Santander Customer Satisfaction: Term Project Report

Namho An, Ariel Brest, Shaul Vin

COSI 123A: Statistical Machine Learning

May 6, 2016

TABLE OF CONTENTS

Method 1 (simple_classification.m)

Description – Code in Appendix A

This code is based on a simple regression tree. First the training data is loaded and separated into parts, which are the labels, data, and results. Then, using the classregtree function, the data is trained based on the results. After that, the testing data is loaded and separated into labels and data. The test data is then classified based on the regression tree calculated by the training data and the results are matched with the labels into a csv file.

Results

This code did not produce a usable result that we were able to submit to Kaggle. As a result, some changes were made that are reflected in Method 2.

What Did We Learn?

As this was the first attempt, we did not have high expectations of success. Through this attempt we were able to figure out processing times and how to properly submit data. Since we did not get a result, we were unable to figure out how well our model fit the data until the next method.

Method 2 (simple_classification1.m)

Description – Code in Appendix B

This code is similar to the code used in Method 1. First the training data is loaded and separated into labels, data, and results. Then a classification decision tree is calculated based on the function ClassificationTree.fit, which utilizes a binary classification tree. Then the testing data is loaded and separated into labels and test data before the classification tree is applied to the test data. The results are loaded into a csv file with the labels.

Results

When the results were submitted on Kaggle, it received a score of 0.534850.

What Did We Learn?

Since our results were low, we decided to seek an alternative route for calculating results, ultimately deciding to implement a neural network to improve our results. The first version of our neural network approach is seen in the next method.

Method 3 (annclass.m and annmain.m)

Description – Code in Appendix C and D

This method utilizes a neural network for classification. The first set of code, annclass.m, trains the decision tree using a random subset of data to reduce the time needed to train the data. annclass.m builds a neural network with two hidden layers. The program utilizes both a tan-sigmoid transfer function and a linear transfer function to calculate an output from the net input. Since the process in annclass.m is random, each run will give a different result. Therefore, it is not necessary to run this script multiple times. Instead, the best run is stored in a file labelled res1.mat. Then this file is used in the script, annmain.m, which classifies the test data. The resulting data was then configured and uploaded to Kaggle for results.

Results

When the results were submitted onto Kaggle, it received a score of 0.61192, which is a significant increase from the last submission (Method 2).

What Did We Learn?

This network overfits as the input increases. The higher score indicated to us to try to improve our neural network approach rather than to try and pursue a different classification methodology.

Method 4 (draft3.m and draft3_main.m)

Description - Code in Appendix D and F

Similarly to the previous approach, this approach is also based on a neural network. In this version, the entire training set is broken down and the program determines the ideal hidden neurons for classification. Reduction is also minimized according to memory capacity and the results are then calculated from this model.

Results

When the results were submitted on Kaggle, they received a score of 0.571293, a decrease from the previous submission.

What Did We Learn?

Large hidden neuron size does not make a net 'smarter.' The common approach of having neurons somewhere between the size of the input and output, that is, between 1 and 369, is in the majority of cases a good foundation for neural net development. As seen in the comments of draft3.m, input size was misread as training set size, which was several tens of thousands, rather than several hundred. In this case, it was likely that the neural net was overfit and furthermore possible that a complicated problem such as the customer satisfaction issue may not have been a linear one, therefore requiring a deeper network design.

Method 5 (draft4.m and draft3_main.m)

Description - Code in Appendix F and G

This method is also based on a neural network. Initially, the data was normalized but this approach did not produce any usable results, so while it remains shown in the code in comment form, normalization is not used to calculate results. The `normc()` function was attempted alongside the more specialized function that tried to normalize according to input type—binary, classification, and numeric value—but neither showed promising improvements.

This method differed from draft3 in that it attempted a deeper but narrower network approach, using two hidden layers (and later, three up to ten) rather than one. However, memory restrictions began to take their toll as soon as the earliest attempts at deeper networks. Some of the deepest networks able to be implemented without hitting the limit were [36 20 36 20 36]. Only when switching `net.trainFcn` from `'trainlm'` to the more memory efficient `'trainscg'` could deeper networks be implemented, but of the many different implementations none achieved a better result. Submissions 6 through 8 implement the new training function. Classification line was drawn at `E=mad(prune)` for the 5th submission, but for deeper nets it seemed more effective to draw the line at `E=median(prune)`.

This method was submitted an additional three times with different neural network configurations.

The 6th submission replaces `net = configure(fitnet([36 36]),'outputs',1);` with `net = configure(fitnet([36 20 36 20 36]),'outputs',1);`

The 7th submission utilizes `net = configure(fitnet([36 36 36 36 36 36 36 36 36]),'outputs',1);`

While the 8th submission uses `net = configure(fitnet([30 70 100 120 135 150 170 200 140]),'outputs',1);`

This method was also re-submitted with the original configuration of `net = configure(fitnet([36 36]),'outputs',1);` However, the line `rng('shuffle');` was added at the top of the code directly below line 3 (`close all`). This version of the code was used to create three different versions of the results.

The 9th submission was entitled AVG_SEV.csv and used a compilation of several network values to calculate the results (see Appendix I).

The 10th submission, AVG_3.csv used the first three averaged values.

The 11th submission, AVG_5.csv utilized 5 averaged values.

Results (Graphs in Appendix H)

When the results from the initial Method 5 were submitted on Kaggle, they received a score of 0.721357. This score was a significant improvement over previous scores and later became our best score overall.

Submission 6 received a score of 0.683027, submission 7 received a score of 0.692547, and submission 8 received a score of 0.668543 when the results from those programs were submitted on Kaggle.

Submission 9 received a score of 0.698776, submission 10 received a score of 0.693481, and submission 11 received a score of 0.685285.

What Did We Learn?

Deeper and bigger does not necessarily mean better. Although the network significantly improved with increased layering, as the issue approached is likely complicated and nonlinear, adding more layers doesn't improve the network very much past a certain point.
Different configurations provided different results in classification confidence both complementing and in spite of actual correlations.

Other attempts after Method 5 are listed in Appendix I.

Best Method

Our best method was the first version of Method 5, which received a score of 0.721357 on Kaggle.

Conclusion

In conclusion, though our best result did not come from our final method, we were able to try a number of approaches using a neural network. Efforts to optimize the network were initially successful in raising our accuracy but these efforts eventually tapered off. Later we discovered that Matlab uses the same seed for training neural networks, which factored into less successful efforts in later attempts.

APPENDIX

Appendix A – Method 1 Code

simple_classification.m

```
clc
clear all
close all

% ========================================================================
%                  Classification using Decision Tree
% ========================================================================

% Training.

% Read Training data.

fprintf('Reading Training Data . . .\n')
trdata = csvread('train.csv',1,0);
fprintf('\nTraining data loaded.\n')

% Separate training data.

datanum = trdata(:,1);
inputs = trdata(:,2:end-1);
target = trdata(:,end);

fprintf('\nTraining decision tree . . . \n')

% Train a decision tree.

dTree = classregtree(inputs,num2str(target));

dTree
view(dTree)

fprintf('Training Complete. \n')

% Testing.

% Load testing data.

fprintf('\nReading test data . . . \n')
tedata = csvread('test.csv',1,0);
fprintf('\nTest data loaded. \n')

testdatanum = tedata(:,1);
testinputs = tedata(:,2:end);

dataclass = dTree.eval(testinputs);
```

```
% Save Test results.

dclass_dt = str2num(cell2mat(dataclass));

result = [testdatanum,dclass_dt];

csvwrite('dectree_result.csv',result)
```

Appendix B – Method 2 Code

simple_classification1.m

```
% Separate training data.

datanum = trdata(:,1);
inputs = trdata(:,2:end-1);
target = trdata(:,end);

fprintf('\nTraining decision tree . . . \n')

% Train a decision tree.

dTree = ClassificationTree.fit(inputs,num2str(target));

dTree
view(dTree)

fprintf('Training Complete. \n')

% Testing.

% Load testing data.

fprintf('\nReading test data . . . \n')
tedata = csvread('test.csv',1,0);
fprintf('\nTest data loaded. \n')

testdatanum = tedata(:,1);
testinputs = tedata(:,2:end);

dataclass = dTree.predict(testinputs);

% Save Test results.

dclass_dt = str2num(dataclass);

result = uint32([testdatanum,dclass_dt]);

csvwrite('dectree_result.csv',result)
```

Appendix C – Method 3 Code

annclass.m

```
clc
clear all
close all

% ========================================================================
%                  Classification using Decision Tree
% ========================================================================

% Training.

% Read Training data.

fprintf('Reading Training Data . . .\n')
trdata = csvread('train.csv',1,0);
fprintf('\nTraining data loaded.\n')

% Separate training data.

datanum = trdata(:,1);
inputs = trdata(:,2:end-1);
target = trdata(:,end);

I = find(target == 0);

I1 = randi(length(I),100);
I1 = I(I1);

I = find(target == 1);
I2 = randi(length(I),100);
I2 = I(I2);
I = sort([I1,I2]);

inputs = inputs(I,:)';
target = target(I,:)';

clear('trdata')

net = newpr(inputs,target,100);
net.layers{1}.transferFcn='tansig';

net.layers{2}.transferFcn='purelin';
net.trainParam.epochs = 100;
net = train(net,inputs,double(target));

fprintf('\nReading test data . . . \n')
tedata = csvread('test.csv',1,0);
fprintf('\nTest data loaded. \n')

testdatanum = tedata(:,1);
```

```
testinputs = tedata(:,2:end);

res = sim(net,testinputs');
res = res';
res = res >= 0;
res = double(res);
```

Appendix D - Method 3 Code

annmain.m

```
clc
clear all
close all

% Neural network based solution.

load res1;

fprintf('\nReading test data . . . \n')
tedata = csvread('test.csv',1,0);
fprintf('\nTest data loaded. \n')

testdatanum = tedata(:,1);
testinputs = tedata(:,2:end);

res = sim(net,testinputs');
res = res';
res = res >= 0;
res = double(res);

result = uint32([testdatanum,res]);

csvwrite('ann_result.csv',result)
```

Appendix E – Method 4 Code

draft3.m

```
clc
clear all
close all

% Load dataset

fprintf('Reading Training Data . . .\n')
trdata = csvread('train.csv',1,0);
fprintf('\nTraining data loaded.\n')

% Separate training data.

datanum = trdata(:,1);
inputs = trdata(:,2:end-1);
target = trdata(:,end);

% Train network
% TODO?: Determine ideal hidden neurons (3rd value of newpr)
% Currently ~1/100th of the input size (76021)
% TODO: Minimize reduction according to memory capacity (N).
% 'useParallel' possibly?
net = patternnet(700);
%net.layers{1}.transferFcn='tansig';
%net.layers{2}.transferFcn='purelin';
N = 100;
net.trainParam.epochs = 1000; % (default 1000?)
net.trainParam.showWindow = false;          % no GUI, for server use
net.trainParam.showCommandLine = true;      % display in command line
net.trainParam.show = 1;                    % display every iteration
net = train(net, inputs', double(target)', 'reduction', N);

% Save trained network for future possible test use

save net;

% Performance on self
output = sim(net, inputs');
output = double(output' > 0);
%[c,cm] = confusion(target,output);
off = sum(target == output)/size(target, 1);
fprintf('\n%d\n', off);

% Performance on test

fprintf('\nReading test data . . . \n')
tedata = csvread('test.csv',1,0);
fprintf('\nTest data loaded. \n')

testdatanum = tedata(:,1);
```

```
testinputs = tedata(:,2:end);

res = sim(net,testinputs');
res = double(res' > 0);

result = uint32([testdatanum,res]);

csvwrite('ann_result.csv',result)
```

Appendix F - Method 4 Code (also used in Method 5)

Draft3_main.m

```
clc
clear all
close all

% Neural network based solution.

load net;

fprintf('\nReading Training data . . . \n')
tedata = csvread('train.csv',1,0);
fprintf('\nTest data loaded. \n')

% Separate training data.

datanum = trdata(:,1);
inputs = trdata(:,2:end-1);
target = trdata(:,end);

% Performance on self
output = sim(net, inputs');

prune = output';
i = size(prune,1);
while i > 0
    if(target(i) == 0)
        prune(i) = [];
    end
    i = i-1;
end

E = floor(100*median(prune))/100;

output2 = double(output' >= E);
%[c,cm] = confusion(target,output);
off = sum(target == output2)/size(target, 1);
fprintf('\n%d\n', off);

figure;
n_bins = 1000;
start = floor(min(output')*n_bins)/n_bins;
ending = floor(max(output')*n_bins)/n_bins;
bins = start:(ending-start)/n_bins:ending;
hist(output',bins);
h1 = findobj(gca,'Type','patch');
set(h1,'FaceColor','r','EdgeColor','k'); % Alt blue: [.4,.7,1]
hold on;
```

```
hist(prune,bins);
h2 = findobj(gca,'Type','patch');
%set(h2,'FaceColor',[.4,.7,1],'EdgeColor','k');
hold off;

% Performance on test

fprintf('\nReading test data . . . \n')
tedata = csvread('test.csv',1,0);
fprintf('\nTest data loaded. \n')

testdatanum = tedata(:,1);
testinputs = tedata(:,2:end);

res = sim(net,testinputs');
res = double(res' >= E);

result = uint32([testdatanum,res]);

csvwrite('ann_result.csv',result)
```

Appendix G - Method 5 Code

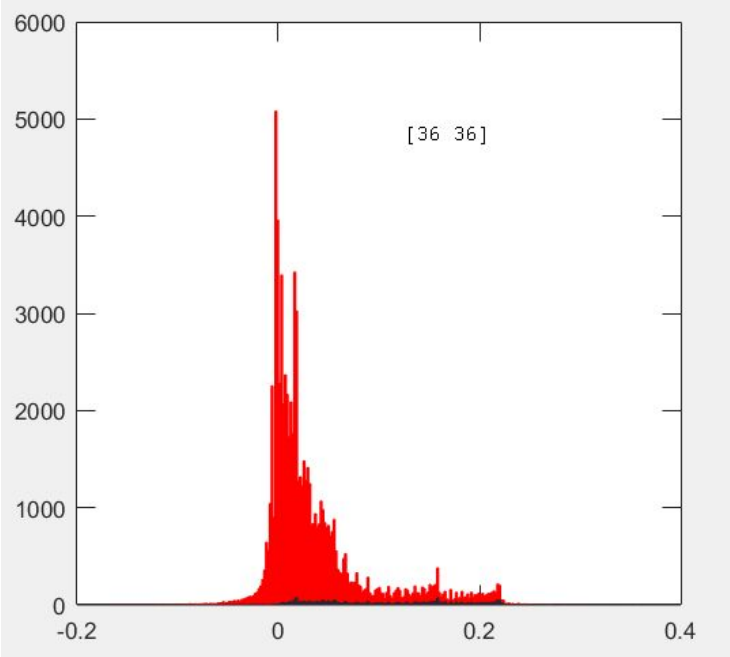
draft4.m

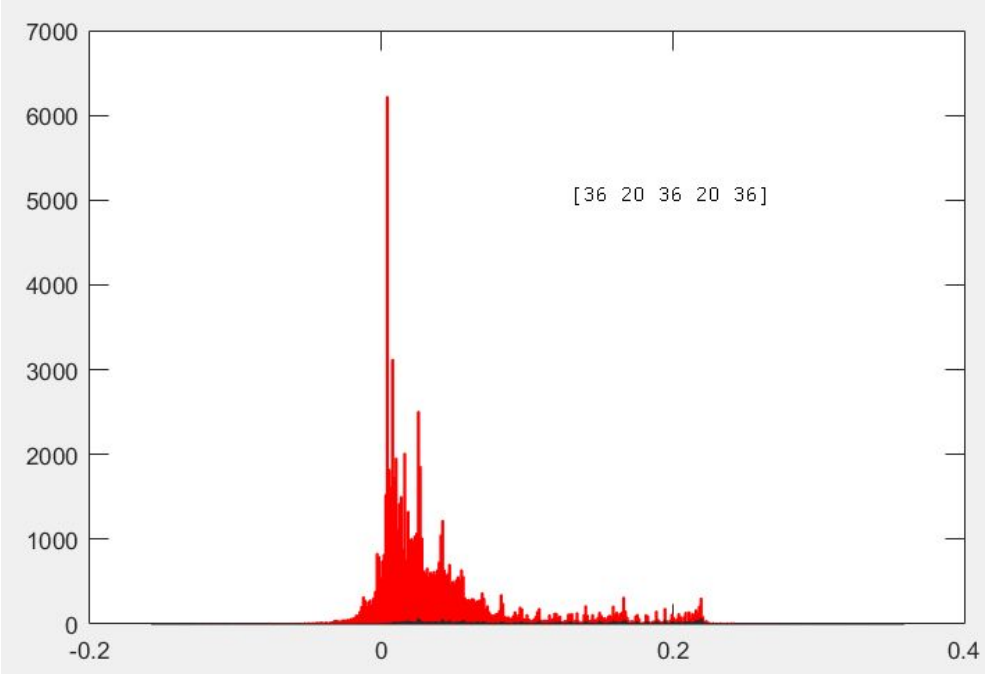
```
clc
clear all
close all

% Load dataset

fprintf('Reading Training Data . . .\n')
trdata = csvread('train.csv',1,0);
fprintf('\nTraining data loaded.\n')

% Separate training data.

datanum = trdata(:,1);
inputs = trdata(:,2:end-1);
target = trdata(:,end);

% Normalize training data.

% uniques = zeros(1,size(inputs,2));
% norm_id = zeros(1,size(inputs,2));
% for i = 1:size(uniques,2)
%     uniques(i) = size(unique(inputs(:,i)),1);
%     if(uniques(i) == 1)
%         norm_id(i) = 0;                       % Null, do nothing
%     elseif(uniques(i) == 2)
%         if(sort(unique(inputs(:,i))) == [-1;1])
%             norm_id(i) = 0; % Already normalized
%         else
%             norm_id(i) = 1;                   % Binary
%         end
%     elseif(std(inputs(:,i)) < 1) %temp value
%         %norm_id(i) = 2;                      % Categorical
%         norm_id(i) = 3; % Currently avoiding trying to recognize discrete
%     else
%         norm_id(i) = 3;                       % Continuous/Numeric
%     end
% end

%Normalization of inputs:

%normalized = normc(inputs);

% normalized = inputs;
% for i = 1:size(inputs,2)
%     switch norm_id(i)
%         case 0
%             % Do nothing
%         case 1
%             % Replace the lesser value with -1, the greater with 1
```

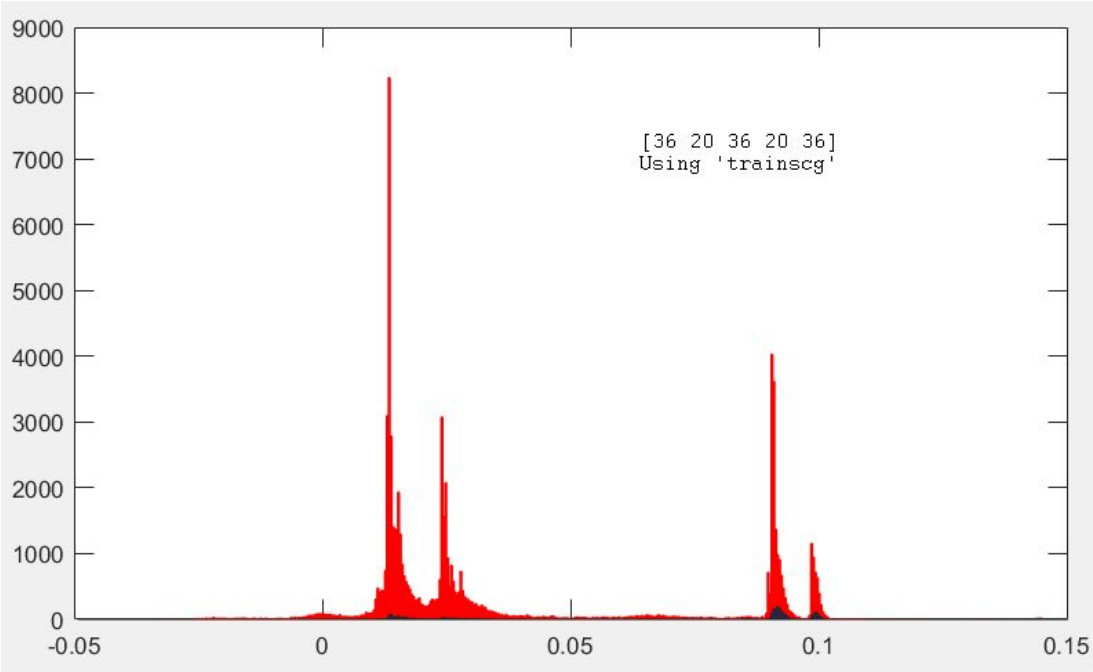
```
%             pos = max(inputs(:,i));
%             normalized(:,i) = (inputs(:,i)==pos) + -1*(inputs(:,i)~=pos);
%         case 2
%             % Apply 1-of-(C-1) categorization to case
%             cats = zeros(size(unique(inputs(:,i))));
%             cats(end) = []; %Prune one dimension
%             normalized(:,i) = zeros(size(normalized(:,i)));
%             temp_dim = 1;
%             for j = sort(unique(inputs(:,i)))
%                 if(inputs(j,i)==median(inputs(:,i)))
%                     cats = ones(size(cats)) * -1;
%                 else
%                     cats(temp_dim) = 1;
%                     temp_dim = temp_dim + 1;
%                 end
%                 normalized(:,i) = normalized + (cats * (input(:,i) == j));
%                 cats = zeros(size(unique(inputs(:,i))));
%             end
%         case 3
%             % Apply median-and-MAD normalization
%             p = median(inputs(:,i)) * ones(size(inputs(:,i)));
%             q = mad(inputs(:,i));
%             normalized(:,i) = (inputs(:,i)-p)/q;
%     end
% end

% Train network
% TODO?: Determine ideal hidden neurons (3rd value of newpr)
% 'useParallel' possibly?
net = configure(fitnet([36 36]),'outputs',1);
% net.numLayers = 3;
%net.layers{1}.transferFcn='tansig';
net.biases{1}.learnFcn='trainscg';
net.biases{2}.learnFcn='trainrp';
% net.layers{2}.size=36;
net.biases{3}.learnFcn='purelin';
net.layers{3}.size=1;
net.trainParam.epochs = 100;                        % (default 1000)
net.trainParam.showWindow = false;                  % no GUI, for server use
net.trainParam.showCommandLine = true;              % display in command line
net.trainParam.show = 1;                            % display every iteration
net = train(net, inputs', double(target)', 'showResources', 'yes');

% Save trained network for future possible test use

save net;

% Performance on self
% TODO: Replace linear classification with maybe a real predictor
output = sim(net, inputs');

prune = output';
i = size(prune,1);
while i > 0
```

```
    if(target(i) == 0)
        prune(i) = [];
    end
    i = i-1;
end

E = sum(prune)/size(prune,1);

output2 = double(output' >= E);
%[c,cm] = confusion(target,output);
off = sum(target == output2)/size(target, 1);
fprintf('\n%d\n', off);

% Histogram of neural network results + classification of training set
% Turn off for use in console, no GUI

graph = 1;
if graph
    n_bins = 1000;
    start = floor(min(output')*n_bins)/n_bins;
    ending = floor(max(output')*n_bins)/n_bins;
    bins = start:(ending-start)/n_bins:ending;
    hist(output',bins);
    h1 = findobj(gca,'Type','patch');
    set(h1,'FaceColor','r','EdgeColor','k'); % Alt blue: [.4,.7,1]
    hold on;
    hist(prune,bins);
    h2 = findobj(gca,'Type','patch');
    %set(h2,'FaceColor',[.4,.7,1],'EdgeColor','k');
    hold off;
end

fprintf('\nReading test data . . . \n')
tedata = csvread('test.csv',1,0);
fprintf('\nTest data loaded. \n')

testdatanum = tedata(:,1);
testinputs = tedata(:,2:end);

res = sim(net,testinputs');
res = double(res' >= E);

result = uint32([testdatanum,res]);

csvwrite('ann_result.csv',result)
```

Appendix H - Graphs From Method 5






[36 20 36 20 36]
Using 'trainscg'
9000
8000
7000
6000
5000
4000
3000
2000
1000
0
-0.05
0
0.05
0.1
0.15

Appendix I - All Submitted/Unsubmitted Network Attempts (After draft4.m)

AVG_SEV.csv was compiled using these networks. Regarding [36 20 36 20 36], Take 1 was counted twice because of a redundant save, and Takes 2 and 3 produced the same network and so one of them could be said to have been counted twice as well.

Trained without trainscg:

[36 20 36 20 36]

[36 10 10 10 36]

[36 28 20 12 4]

[30 30 30 30 30]

Trained with trainscg:

[80 80 80 80 80]

Biases disabled:

[36 20 36 20 36], Take 2 (Submission 6)

[369 150 150 150 150]

[150 75 150 75 150]

Biases reenabled - discovering they do nothing:

[3 36 20 36 20 36], Take 3

Further attempts:

[75 75 75 75 75]

[100 36 36 36 36]

[36 20 36 20 36 20 36]

[36 36 36 36 36 36 36 36 36] (Submission 7)

[180 180 180 180 180 180 180 180 180]

[180 120 60 40 36 40 60 120 180]

[180 12 120 36 90 36 120 12 180]

[180 24 180 24 180 24 180 24 180]

[180 24 36 24 36 24 36 24 180]

[180 24 36 24 36 24 36 24 36]

[180 160 140 120 100 80 60 40 30]

[30 70 100 120 135 150 170 200 240] (Submission 8)

Appendix J - Team Member Contributions

Namho An

Wrote Methods 1, 2, and 3 and edited Methods 4 and 5 and the project paper.

Ariel Brest

Wrote the project paper and edited the methods.

Shaul Vin

Wrote Methods 4 and 5 and edited Methods 1, 2, and 3 and the project paper.