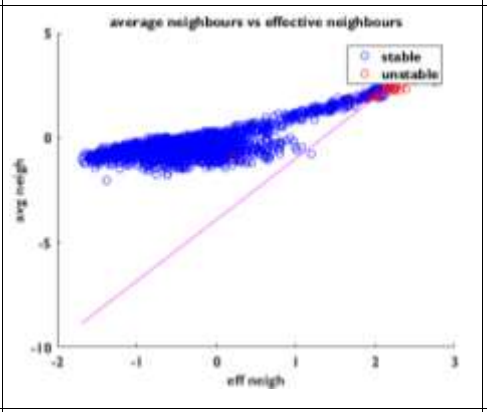
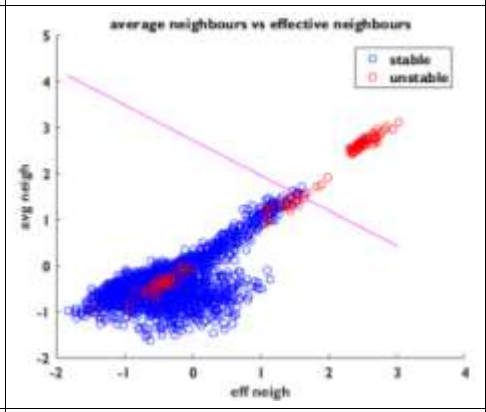
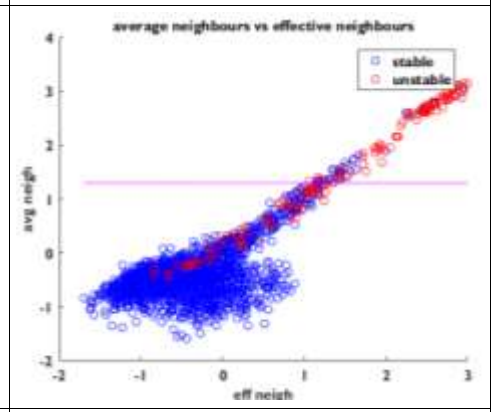
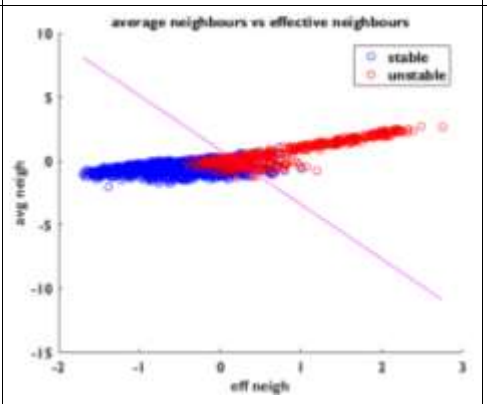
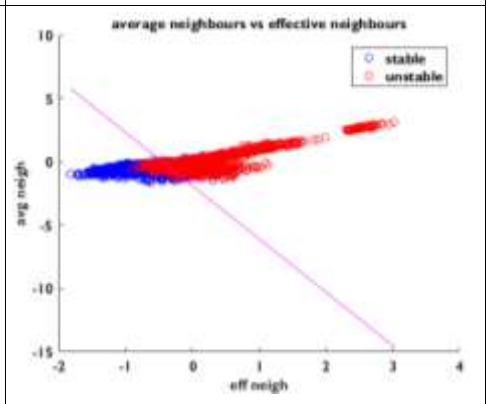
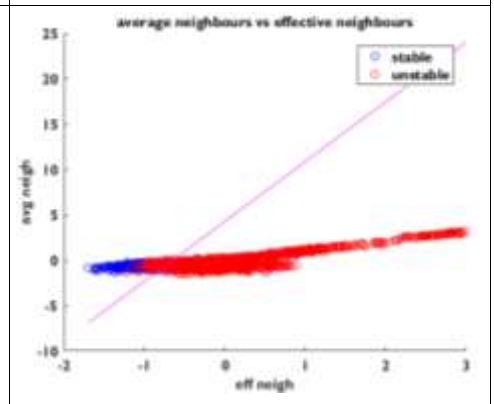
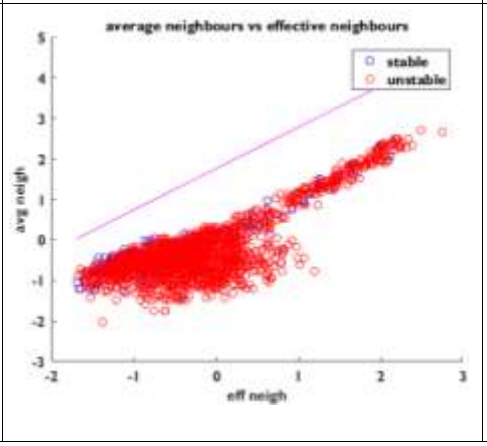
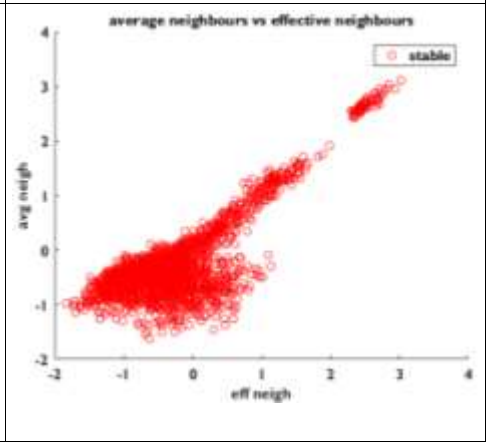
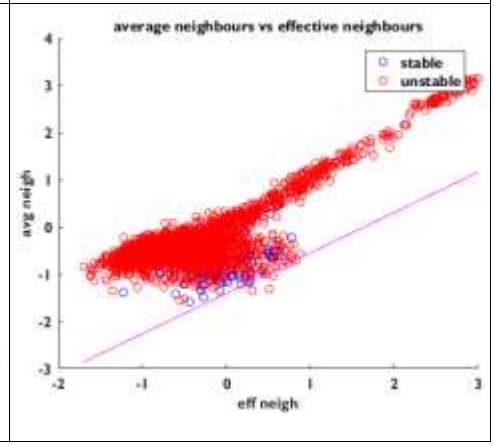
	144	196	225
0.9			
1			
1.1			

Table 4.6: Two factor plots – Effective neighbors vs average neighbors

We can see that if stable and unstable area overlaps the classifier performs badly. Same conclusions as to single factor plots can be drawn here too. Now considering all the factors together we obtain the following results.

CHAPTER 5: CONCLUSION

Total number of observations: 20790

Factors considered: Effective neighbors, average neighbors, NR, SR and average area

Order of importance:

1) Effective neighbors
2) Average neighbors
3) SR
4) NR
5) Average area

Best accuracy obtained: 8.53%

Best precision: 90.6%

This is the first time where machine learning technique has been applied in the field of soft matter. An error of 8.53% and precision of 90.6% is a good step considering that basic classification method was used. This method has its own limitations. Also, it was tried out with limited number of factors chosen based on physical knowledge. Given time a better data set can also be generated.

Hence the model can be improved a lot more by increasing the data set size and number of factors. Several other classifiers can be applied and tested. This is the first step taken, hoping it to be taken forward.

Bibliography

1) D. Y. C. Chan, E. Klaseboer and R. Manica, Soft Matter, 2009, 6, 20
2) D. Y. C. Chan, E. Klaseboer and R. Manica, Soft Matter, 2011, 7, 2235
3) A. Lai, N. Bremond and H. a. Stone, J. Fluid Mech., 2009, 632, 97
4) N. Bremond, H. Dome´jean and J. Bibette, Phys. Rev. Lett., 2011, 106, 214502
5) M. Danny Raj and Raghunathan Rengaswamy, Coalescence of drops in a 2D microchannel: critical transitions to autocatalytic behavior, 2016
6) KS Desmond, Random close packing of disks and spheres in confined geometries, 2009