`

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

DATA MINING

(8900-5-R1)

Date: April 3, 2019

“Self-Organizing Map”

(C-17)

Instructor:

Dr. Roozbeh Razavi Far

Submitted by:

Siddhartha Tamal Taru (105017813)

Vedant Dave (105101020)

University of Windsor

ON, CANADA

ABSTRACT

Computer and technology are the most important part of our current life and there has been no issue to ignore that. Nowadays in our professional and personal life, we have to face lots of data and information and we have to make proper use of them. Data mining is such kind of subject what makes our life easier to deal with this enormous kinds of data. Data mining is widely used nowadays in those fields where there has use of big data and statistics, such as healthcare, finding out customer behavior in the market, in education system finding out student's best choice for future career, in industries by finding out the relationship between product and consumers, and so on. Overall we can say, wherever there has been a use of good amount of datasets, there has been the use of "Data Mining" to bring more accuracy and efficacy in work. When dealing with a big amount of datasets, it is very common to see missing data or noisy data in them and they create barrier in finding the desired result. Missing values of a dataset decreases the efficiency of performance creating the computational complexity and distracting the final output from the original result. The goal of "Data Mining" is to deal with those missing values or noisy value by estimating them with a reliable logic. When there has been missing values in dataset, estimating and filling up those values are important to bring better efficiency in work. Missing values from the datasets should be replaced by appropriate data so that they cannot do any obstacle in the process. Those values can be imputed by using some methods which are almost near to the real value. There has been several methods in data mining to estimates missing values from the datasets. Among them some common methods are "K-nearest neighbors (KNN)", " K-mean clustering" , "local least square regression (LLS)" , " Self-organizing map (SOM)" and so on. In this project, missing values from dataset is estimated by the algorithm of " Self- Organizing Map (SOM)" and also analyzed the computational complexity and effectiveness of the algorithm for the imputed missing values.

Keywords

Data mining, Missing Data Imputation, Neighboring function, Self-organizing map, Clustering, Computational Complexity, Euclidean distance, Winning Data set.

Development Software

MATLAB R2018b & MATLAB R2014a

INTRODUCTION

As day by day our life is getting dependable on the use of computer, the use of data mining is getting more common to our life. Every day we have to face big amount of data, like in our computer desktop, grocery stores, in our school or university, at our work and even when we play games. Most of those data are raw data and it's hard to find out best information and proper use from them. Raw data always have some missing values, noisy values or some false values that can confuse us. Data in our dataset can be missed for various reasons and that's why a proper use of imputation method is required for this to get proper output.

"Kohonen self–organizing future maps" or "**Self organizing map**" is one of the method based on cluster analysis, which is widely used for big dataset analysis for neural network. Self-organizing map illustrate its high dimensional source taking help from its low dimensional space and it works in both 2D and 3D models [1]. Although the use of self-organizing map is mostly used in the neural network field, but it is also common in some financial organizations where there has been use of massive data, in industries where there are use of big data entry, statistical organizations in analysis of survey reports, image processing and identifying spam email data for security.

Self-organizing map can work with several variables in dataset as well as it uses nonparametric regression predicting data from its own datasets [2]. Self-organizing map works on the concept of clustering with unsupervised data. It assumes the initial data randomly and after assumption it checks the distance between predicting values for every data. It measures the distance between those data with similar characteristics and plot them as matching units and thus it creates the clustering with the similar data. Self-organizing map not only can estimate missing value in datasets but also it handle noisy data from the datasets to remove the corrupted information from the clustering assumption [3].

Self-organizing map does not require any supervised data for its imputation. It also have less computational complexity when there has big dataset. The imputation by self-organizing highly depends on frequent inputs. The imputation depends on parameter input and sometimes it gives error if the parameter is not given in proper way. Also when there are enormous amount of data, it takes long time to impute those data making long time complexity.

Implementation Procedure:

- For this project we use the **MATLAB** software with two different version **R2018b** and **R2014a** for the different type of missing data imputation and their result analysis based on error rate and accuracy of data.
- We use some MATLAB logical code functions with mathematical formulas and also use some predefined function to get easier output for desire condition given form the instructor.
- We can handle the numerical dataset and for handling categorical dataset we use the **categorical to numerical conversion** coding for getting best AE value for higher accuracy.
- We use the coded matrix format to import given dataset, which help to handle massive data with Self organizing mapping Algorithm.
- First we need the **training dataset** form given missing dataset by removing the missing data rows and use the training dataset for the further process in algorithm.
- After that we defined the required **parameters** for reliable output such as learning rate, neighboring function width, time constant , iteration and mapping size sample's rows & columns for defining the SOM architecture.
- Initializing the weights for the given input dataset for finding the winning data point and also try to adjust the Weight based location of that winning data point in SOM mapping.
- First with the use of neighboring function for finding distance of each point with '**Euclidean_Distance'** code and then use the competitive process to compare all values.
- By the use of competitive process, we can get the BMU (Best matching unit) and get the winning data set with Cooperative process.
- Adaptive Process is used in algorithm to update the value of winning data set. With **'update_weight'** function we get the new weight of all data points and reuse them for getting precise value through clustering
- SOM generally use the clustering fundamentals, and for getting best output, algorithm need to repeat the procedure with number of iterations until the negligible significant change in weight function.
- Algorithm give imputed dataset at last and by comparing it with the original dataset we can get the value for **NRMS** (for numerical dataset) and **AE** (for Categorical dataset).
- For categorical dataset we need to covert the numerical result into desired categorical format.
- By using predefined '**Plot**' function we can get the selforganizning map.

Key MATLAB files used for Program:

We use the following code files for getting our result. This files are provide by professors to us for getting the result of provided dataset.

1. Aggregation.txt: It's a training dataset file from missing data set for SOM learning algorithm

2. main.m: This file contain the main code for handling the SOM algorithm for handling the Numerical dataset. This file automatically call its sub function which required during the program execution

3. findBestMatch.m: Its use for finding the best value from the Euclidean distance getting after the each dataset distance measurement. And define the smallest value as winning dataset (neuron).

4. randInitializeWeights.m: This file start the clustering with the initializing random dataset weight. But for the identical answer we can use the rng('default') function.

5. computeNeighbourhood.m: This code compute the neighboring function between the winning dataset and other dataset on SOM.

6. updateWeight.m: This code use for adaptive process to update the weight of all data points with new winning data weight.

7. plotData.m: Plot the training clustering dataset for the further analysis of output.

8. Lalith_boss.m: This file use for modification of main program for handling the categorical dataset.

9. nrms.m: It contains the mathematical function of finding error rate between the imputed dataset (trined_dataset) to original given dataset.

Key Parameters for getting the value:

1. **SOM Mapping Size (**Rows and columns**):** Need to change the size of row and column to get the output with more accuracy. Big dimension give good accuracy but also take more time for execution of code.
2. **No. of Iteration (t) :** More No. of iteration give more precise value due to updating weight vector after each iteration but take considerable time and space of device disk memory.
3. **Initial Width(σ)** : Effective initial value can give accurate answer with less time and less no. of iteration. We initially found it as **1** for most of dataset.

4. **Learning rate (η):** Learning rate make significant change to define the time constant for the next width and updated weight vector.

Algorithm Steps:

Here “**i”** and “**j”** are the two data points on the plane. And the “**I**” is taken as a reference dataset.

1. **Initialization (Random Initialization)**

- First choose the initial weight vector **$\mathbf{W_j(0)}$** ,randomly from the training dataset from the given missing dataset.(X) with dimension (m x n)

2. **Sampling**

- Drawing a sample of x (input dataset) using the input vector.
- Sampling must use a certain probability
- The dimension of sampling is equal to population size m.

3. Matching Similarity (Competitive Process)

- With the help of minimum distance criterion, finding out best sample of i(x) for the time interval of n.

$$\boldsymbol{i(x) = arg\ \ minj||x(n) - Wj||, \quad J = 1, 2, \dots, l}$$

4. Updating the sample (Adaptive Process)

- Synaptic weight vector $w_j(n+1)$ is updated using learning rate parameter $\eta(n)$ and neighboring function $h_{j,i(x)}$(n)
- Neighboring function are selected from nearby $i(x)$ vector.

$$\boldsymbol{w_j(n+1) = w_j(n) + \eta(n) h_{i,j(x)} \left(x(n) - \omega_j(n)\right)}$$

5. Continuation (Repetition Process)

- The process must continue and start from step 2 (sampling) until step 4 (updating) until negligible significant changes are observed in feature map.

MATLAB functions used in code:

- **“rng('default');”** =for getting same output every time to remove the effect of random weight initialization.

- **“[dataRow, dataCol] = size(train_data);”=** for giving training data from the main given incomplete matrix dataset and give size of training dataset dimension.

- **“t_width = Iteration/log(width_Initial);” = define the constant for defining neighboring size.**

- **“t_learningRate = Iteration;”**= define time constant for the learning rate.

- **“somMap = randInitializeWeights(somRow,somCol,dataCol);”=** for taking initial weight vector of neuron.

- **“plotData(train_data, y);”** = plot training data.

- **“[euclideanDist, index] = findBestMatch(train_data, somMap, somRow, ... somCol, dataRow, dataCol);”**= For Computing the Euclidean distance between each dataset and input dataset.

- **“[minM,ind] = min(euclideanDist(:));” =** Find minimum distance from all data-set.

- **“[win_Row,win_Col] = ind2sub(size(euclideanDist),ind);”=** finding the winning column and winning rows.

- **“neighborhood = computeNeighbourhood(somRow, somCol, win_Row, ... win_Col, width_Variance);”** = winning dataset compute the neighboring topological function of each data set.

- **“somMap = updateWeight(train_data, somMap, somRow, somCol, ... dataCol, index, learningRate, neighborhood);”=** Update the weight of all dataset for next iteration.

- **“[missr missc] = find(isnan(train_data));”=** Find the value of missing rows and columns from trainining dataset for the NAN value.

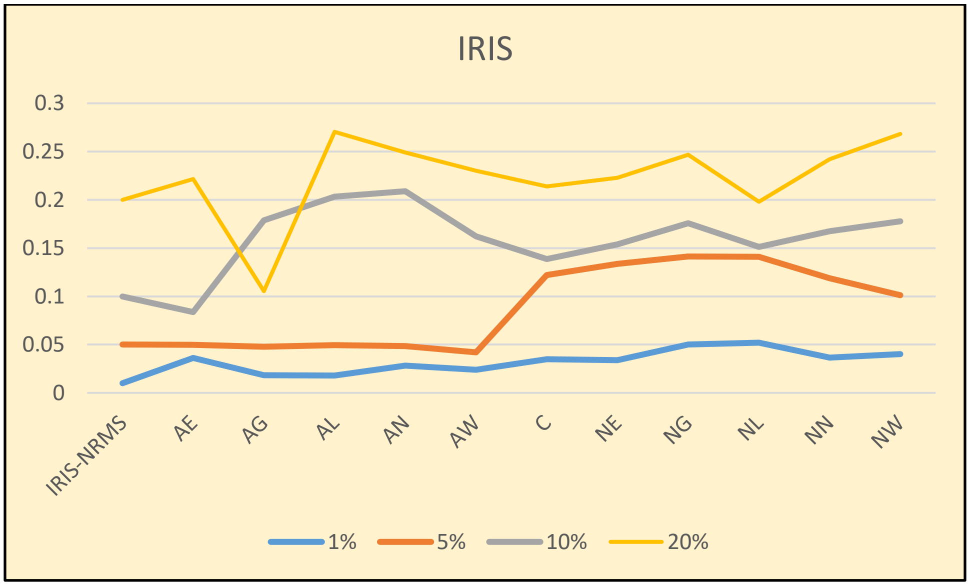
- **“nrms(train_data,original)”=** Find the NRMS value for the dataset by comparing the training data set with original dataset.

NRMS Calculation for Numerical datasets

The calculated NRMS and AE values for given datasets are shown below:

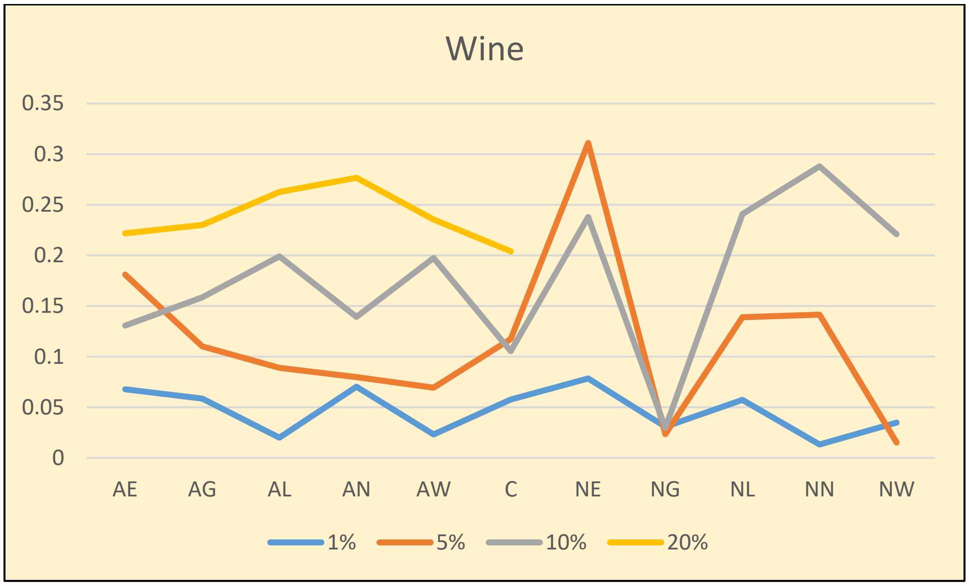
IRIS-NRMS: (Avg. NRMS: 0.1281)

IRIS-NRMS	1%	5%	10%	20%
AE	0.0363	0.0499	0.0837	0.2215
AG	0.0182	0.0479	0.1789	0.1054
AL	0.0181	0.0496	0.2032	0.2704
AN	0.0281	0.0485	0.2089	0.2489
AW	0.0238	0.042	0.162	0.23
C	0.0348	0.1221	0.1387	0.2139
NE	0.0339	0.1337	0.154	0.2229
NG	0.0501	0.1413	0.1756	0.2467
NL	0.052	0.141	0.1511	0.198
NN	0.0365	0.1187	0.1675	0.2419
NW	0.0401	0.1011	0.1777	0.2681



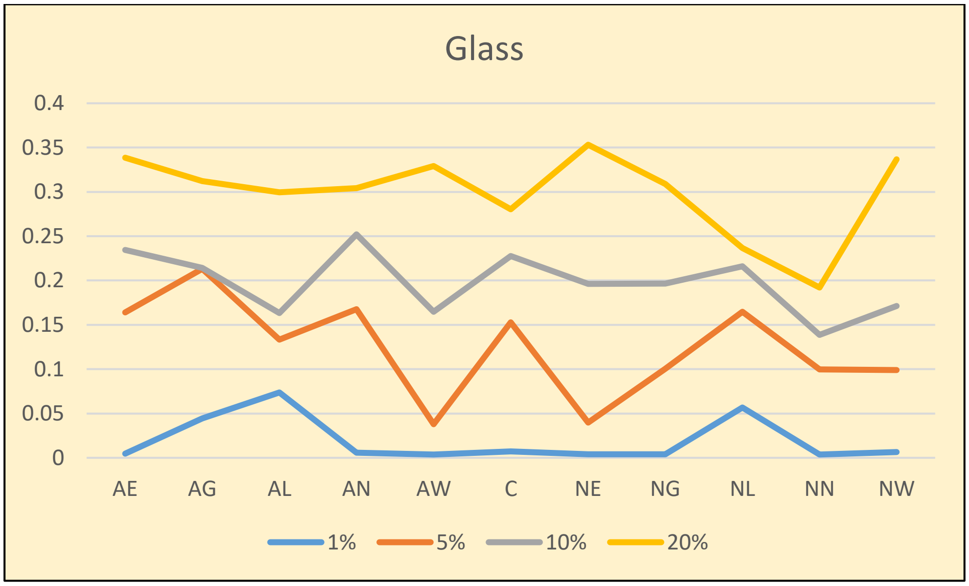
WINE-NRMS: (Avg. NRMS: 0.1360)

Wine-NRMS	1%MV	5%MV	10%MV	20%MV
AE	0.0679	0.1809	0.1307	0.2218
AG	0.0586	0.1102	0.1584	0.2301
AL	0.0202	0.089	0.199	0.2627
AN	0.0704	0.0799	0.1392	0.2766
AW	0.0233	0.0693	0.1974	0.2353
C	0.0578	0.1174	0.1054	0.204
NE	0.0785	0.311	0.2379	NAN
NG	0.0307	0.0235	0.0296	0.2739
NL	0.0575	0.139	0.2409	NAN
NN	0.0133	0.1415	0.2879	NAN
NW	0.035	0.0152	0.2207	NAN



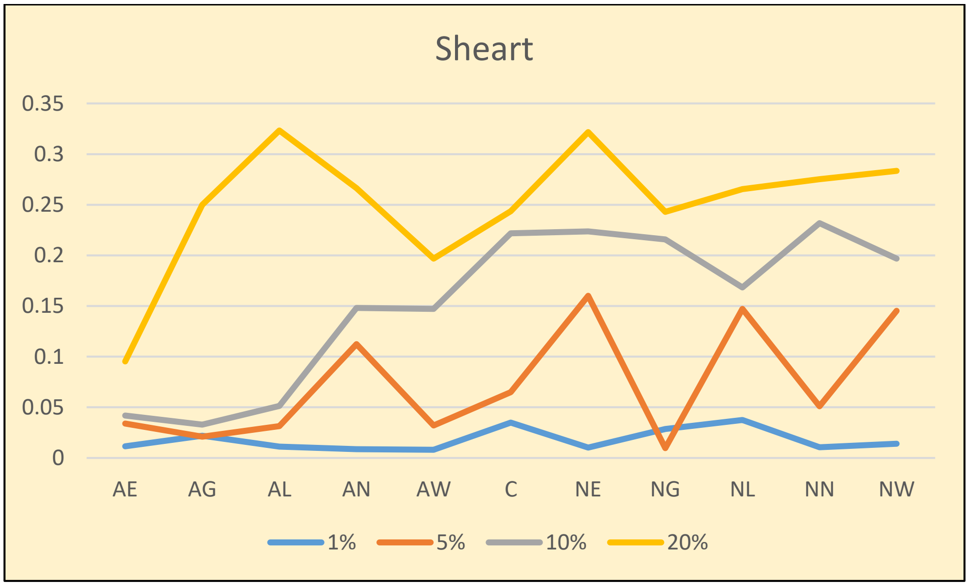
GLASS- NRMS: (Avg. NRMS: 0.1603)

Glass-NRMS	1%	5%	10%	20%
AE	0.0047	0.1642	0.2346	0.3384
AG	0.0443	0.2132	0.2141	0.3121
AL	0.0737	0.1332	0.1633	0.2996
AN	0.0059	0.1675	0.2521	0.3044
AW	0.0037	0.0379	0.1648	0.329
C	0.00734	0.153	0.2278	0.2804
NE	0.0041	0.0396	0.1961	0.3532
NG	0.0041	0.1004	0.1965	0.3091
NL	0.0569	0.1648	0.216	0.2365
NN	0.0037	0.0997	0.1387	0.1921
NW	0.0064	0.0991	0.1712	0.3366



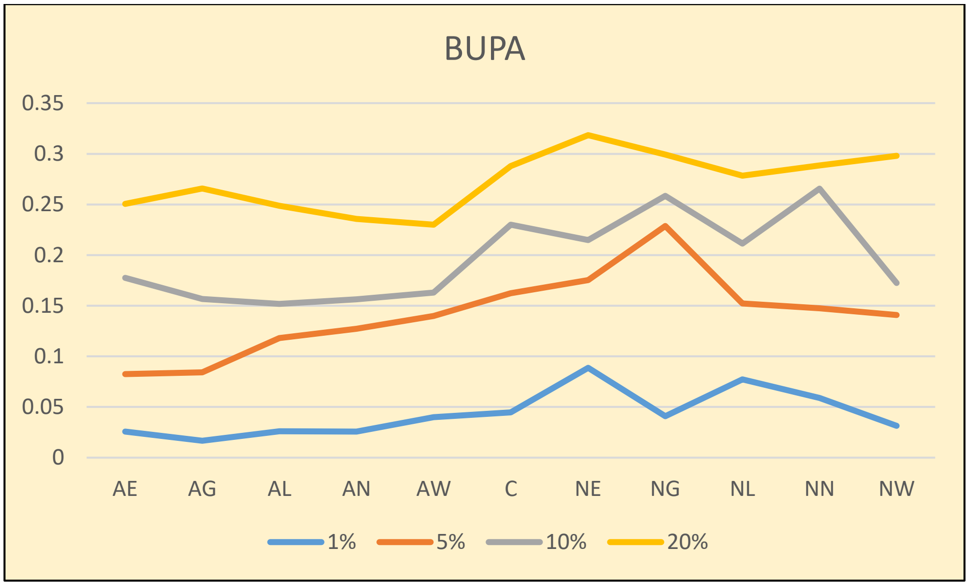
Sheart-NRMS: (Avg. NRMS: 0.1235)

Sheart-NRMS	1%	5%	10%	20%
AE	0.0116	0.034	0.0419	0.0953
AG	0.022	0.021	0.0329	0.2498
AL	0.0113	0.0314	0.0512	0.3233
AN	0.0087	0.1124	0.1482	0.2665
AW	0.0081	0.032	0.1472	0.1969
C	0.0348	0.0651	0.2217	0.2435
NE	0.0104	0.1601	0.2238	0.3217
NG	0.0287	0.0097	0.2159	0.243
NL	0.0375	0.1471	0.1683	0.2653
NN	0.0107	0.0509	0.2318	0.2751
NW	0.014	0.1454	0.1968	0.2835



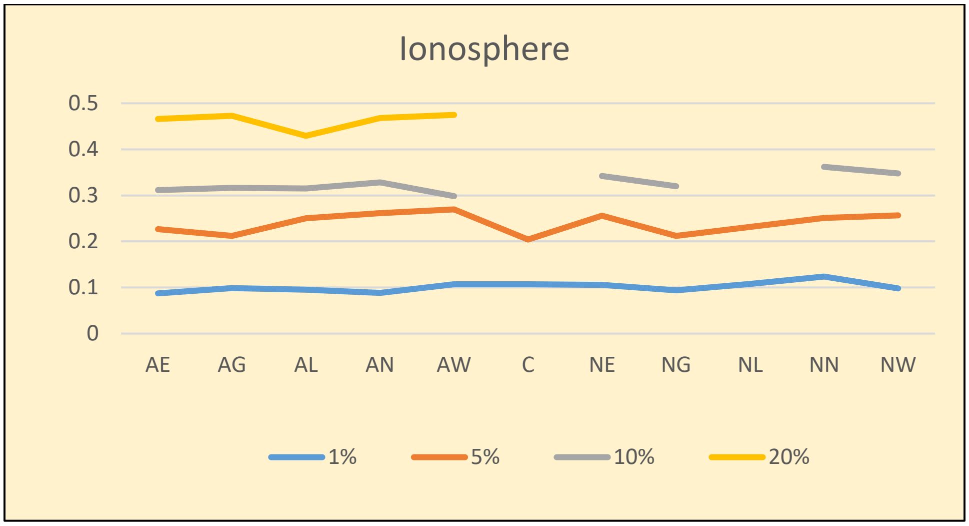
BUPA-NRMS: (Avg. NRMS: 0.1634)

BUPA-NRMS	1%	5%	10%	20%
AE	0.0258	0.0825	0.1775	0.2504
AG	0.0167	0.0841	0.1566	0.2657
AL	0.026	0.1182	0.1518	0.2488
AN	0.0258	0.1273	0.1564	0.2358
AW	0.0399	0.1398	0.1629	0.23
C	0.0447	0.1622	0.23	0.2878
NE	0.0887	0.1754	0.2149	0.3184
NG	0.041	0.2287	0.2583	0.2991
NL	0.0772	0.1523	0.2115	0.2784
NN	0.059	0.1474	0.2656	0.2884
NW	0.0315	0.1407	0.1724	0.2981



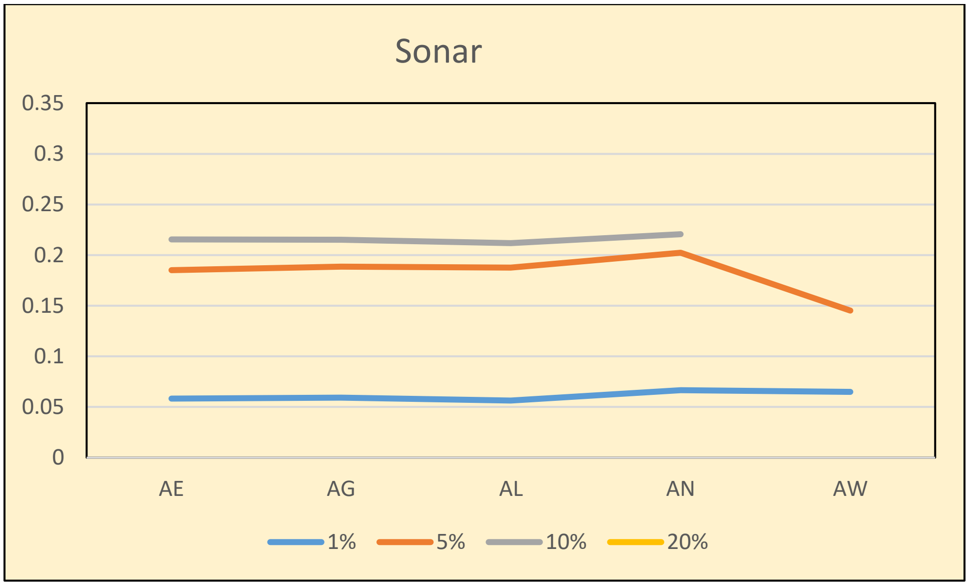
Ionosphere-NRMS: (Avg. NRMS: 0.2499)

Ionosphere-NRMS	1%	5%	10%	20%
AE	0.0873	0.227	0.3118	0.4661
AG	0.0986	0.2124	0.3166	0.4729
AL	0.0952	0.2501	0.3151	0.4293
AN	0.0884	0.2617	0.3279	0.468
AW	0.1069	0.2696	0.2984	0.4748
C	0.1071	0.2042		
NE	0.1058	0.2562	0.3424	
NG	0.0937	0.2119	0.3198	
NL	0.1079	0.2319		
NN	0.1238	0.251	0.3619	
NW	0.0979	0.2563	0.3478	



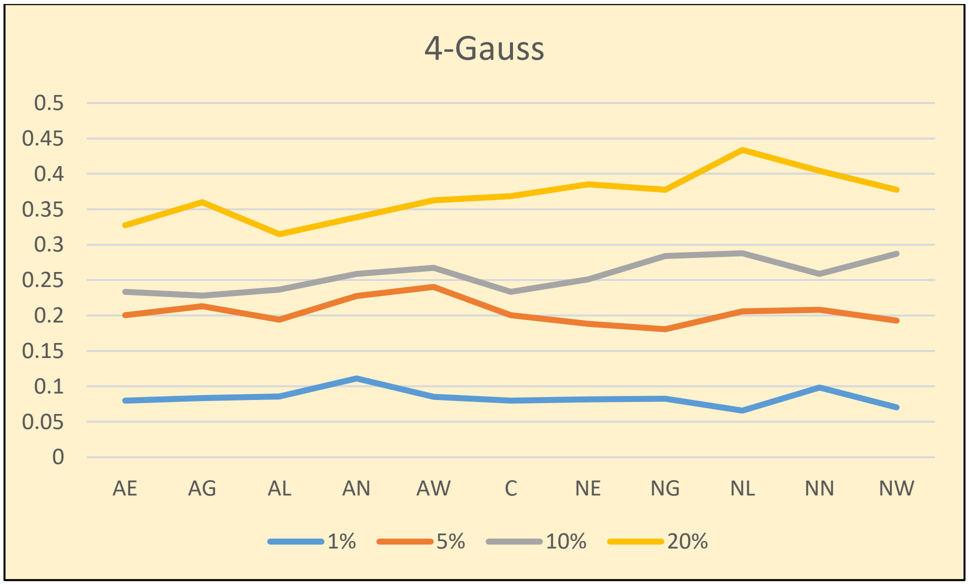
Sonar-NRMS: (Avg. NRMS: 0.1458)

Sonar-NRMS	1%	5%	10%	20%
AE	0.0582	0.1852	0.2156	0.2992
AG	0.0592	0.1886	0.215	
AL	0.0564	0.1875	0.2118	0.3191
AN	0.0666	0.2023	0.2206	
AW	0.0649	0.1452		
C				
NE	0.0649	0.1452		
NG	0.0496			
NL	0.0558			
NN	0.0528			
NW				



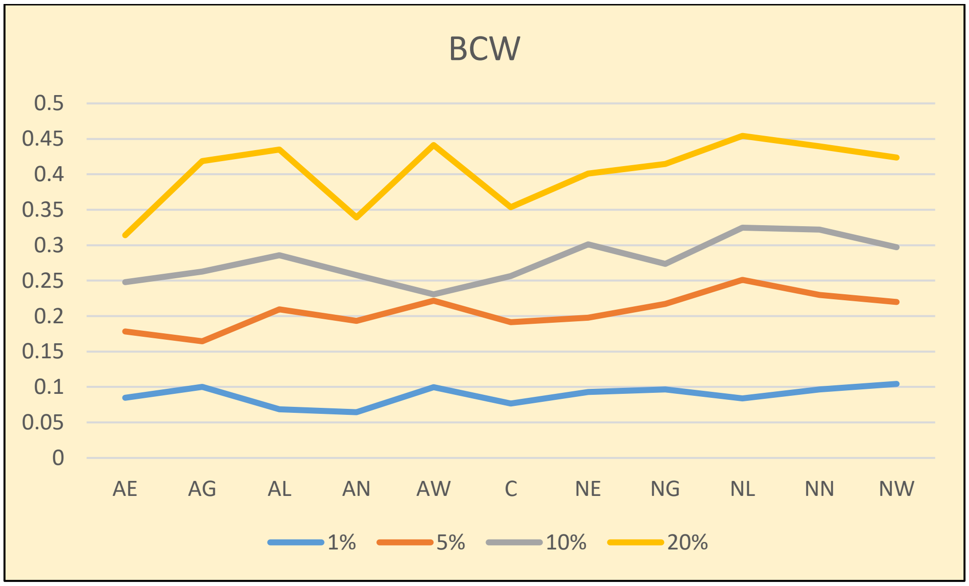
4-Gauss-NRMS: (Avg. NRMS: 0.2284)

4-Gauss-NRMS	1%	5%	10%	20%
AE	0.08	0.2007	0.2335	0.3273
AG	0.0833	0.2133	0.2281	0.3598
AL	0.0856	0.1943	0.2364	0.3148
AN	0.1112	0.2274	0.2587	0.3385
AW	0.0853	0.2404	0.2673	0.3627
C	0.08	0.2007	0.2335	0.3686
NE	0.0819	0.1881	0.2509	0.3851
NG	0.0826	0.1807	0.2841	0.3775
NL	0.0658	0.2059	0.2879	0.4337
NN	0.0985	0.2083	0.2588	0.4041
NW	0.0702	0.1929	0.2873	0.3776



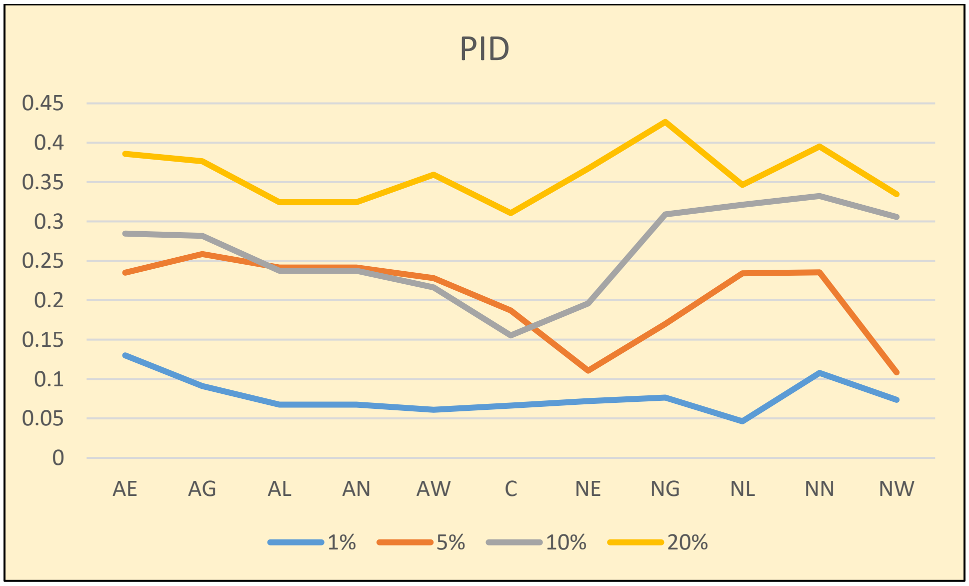
BCW-NRMS: (Avg. NRMS: 0.2393)

BCW-NRMS	1%	5%	10%	20%
AE	0.085	0.1783	0.2481	0.3141
AG	0.1002	0.1645	0.2627	0.4187
AL	0.0687	0.2097	0.286	0.4349
AN	0.0645	0.1933	0.2579	0.3393
AW	0.0997	0.2219	0.2307	0.4412
C	0.0767	0.1916	0.2566	0.3535
NE	0.0928	0.1979	0.3013	0.4012
NG	0.0968	0.2172	0.2738	0.4144
NL	0.0841	0.2511	0.3247	0.4542
NN	0.0964	0.2299	0.3218	0.4394
NW	0.1044	0.2197	0.297	0.4236



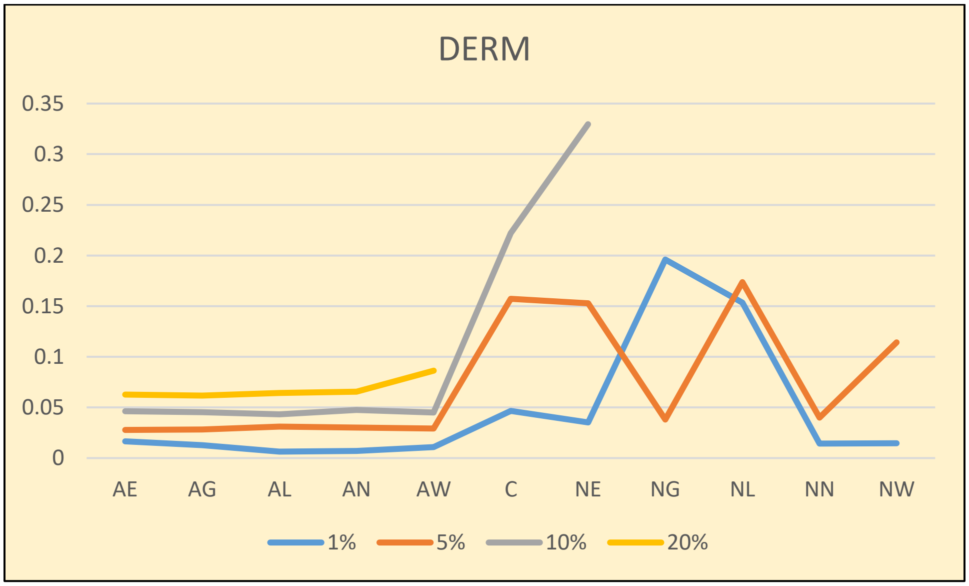
PID-NRMS: (Avg. NRMS: 0.2259)

PID-NRMS	1%	5%	10%	20%
AE	0.1301	0.2352	0.2847	0.3859
AG	0.091	0.2587	0.2818	0.3767
AL	0.0676	0.2415	0.2377	0.3245
AN	0.0676	0.2415	0.2377	0.3245
AW	0.061	0.228	0.2164	0.3596
C	0.0664	0.1871	0.1554	0.3107
NE	0.0721	0.1109	0.1961	0.3671
NG	0.0765	0.1701	0.3089	0.4265
NL	0.0464	0.2342	0.3214	0.3463
NN	0.108	0.2354	0.3324	0.3953
NW	0.0738	0.1085	0.3057	0.3346



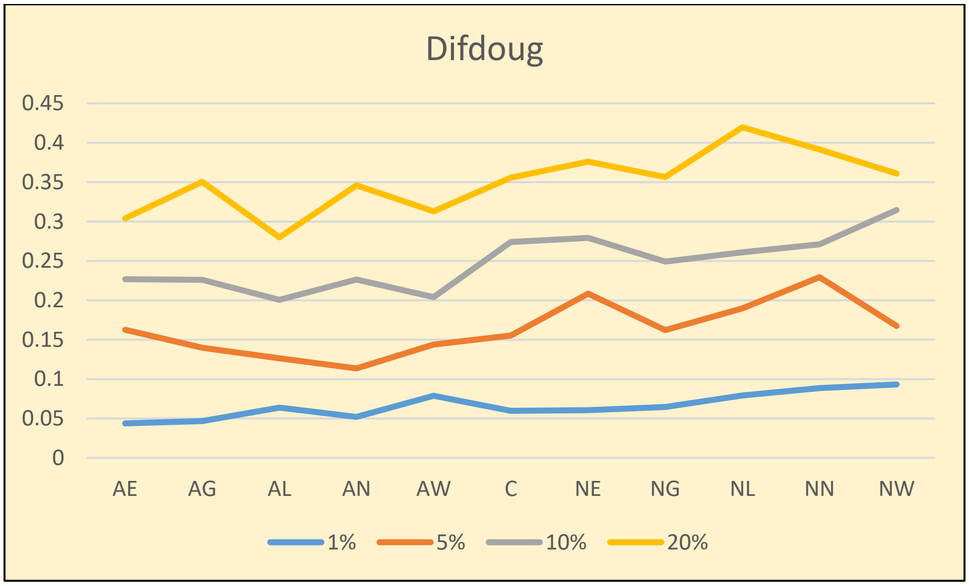
DERM-NRMS: (Avg. NRMS: 0.07816)

DERM-NRMS	1%	5%	10%	20%
AE	0.0166	0.0278	0.0463	0.0628
AG	0.0126	0.0283	0.0454	0.0617
AL	0.0064	0.031	0.0432	0.0644
AN	0.0072	0.03	0.0476	0.0655
AW	0.0107	0.0292	0.045	0.0863
C	0.0466	0.1573	0.222	
NE	0.0351	0.1529	0.3296	
NG	0.196	0.0381		
NL	0.1534	0.1737		
NN	0.0143	0.04	0.2802	
NW	0.0146	0.1141		



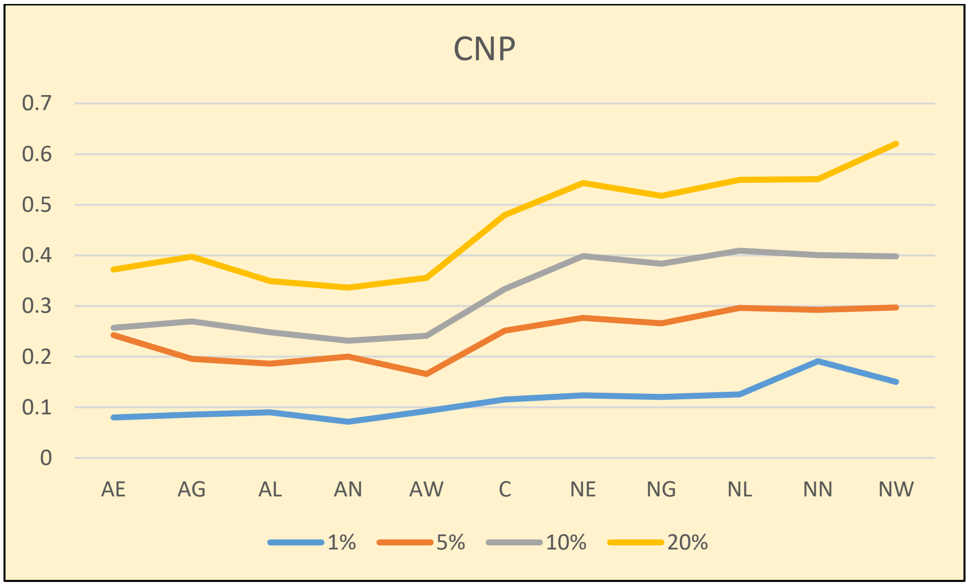
Difdoug-NRMS: (Avg. NRMS: 0.2071)

Difdoug-NRMS	1%	5%	10%	20%
AE	0.0437	0.1624	0.2269	0.3041
AG	0.0467	0.1398	0.2261	0.3502
AL	0.0639	0.1261	0.2005	0.2798
AN	0.052	0.1135	0.2263	0.3458
AW	0.0787	0.1438	0.2041	0.313
C	0.0597	0.1551	0.274	0.3556
NE	0.0603	0.2086	0.2792	0.3761
NG	0.0646	0.1621	0.249	0.3562
NL	0.0791	0.1897	0.2608	0.4195
NN	0.0887	0.2293	0.271	0.3912
NW	0.0931	0.1674	0.3145	0.3609



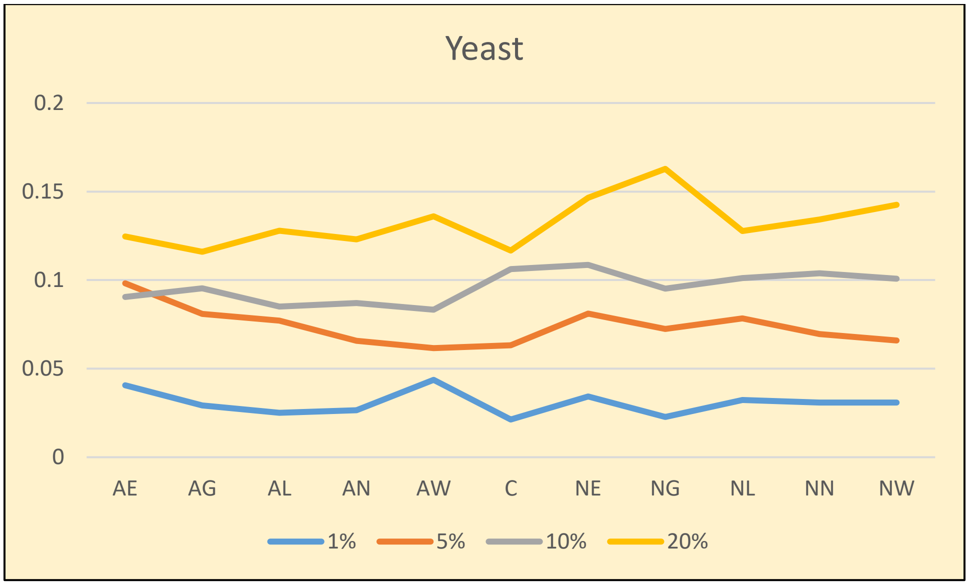
CNP-NRMS: (Avg. NRMS: 0.2853)

CNP-NRMS	1%	5%	10%	20%
AE	0.08	0.2424	0.2569	0.372
AG	0.0855	0.1959	0.2695	0.3972
AL	0.0902	0.186	0.2482	0.3493
AN	0.0715	0.1999	0.2315	0.3363
AW	0.0928	0.1658	0.2411	0.3555
C	0.1153	0.2513	0.3338	0.4793
NE	0.1237	0.2764	0.3984	0.5429
NG	0.1204	0.2658	0.3835	0.5177
NL	0.1256	0.2961	0.4092	0.5492
NN	0.1911	0.2922	0.4003	0.5505
NW	0.1502	0.2971	0.3983	0.6205



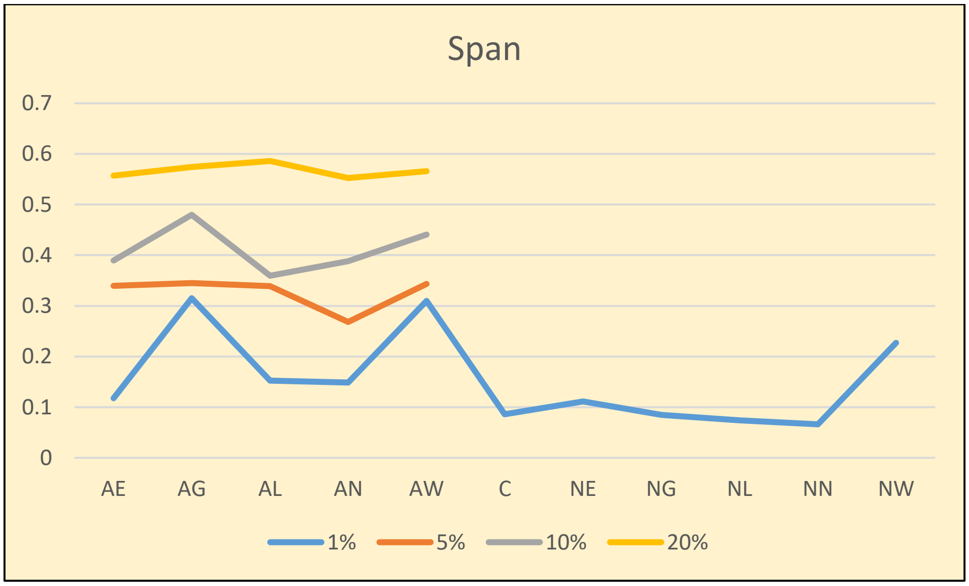
YEAST-NRMS: (Avg. NRMS: 0.0833)

Yeast-NRMS	1%	5%	10%	20%
AE	0.0406	0.0982	0.0905	0.1246
AG	0.0292	0.081	0.0953	0.116
AL	0.025	0.0771	0.085	0.1279
AN	0.0265	0.0658	0.0871	0.123
AW	0.0437	0.0616	0.0833	0.1361
C	0.0213	0.0632	0.1062	0.1166
NE	0.0343	0.0811	0.1086	0.1465
NG	0.0227	0.0724	0.0951	0.1628
NL	0.0324	0.0784	0.1012	0.1278
NN	0.0308	0.0695	0.1039	0.1342
NW	0.0309	0.0659	0.1008	0.1426



SPAN-NRMS: (Avg. NRMS: 0.3162)

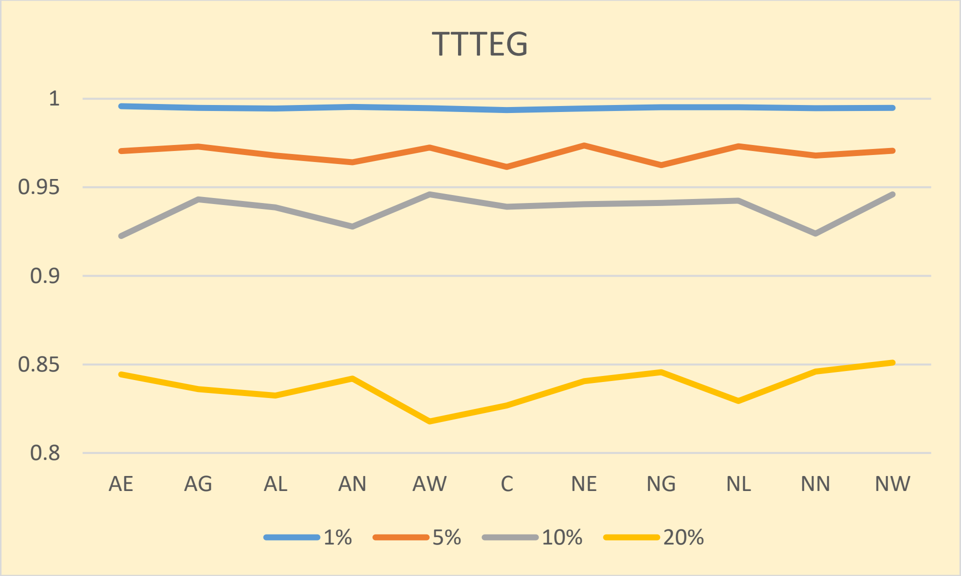
Span-NRMS	1%	5%	10%	20%
AE	0.1173	0.3398	0.3894	0.5574
AG	0.3151	0.345	0.4798	0.5744
AL	0.1524	0.3392	0.3595	0.586
AN	0.1486	0.2682	0.3882	0.5524
AW	0.3098	0.3435	0.441	0.5658
C	0.0858			
NE	0.1112			
NG	0.0849			
NL	0.0741			
NN	0.0661			
NW	0.2272			



AE Calculation for Categorical dataset

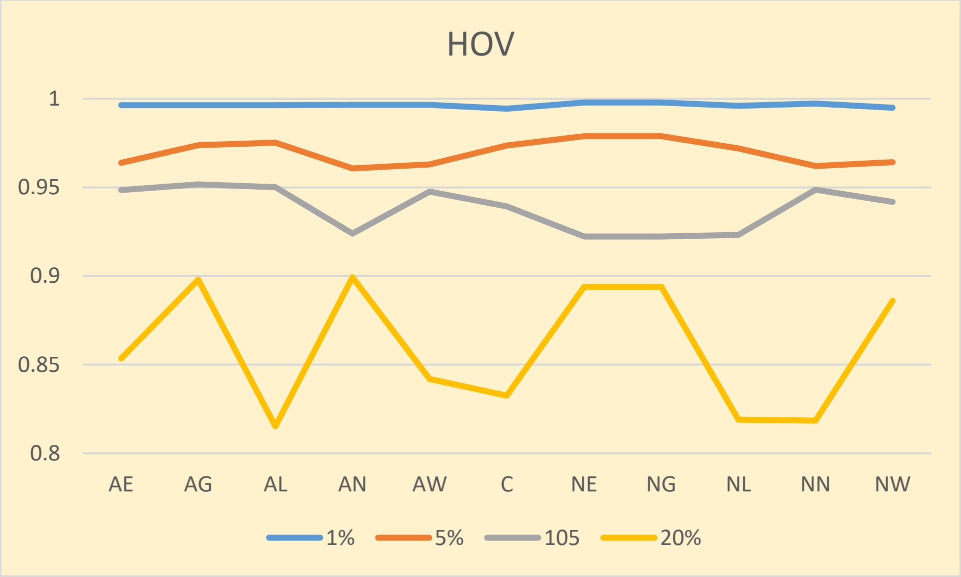
TTTEG-AE: (Avg. AE: 0.9446)

TTTEG-NRMS	1%	5%	10%	20%
AE	0.9958	0.9705	0.9225	0.8444
AG	0.9949	0.973	0.9432	0.8361
AL	0.9945	0.9679	0.9386	0.8325
AN	0.9954	0.9642	0.9279	0.842
AW	0.9947	0.9724	0.946	0.8178
C	0.9936	0.9615	0.939	0.8269
NE	0.9944	0.9736	0.9405	0.8405
NG	0.9952	0.9625	0.9412	0.8456
NL	0.9952	0.9731	0.9424	0.8294
NN	0.9947	0.968	0.9239	0.846
NW	0.9948	0.9707	0.946	0.851



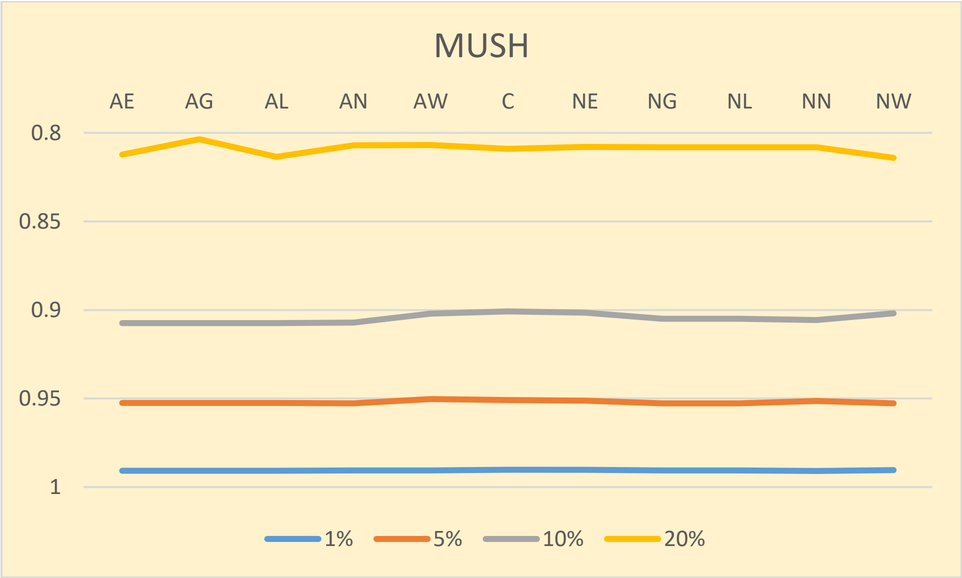
HOV-AE: (Avg. AE: 0.9408)

HOV-NRMS	1%	5%	10%	20%
AE	0.9963	0.9639	0.9485	0.8534
AG	0.9963	0.9738	0.9517	0.898
AL	0.9963	0.9752	0.9501	0.8152
AN	0.9965	0.9607	0.9239	0.8993
AW	0.9965	0.9629	0.9476	0.8419
C	0.9944	0.9736	0.9393	0.8325
NE	0.9979	0.9789	0.9223	0.894
NG	0.9979	0.9789	0.9223	0.894
NL	0.996	0.972	0.9233	0.8189
NN	0.9973	0.962	0.9487	0.8184
NW	0.9949	0.9642	0.9418	0.886



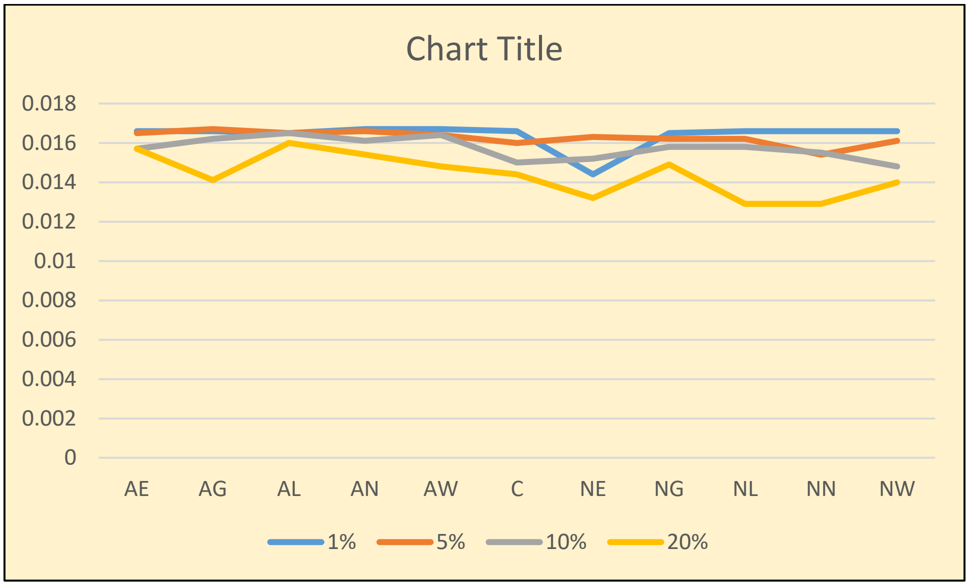
MUSH-AE: (Avg. AE: 0.9140)

MUSH-NRMS	1%	5%	10%	20%
AE	0.9907	0.9524	0.9074	0.8122
AG	0.9907	0.9525	0.9074	0.8036
AL	0.9907	0.9525	0.9074	0.8135
AN	0.9906	0.9527	0.9071	0.8071
AW	0.9906	0.9503	0.902	0.8068
C	0.9902	0.9508	0.9008	0.8091
NE	0.9903	0.9512	0.9016	0.8079
NG	0.9906	0.9527	0.905	0.8081
NL	0.9906	0.9527	0.905	0.8081
NN	0.9909	0.9514	0.9056	0.8082
NW	0.9905	0.9527	0.9018	0.8141



Splice-AE: (Avg. AE: 0.01569)

Splice-NRMS	1%	5%	10%	20%
AE	0.0166	0.0165	0.0157	0.0157
AG	0.0166	0.0167	0.0162	0.0141
AL	0.0165	0.0165	0.0165	0.016
AN	0.0167	0.0166	0.0161	0.0154
AW	0.0167	0.0164	0.0164	0.0148
C	0.0166	0.016	0.015	0.0144
NE	0.0144	0.0163	0.0152	0.0132
NG	0.0165	0.0162	0.0158	0.0149
NL	0.0166	0.0162	0.0158	0.0129
NN	0.0166	0.0154	0.0155	0.0129
NW	0.0166	0.0161	0.0148	0.014



Note: In the splice dataset due to error in the given missing dataset value average accuracy rate AE is very less which is unacceptable value.

- For the **Mixed dataset** we tried to split dataset in two separate Numerical and Categorical values and run them with code, but due to splitting of data particular for neural network algorithm we cannot get the effective values.

Complexity:

Efficiency of any algorithms are related to complexity of that algorithm. Generally it is given by the mathematical O function. Which shows the limited behavior of the algorithm function output with change of parameters.

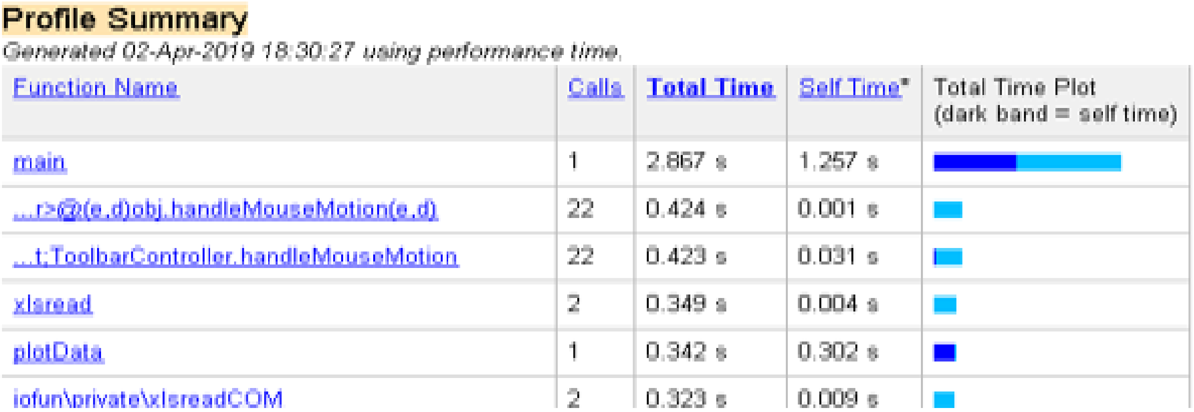
The Algorithm complexity is also a measure of operation counting and its generally related to the approximation of number of steps required to execution of algorithm.

In **‘Self-organizing map’** there are two main complexities.

1) **Time Complexity:**
   This is given by $\mathbf{T = O(Xt)}$, where the “**X**” is the given matrix data and “**t**” is the number of iteration used for the execution. And complexity s in order of **“X^2**”

- **Matrix Dimension:** The given matrix is **X** which have **(m x n)** dimensions in which the “**m**” suggest Number of row and “**n**” suggest the Number of column. When we change the dimension of the mapping function we observe that the time increase.

- **Example:**
  If we run the dataset **Sheart_AG_1** with mapping size (20 X 20) for 5 iteration, 10 learning rate, and 15 initial width then time require for the code is **2.867sec**:



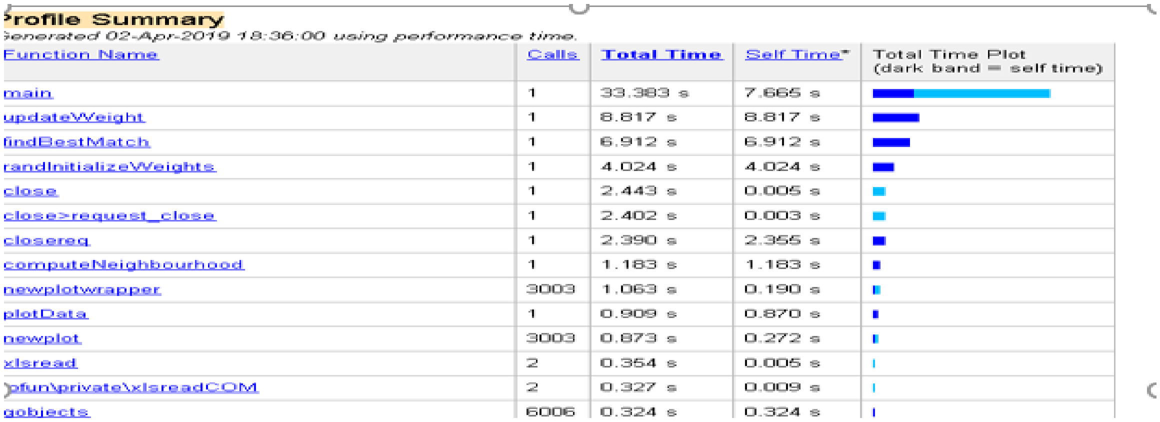
Profile Summary

Generated 02-Apr-2019 18:30:27 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
main	1	2.867 s	1.257 s	
...r>@(e,d)obj.handleMouseMotion(e,d)	22	0.424 s	0.001 s	
...t;ToolbarController.handleMouseMotion	22	0.423 s	0.031 s	
xlsread	2	0.349 s	0.004 s	
plotData	1	0.342 s	0.302 s	
iofun\private\xlsreadCOM	2	0.323 s	0.009 s	

Fig: Program execution time for Sheart_AG_1 with (20 x 20)

- And **Sheart_AG_1 (270 X 13)** run by mapping size (1500 X 1500) for 1 iteration, 10 learning rate, and 15 initial width then time require for the code is **33.383sec**:



rofile Summary

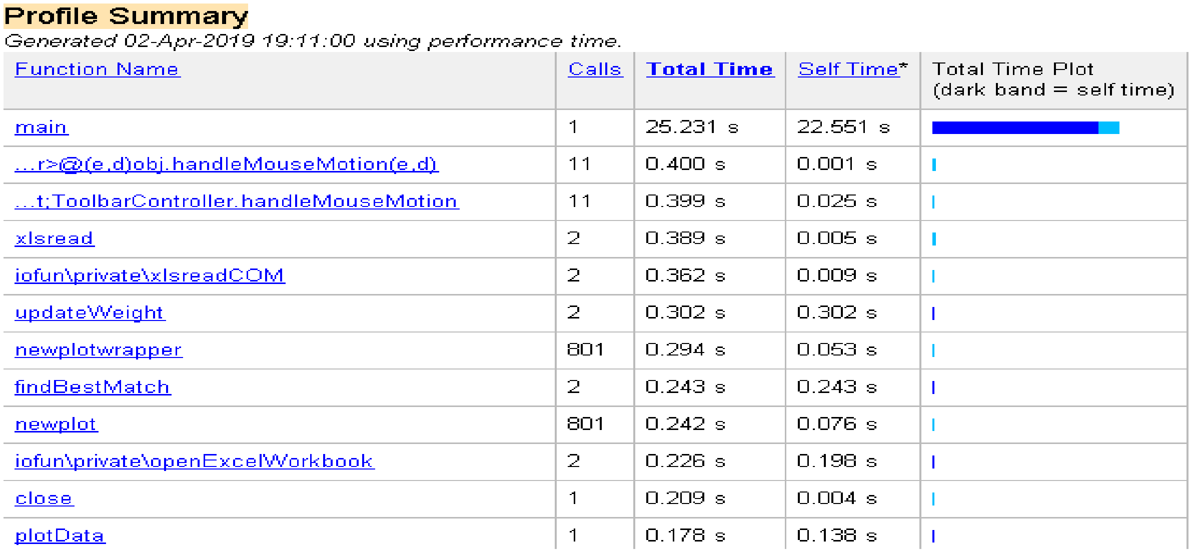
enerated 02-Apr-2019 18:36:00 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
main	1	33.383 s	7.665 s	
updateWeight	1	8.817 s	8.817 s	
findBestMatch	1	6.912 s	6.912 s	
randInitializeWeights	1	4.024 s	4.024 s	
close	1	2.443 s	0.005 s	
close>request_close	1	2.402 s	0.003 s	
closereq	1	2.390 s	2.355 s	
computeNeighbourhood	1	1.183 s	1.183 s	
newplotwrapper	3003	1.063 s	0.190 s	
plotData	1	0.909 s	0.870 s	
newplot	3003	0.873 s	0.272 s	
xlsread	2	0.354 s	0.005 s	
pfun\private\xlsreadCOM	2	0.327 s	0.009 s	
gobjects	6006	0.324 s	0.324 s	

Fig: Program execution time for Sheart_AG_1 with (1500 x 1500)

So, from observation we can analyze that with more mapping size dimension it take more time to run for same dataset. For other program with more missing data percentage value it take time around 1 hour, 30 minute to 2 hours.

- When we run big data set **4-gauss_AG_1 (800 X 12)** which have more data size then the Sheart then for same no. of missing data we found more time of execution for the code is **25.231sec**.



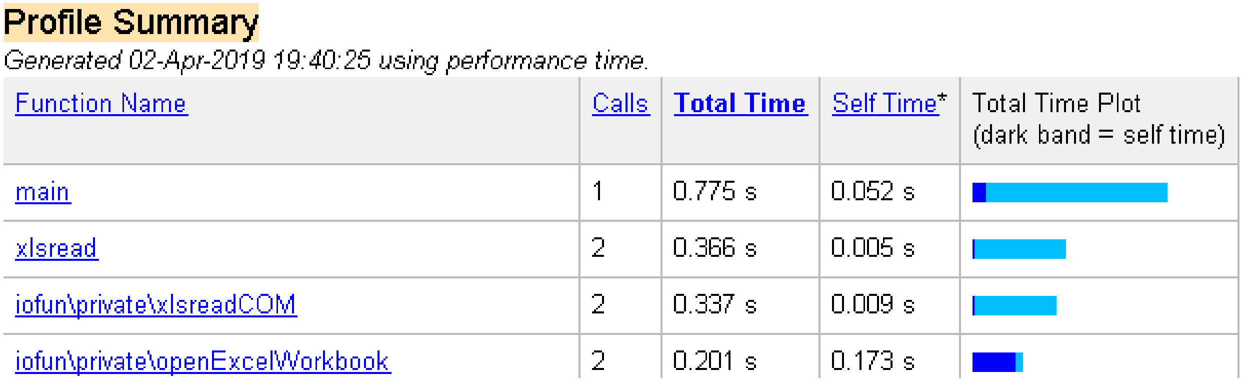
Profile Summary

Generated 02-Apr-2019 19:11:00 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
main	1	25.231 s	22.551 s	
...r>@(e,d)obj.handleMouseMotion(e,d)	11	0.400 s	0.001 s	
...t;ToolbarController.handleMouseMotion	11	0.399 s	0.025 s	
xlsread	2	0.389 s	0.005 s	
iofun\private\xlsreadCOM	2	0.362 s	0.009 s	
updateWeight	2	0.302 s	0.302 s	
newplotwrapper	801	0.294 s	0.053 s	
findBestMatch	2	0.243 s	0.243 s	
newplot	801	0.242 s	0.076 s	
iofun\private\openExcelWorkbook	2	0.226 s	0.198 s	
close	1	0.209 s	0.004 s	
plotData	1	0.178 s	0.138 s	

Fig: Program execution time for 4-gauss_AG_1

- **<u>No. of Iterations:</u>** Now let’s find the time complexity with the no of iteration change.so here for the same data **Sheart_AG_1** with **1** iteration the time taken for executing the main code is **0.775sec**, which is comparatively low :



Profile Summary

Generated 02-Apr-2019 19:40:25 using performance time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
main	1	0.775 s	0.052 s	
xlsread	2	0.366 s	0.005 s	
iofun\private\xlsreadCOM	2	0.337 s	0.009 s	
iofun\private\openExcelWorkbook	2	0.201 s	0.173 s	

This way we can conclude that the change of iteraton numbers and the dimension of matrix will affect the time of the code execution procedure.

2) <u>Space complexity:</u>

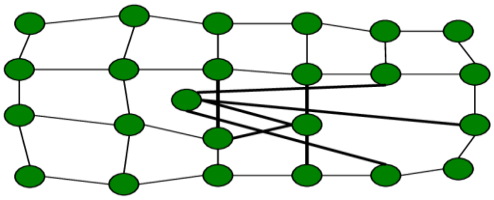
When calculate the big data set for more iterations with high dimension of samples then the value of the data require more space in computer RAM and ROM. For some dataset like KDD, latter, Adult, Credit we cannot get the answer due to limitation of device space. For running these codes we use 16GBRAM, I-7 Process and 8GB RAM, I-5 Processor with SSD hard drive.

Assumption for the Complexity:

1. If **m>>n** then the order of the complexity is T= (**mt**)
2. If **n>> m** then the order of the complexity is T = (**nt**)
3.

Improvement

1. The quality of Self organizing map is improve by "Avoidance of Self Intersection" of each data [7]



2. Improving the self-organizing map with the "Efficient Initialization Scheme."… Which save the time and the accuracy of the data set with less amount of time (iteration). [8]

3. Improving the accuracy of self-organizing map by using the "Distance Metrix Learning" method

Conclusion:

- To conclude, the project for the missing dataset imputation, self- Organizing map can give the value of NRMS and AE. The mapping size, learning rate, and initial width are impacting parameters. We get good value of NRMS for 1 and 5 percent missing dataset and for 10 and 20 percent missing dataset the error rate of the output are comparatively more. So, based on that analysis we clearly represent that the Algorithm take less time for low and medium missing data but for higher level of missing data it takes more time.

- Accuracy of the imputed values are also affected by the percentage of missing data set.

References:

[1] J. H. a. M. Kamber, Data Mining , Concepts and Techniques, Diane Cerra, 2006.

[2] B. A. L. a. G. Mercier, "Self-Organising maps for processing of data with missing values and outlines," p. 210, 2010.

[3] Starkey A. U. (2017, April 22). Application of feature selection methods for automated clustering analysis: a review on synthetic datasets. Retrieved from Neural Computing and Applications: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5857284/

[4] Midenet, F. F. (2002). Self-organising map for data imputation and correction in surveys," Neural Computing & Applications. vol. 10, pp. 300–310

[5] "Self organising map tutorial," 02 November 2017. [Online]. Available: https://algobeans.com/2017/11/02/self-organising-map/. [Accessed 05 February 2019].

[6] K. Pang, "Self-organizing Maps", *Cs.hmc.edu*, 2003. [Online]. Available: http://www.cs.hmc.edu/~kpang/nn/som.html. [Accessed: 19- Mar- 2019].

[7]E. López-Rubio, "ResearchGate," August 2013. [Online]. Available: https://www.researchgate.net/figure/Example-of-self-intersection_fig4_262149778. [Accessed 21 February 2019].

[8]I. D. a. S. P. Mike Huang, "Self organizing features map," [Online]. Available: http://pages.cpsc.ucalgary.ca/~jacob/Courses/Winter2000/CPSC533/Slides/05.2.3-SOFM.pdf. [Accessed February 2019].