Technical Report on SVM Analysis of Stability Criterion of Random Matrices by Christopher Ibarra

This report concerns itself with the determining the classification of five parameters of an ecological system into yielding either stable or unstable asymptotic behavior of the system. Code was developed to accomplish this task. These five parameters are N the size of the ecological system, d the order of magnitude of the population of a species by the growth rate over the carrying capacity, delta (δ) the small variation in d, C the proportion of predator-prey relations, and sigma (σ) the variation in the population size of a species times the coefficient pertaining to the predator-prey relation of the species and another.

Overall, the five control parameters N, d, delta, C, and sigma are empirical values that determine a community matrix representing the ecological system. These matrices are produced trough a Monte Carlo simulation, and many samples of these matrices are produced to determine the stability the five parameters yield. However, other parameters were used instead to compute the five control parameters. More is explained in section one. Section two discusses the SVM algorithm developed to predict stability given a trial set of the parameters.

1. Generating Data

The five parameters N, d, delta, C, and sigma are used compute a community matrix M in the following manner

```
//$M_{ii}$ ~ N(-d,delta^2)
    //For all i < j
    //    with possibility C/2 generate $M_{ij}$ ~ |N(0,sigma^2)| and $M_{ij}$ ~ -|N(0,sigma^2)|
    //    with possibility C/2 generate $M_{ij}$ ~ -|N(0,sigma^2)| and $M_{ji}$ ~ |N(0,sigma^2)|
    //    with possibility 1 - C set $M_{ij}$ = $M_{ji}$ = 0
```

where the subscripts indicate the element coordinates of the matrix M. This criteria is implemented in the C code file GenerateM.c as the function void GenerateM(double** pptrM, int N, double d, double delta, double sigma, double C).

The normal distributions from which random values are take for the matrix elements are implemented inside two C code files DiagRandomNormal.c and RandomNormal.c as functions void DiagRandomNormal(double mean, double stdDev,double* X, double* Y) and void RandomNormal(double mean, double stdDev,double* pos, double* neg). The function DiagRandomNormal() determines values for the diagonal entries of the matrix M given the mean -d and the standard deviation delta. Similarly, the function RandomNormal() gives values to the off-diagonal elements given a mean of 0 and a standard deviation of sigma. Both functions are used within GenerateM(). The Box-Muller transform combined with shift and scaling is used to produced the normal sampling in C.

Given these functions GenerateM(), DiagRAndomNormal(), and RandomNormal() and feeding the parameters to GenerateM(), a communit matrix stored in an array is determined, noting that M is a square matrix with N as it's side length.

To determine the stability of the system the community matrix M represent, one has to determine the eigenvalues of M. If the real part of all the eigenvalues of M is non-positive, then M represents a stable system, otherwise an unstable system. For this purpose, two functions were used. To determine the eigenvalues of M the function void dgeev_(char*,char*,int*,double*,int*,double*,double*,double*,int*,double*,int*,double*, int*,int*) from LAPACK was used. It is exported from the LAPACK library. Feeding the appropriate arguments to this function yields as one of its outputs a 1 by N array containing the real part of all the eigenvalues of M. Given these, another function was developed examineEigensReal(double *b, int N) contained within the file examineEigensReal.c. This function returns a 1 if the eigenvalues denote a stable M anda 0 otherwise.

Given the random assignment of the entries of M from a normal random distribution, many samplings of the five parameters fed into GenerateM() must be taken to determine if the parameter set denote a stable M. For this purpose, two more parameters are introduced, N_{mc} and etha_s (η_s). The data analysis parameter N_{mc} determines how many randomly generated M are produced from the five control parameters. Afterwards, the functions dgeev_() and examineEigensReal() are used to determine whether to add to a counter keeping track of the number of community matrices that were stable from the five control parameters used.

Afterwards, the control parameter etha_s is used as a benchmark to determine if the five control parameters yield stability or not. If the proportion of matrices, that is counter/N_{mc}, is greater than or equal to etha_s, then the five parameters are deemed stable, otherwise they are deemed unstable. An output parameter t is assigned to each set of input control parameters. The value of t is 1 if the control parameters are stable and -1 if they are unstable.

For the purposes of generating multiple data points ({N,d,delta,C,sigma},t), numerous runs of the algorithm so far described are enacted. The number of runs is divided into two sets. Set one is named the training set and stores N1 data points. The second set is the testing set and contains N2 data points. N1 is about twice N2. These data points are generated using for loops and OpenMP as seen in the main program contained in the file main.c.

Given this purpose, the values of the input parameters set {N,d,delta,C,sigma} are randomly determined from a uniform distribution. The function accomplishing this is void randomParams(int N_L,int N_U,int *ptrN,double d_L,double d_U,double *ptrd,double delta_L,double delta_U,double *ptrdelta, double C_L, double C_U, double *ptrC, double sigma_L, double sigma_U, double *ptrsigma) and it is contained inside the file randomParams.c.

As can be seen from the function arguments, randomParams() takes in lower and upper bounds for each of the five control parameters. A random value between these bounds is generated from a uniform distribution and assigned to the corresponding parameter. These data generating parameters are define as follows:

N_L: lowest value N can take

N_U: highest value N can take

d_L: lowest value d can take

d_U: highest value d can take

delta_L: lowest value delta can take

delta_U: highest value delta can take

C_L: lowest value C can take

C_U: highest value C can take

sigma_L: lowest value sigma can take

sigma_U: highest value sigma can take

All the data generating parameters and the data analyzing parameters are contained within the text file parameters.txt in the following order and configuration:

N_L

N_U

d_L

d_U

delta_L

delta_U

C_L

C_U

sigma_L

sigma_U

Nmc

etha_s

N1

N2

The main function reads the value from this text file. For producing variations and analyzing the data in a different manner, the values in this file have to be changed accordingly in the right position.

From this, each training set data point is stored in the text file training_set.txt. Similarly, every testing set data point is stored in the text file testing_set.txt. The manner in which the five input parameters and output parameters are stored is as follows:

N d delta C sigma t

where training_set.txt contains N1 of these lines and testing_set.txt N2, as previously alluded.

The purpose of the training set is to develop a support vector machine (SVM) classifier that will try to predict the value of t given a set of the control parameters. The testing set is used to determine the accuracy of the SVM classifier.

The executable implementing all the function mentioned so far is generated using the build.sh file. The executable is run with the run.sh script. The executable is run optimally with 4 processors. The run.sh script also contains the SVM Python implementation described in the next section.

2. SVM Algorithm

The SVM is implemented with a soft-margin in Python. The parameter that determines the sof-margin is C.

The Python code is split into two files: py_readdata.py and svm.py. The file py_readdata.py contains a function called read_data(filename). All this function does is read the training_set.txt and testing_set.txt files to store into lists. The numpy module is used to convert the lists into arrays. The input parameters and the output parameter are stored into different arrays. For the training set, these are arrays x and t. For the training set, these are arrays x_test and t_test.

For the SVM implementation, the necessary matrices to define the problem in standard form to feed into the cvxopt module are as follows:

```
C = 0.5

q = -np.ones((n,1));

K = np.zeros((n,n))

for i in range(0,n):

  for j in range(0,n):

    K[i,j] = Kernel(x[i,:],x[j,:])

P = t*np.transpose(t)*K;

A = t.reshape(1,n);

b = 0.;

G = np.concatenate((np.eye(n),-np.eye(n)));

h = np.concatenate((C*np.ones((n,1)),np.zeros((n,1))));
```

where n is the number of data points in the training set. The kernel function used for the SVM is the following: $(1+\mathbf{x}^T\mathbf{y})^s$ where for this implementation, s = 2. This function is defined in the svm.py file as the function Kernel(x,y) that takes in two vector arrays. It is used to compute the K array as shown above.

After using the cvxopt module to compute the Lagrange multipliers, the classifier is constructed, this in the same file. The classifier is define as the function L(x_test_i,x,Lambda,t,Kernel,b_hat), where b_hat is an array vector constructed as per the SVM algorithm specifications. The other arguments are named after the arrays previously defined for matching in the argument list.

The classifier takes in set of control parameters in x_test and inputs it in the place of x_test. This is done for all the control parameter sets in x_test and the output is collected in a list called result. Finally, the comparison of the entries in result and t_test, matched by same entry index, is done in a for loop. If the quotient of result[i] and t_test[i] is non-negative, then it indicates correct prediction by the classifier a counter is increased by one. At the end of the for loop, the value of the counter represents the number of correct predictions by the classifier L().

Finally, the fraction of correct predictions is computer by counter/n_test, where n_test is the number of data points in the testing set. For the kernel used in the Kernel() function, a minimum score of 0.95 has been reported.