Capstone report

Guide-Dr Vibhor Kumar

Topic-Network inference in sparse datasets using machine learning and python

Libraries used-pandas , NumPy, random forest classifier, random forest regressor ,PCA ,train_test_split

What is Network Inference

Network Inference is a research project theme in which the network end-system (i.e., the computer) infers properties about the behaviour of the network and other end-systems in order to get a better experience. Such improvements might be better sharing or improved latency through reduced queueing .

Network inference's applications can be in various fields such as biology and computer science, and It can also handle large range of the data sources like the gene data and the genomic data available publicly .

How to infer networks from large datasets

1. Data pre-processing:. Publicly available datasets can have noise and redundant information which can be a problem in our network inference so we need to remove these , for this we use a variety of techniques for normalization like standard scaler , qq norm etc .
2. Selecting an appropriate inference method: There are a variety of methods available for network inference according to our requirements we can choose the suitable one for data having noise and dimensions are high we can use ml methods like random forest classifer,regressor or we can also use deep learning . In our project we have made use of random forest regressor .
3. Parameter selection: Many inference methods have parameters that need to be set. The optimal values of these parameters depend on the data and the inference method used. Thus, it is important to carefully select the parameter values and assess their impact on the results.
4. Network validation: Once a network is inferred, it is important to validate it using additional data or experimental validation. This helps to ensure that the network accurately reflects the underlying biological or social processes.

Genie3

GENIE3 (Gene Network Inference with Ensemble of trees) it is an algorithm for inferring networks which makes use of the decision tree ensembles to corelate the corelation and information between the various genes available in the network . On advantage of this is that it has the power to infer both directed as well as undirected networks .

The basic idea behind GENIE3 is to use random forests, which are an ensemble of decision trees, to predict the expression of a target gene based on the expression of other genes in the dataset. The importance of each predictor gene in the random forest is then used as a measure of the strength of the regulatory interaction between the predictor and the target gene. This process is repeated for each gene in the dataset, resulting in a complete regulatory network.

One main disadvantage of genie 3 is an assumption that which assumes that the interactions between the entries are in a linear manner . It also does not consider the dependencies of temporal type in the network .

It is the most widely used method for network inference .

Random Forest for network inference

Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset.

Random forest is a type of classifier which makes the use of multiple decision trees and takes the decision of the majority of the trees decision and tries to make a prediction on the dataset . It is of two types classifier and regressor .

It can be used to infer the network by calculation of the feature importance's of the target variables and then try to form a relationship between them to infer the network . The basic idea is to use random forest to predict the expression level of a target gene based on the expression levels of other genes in the dataset. The importance scores of the predictor genes are then used as a measure of the strength of the regulatory interaction between the predictor and the target gene.

To apply random forest for network inference, the following steps can be taken:

1. Data pre-processing: Here the data is is normalized and cleaned using the methods discussed above .
2. Training the random forest: We then train our random forest model using the target as our y and our data minus y as x and then we predict using x and y .
3. Determining variable importance: The variable importance measures are calculated to determine the strength of the regulatory interactions between the predictor and target genes. The importance measures can be calculated using different methods, such as mean decrease impurity or mean decrease accuracy.
4. Network construction: Once the variable importance measures are obtained, a regulatory network can be constructed by connecting the predictor genes to the target gene using edges that correspond to the strength of the regulatory interaction.

5. Network validation: The network should be validated using external datasets or experimental validation methods to ensure that it accurately reflects the underlying biological or social processes.

We can also use random forest along with other ml methods like neural networks or gradient boost to improve performance .

PCA

PCA stands for Principal Component Analysis it is a technique used for reducing the dimensions of the data , we can customize the number of dimensions we need to transform our data into . PCA aims to reduce the dimensionality of the data without much loss in the information in the dataset .

Approach

We have calculated the feature importances using the random forest regressor ,then we have applied dimension reduction i.e. PCA on the dataset and analysed the feature importances of the dataset , we then have checked at different sparseness of the data i.e. 10,20,30 and analyses the result and tried to find a correlation between the feature importance calculated and try to infer network from it .

We have tried our algorithm on 30 datasets to get a good idea of the problem .

We have tried our algorithm on multiple matrix factorization techniques like

SVD- SVD stands for Singular Value Decomposition. It is a matrix factorization technique commonly used in linear algebra and numerical analysis. SVD decomposes a matrix into three separate matrices, allowing us to represent the original matrix in a more compact form.

Given an m x n matrix A, the SVD factorizes it into three matrices:

A = U * Σ * V^T

where:

- U is an m x m orthogonal matrix. The columns of U are called the left singular vectors of A.
- Σ is an m x n diagonal matrix with non-negative real numbers on the diagonal. These values are known as the singular values of A.
- V^T is the transpose of an n x n orthogonal matrix V. The columns of V are called the right singular vectors of A.

The singular values in Σ are arranged in descending order, meaning the first singular value is the largest, and the last singular value is the smallest. The singular vectors in U and V are ordered accordingly.

NMF- NMF stands for Non-negative Matrix Factorization. It is a matrix factorization technique used to decompose a non-negative matrix into two non-negative matrices. Unlike SVD, which allows negative values in the factorization, NMF imposes the constraint that all elements of the matrices are non-negative.

Given an m x n non-negative matrix V, NMF factorizes it into two non-negative matrices:

$V \approx W * H$

where:

- V is the original matrix that we want to factorize.
- W is an m x r non-negative matrix, where r is a user-defined parameter representing the desired number of components or features.
- H is an r x n non-negative matrix.

The goal of NMF is to find W and H such that their product approximates the original matrix V as closely as possible. This approximation is typically achieved by minimizing a cost function, such as the Euclidean distance or the Kullback-Leibler divergence, between V and the reconstructed matrix obtained from the product of W and H.

Code snippet

```
def corell(pcimp, i1, x, x_pca, met):

    # c
    r = x.shape[1]
    print(r)
    # corr_matrix = x.shape1,i1
    #print("hi")

    corr_matrix1 = pd.DataFrame(pca.components_, columns=x.columns)
    corr_matrix1 = corr_matrix1.transpose()
    corr_matrix1 = corr_matrix1.to_numpy()

    corr_matrix = np.zeros((r, i1))

    for i in range(x.shape[1]):
        for j in range(i1):
            resco = stats.spearmanr(x.iloc[:,i], x_pca[:, j])
            corr_matrix[i][j] = resco.correlation

    corr_matrix = abs(corr_matrix)
    #print(corr_matrix)
    #print(corr_matrix1)


    #res = np.zeros((r, 1));
    res = [0] * x.shape[1]
    print(res)

    if (met == 1):

        for ii in range(x.shape[1]):
            ##kj
            feat1 = 0
            for j in range(i1):
                feat1 = feat1 + pcimp[j] * corr_matrix[ii][j]
            temp = 0
            for k in range(i1):
                # print(k)
                temp = temp + corr_matrix[ii][k]
            #res[ii] = feat1 / temp
            res[ii] = abs(feat1)
    if (met == 2):
        for ii in range(x.shape[1]):
            ##kj
            feat1 = 0
            for j in range(i1):
                feat1 = feat1 + pcimp[j] * corr_matrix1[ii][j]
            temp = 0
            for k in range(i1):
                # print(k)
                temp = temp + corr_matrix1[ii][k]
            #res[ii] = feat1 / temp
            res[ii] = abs(feat1)
```

Results

Sparseness level = 60%

Dataset	**PCA**	**NMF**	**SVD**	**PCA**	**NMF**	**SVD**
Divorce	0.20688	0.2218	0.10477	**0.2289**	**0.37893**	**0.04825**
Musk	**0.0189**	**na**	0.026	**0.0283**	**na**	**0.07932**
Amphibian	**0.2320**	**0.141**	**-0.187813**	**0.3411**	**0.3236**	**0.033**
Insurance	**0.3177**	**0.2931**	**0.1866**	**0.14025**	**0.14117**	**0.0064**

Sparseness level=40%

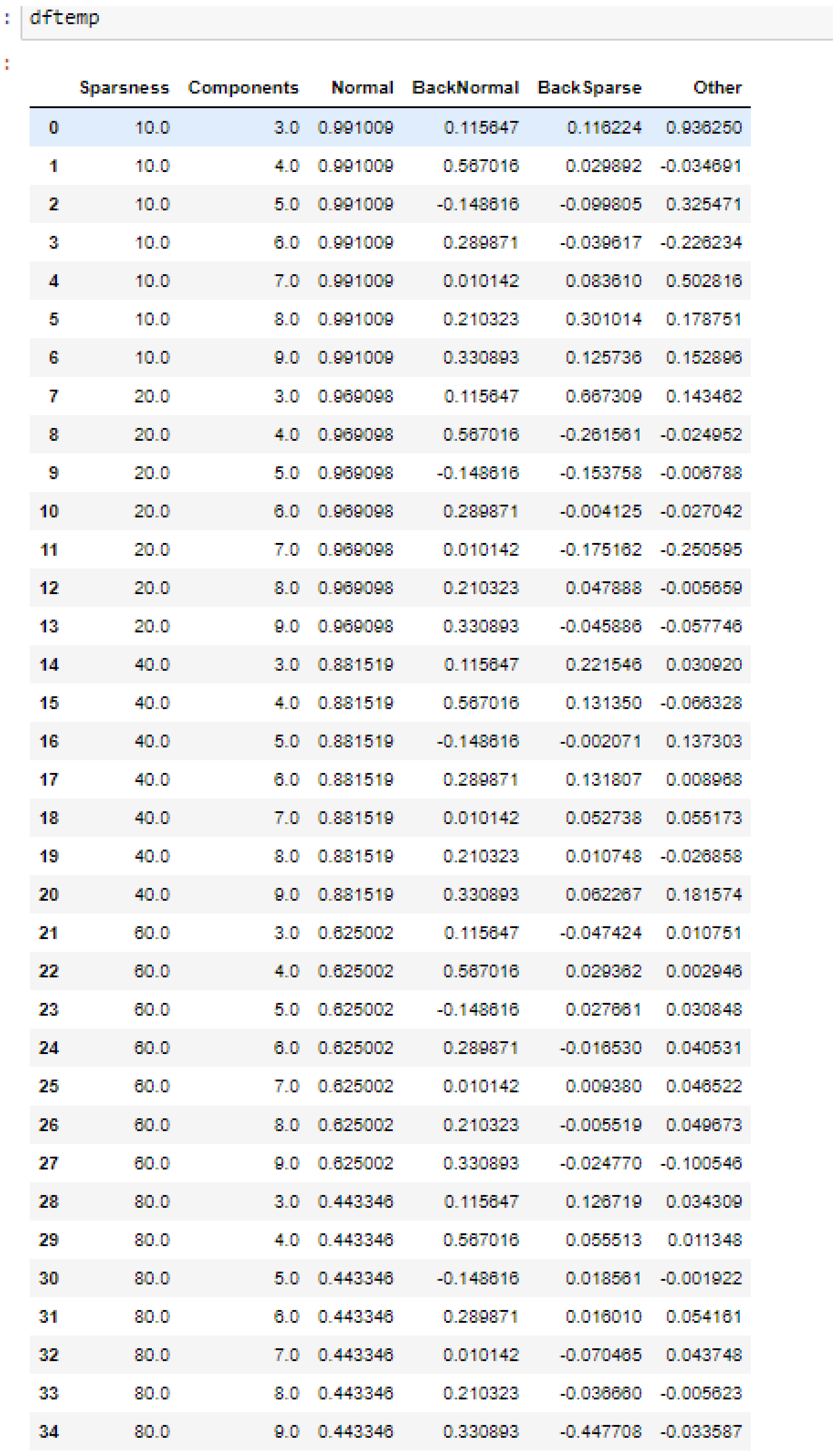
Dataset	**PCA**	**NMF**	**SVD**	**PCA**	**NMF**	**SVD**
Divorce	**0.20688**	**0.2452**	**0.210040**	**0.2305**	**0.2241**	**0.1275**
Musk	**0.04690**	**na**	**0.03620**	**0.0076**	**NA**	**0.00345**
Amphibian	**0.1133**	**0.2910**	**0.170**	**0.4591**	**0.0109**	**0.2625**
Insurance	**0.3177**	**0.2832**	**0.2865**	**-0.0165**	**-0.01793**	**0.1445**

In our research we have found out even after back projection on a few datasets, the feature importances have not degraded severely.

If we are doing without back projection and if the sparseness level is high the n our method is not performing as expected.

We have found out without dividing gives better results on most datasets.

On spambase dataset from UCI repository (with divide)

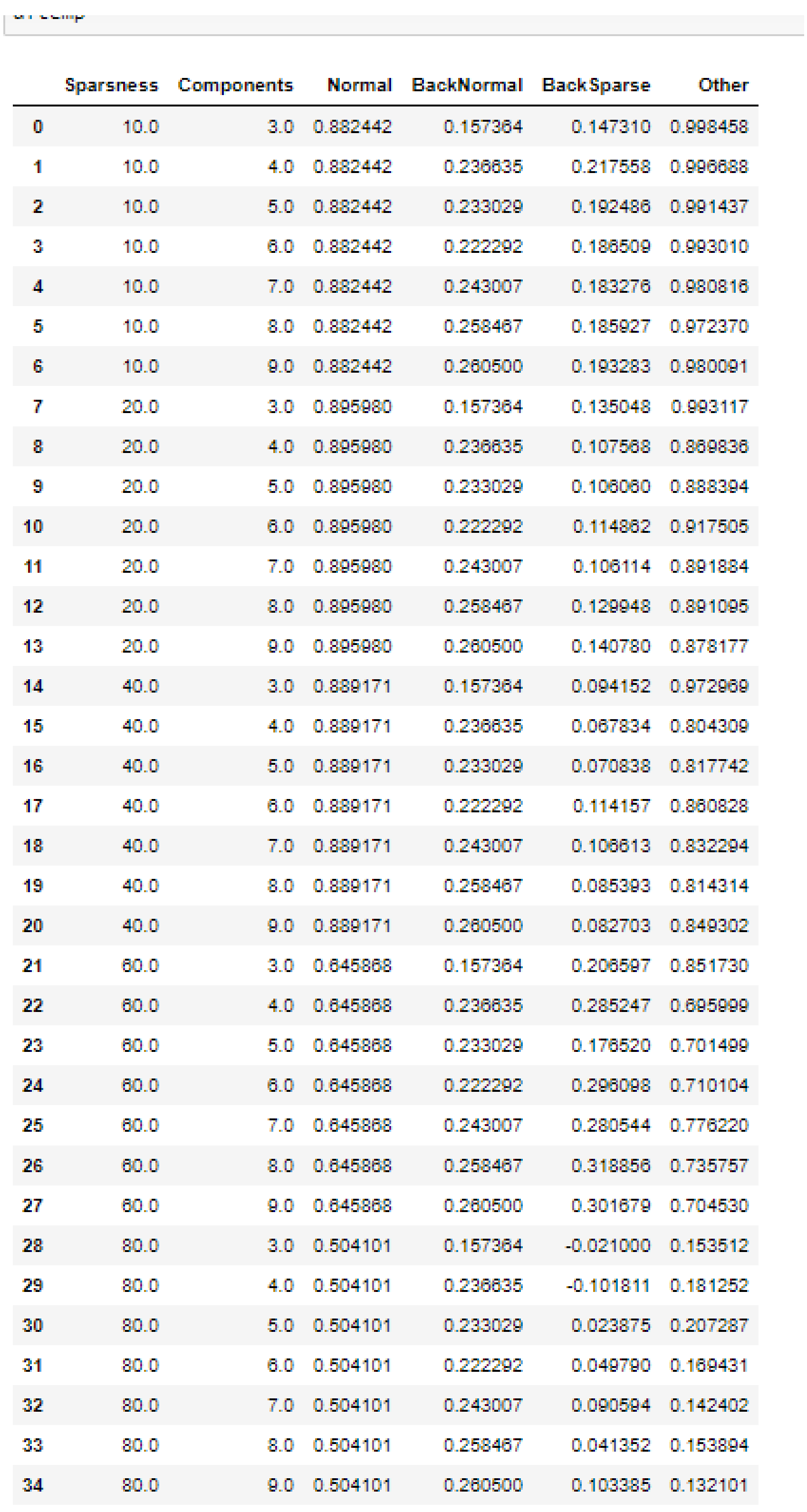


```
: dftemp
```

:

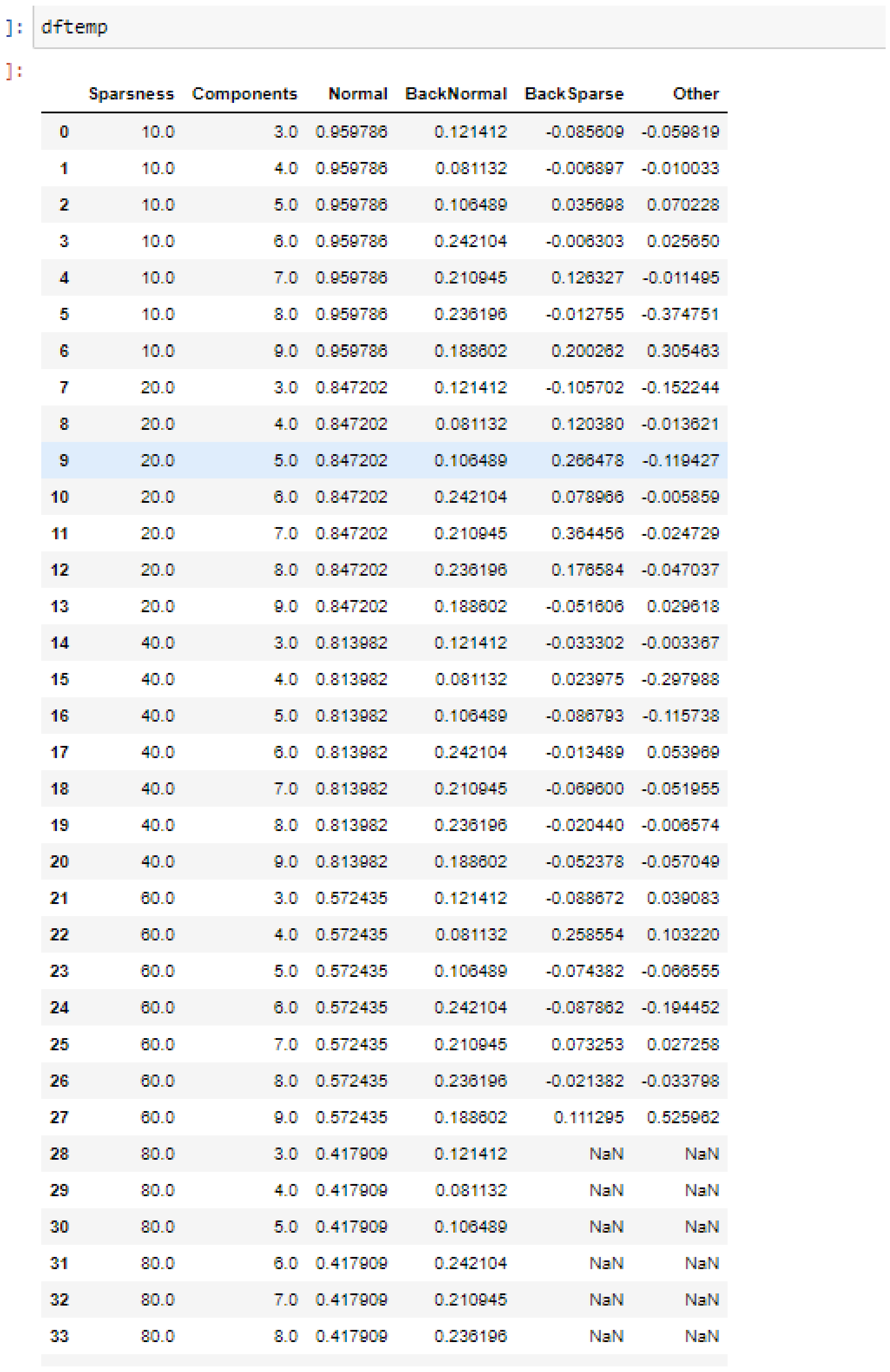
	Sparsness	Components	Normal	BackNormal	BackSparse	Other
0	10.0	3.0	0.991009	0.115647	0.116224	0.936250
1	10.0	4.0	0.991009	0.567016	0.029892	-0.034691
2	10.0	5.0	0.991009	-0.148616	-0.099805	0.325471
3	10.0	6.0	0.991009	0.289871	-0.039617	-0.226234
4	10.0	7.0	0.991009	0.010142	0.083610	0.502816
5	10.0	8.0	0.991009	0.210323	0.301014	0.178751
6	10.0	9.0	0.991009	0.330893	0.125736	0.152896
7	20.0	3.0	0.969098	0.115647	0.667309	0.143462
8	20.0	4.0	0.969098	0.567016	-0.261561	-0.024952
9	20.0	5.0	0.969098	-0.148616	-0.153758	-0.006788
10	20.0	6.0	0.969098	0.289871	-0.004125	-0.027042
11	20.0	7.0	0.969098	0.010142	-0.175162	-0.250595
12	20.0	8.0	0.969098	0.210323	0.047888	-0.005659
13	20.0	9.0	0.969098	0.330893	-0.045886	-0.057746
14	40.0	3.0	0.881519	0.115647	0.221546	0.030920
15	40.0	4.0	0.881519	0.567016	0.131350	-0.066328
16	40.0	5.0	0.881519	-0.148616	-0.002071	0.137303
17	40.0	6.0	0.881519	0.289871	0.131807	0.008968
18	40.0	7.0	0.881519	0.010142	0.052738	0.055173
19	40.0	8.0	0.881519	0.210323	0.010748	-0.026858
20	40.0	9.0	0.881519	0.330893	0.062267	0.181574
21	60.0	3.0	0.625002	0.115647	-0.047424	0.010751
22	60.0	4.0	0.625002	0.567016	0.029362	0.002946
23	60.0	5.0	0.625002	-0.148616	0.027661	0.030848
24	60.0	6.0	0.625002	0.289871	-0.016530	0.040531
25	60.0	7.0	0.625002	0.010142	0.009380	0.046522
26	60.0	8.0	0.625002	0.210323	-0.005519	0.049673
27	60.0	9.0	0.625002	0.330893	-0.024770	-0.100546
28	80.0	3.0	0.443346	0.115647	0.126719	0.034309
29	80.0	4.0	0.443346	0.567016	0.055513	0.011348
30	80.0	5.0	0.443346	-0.148616	0.018561	-0.001922
31	80.0	6.0	0.443346	0.289871	0.016010	0.054161
32	80.0	7.0	0.443346	0.010142	-0.070465	0.043748
33	80.0	8.0	0.443346	0.210323	-0.036660	-0.005623
34	80.0	9.0	0.443346	0.330893	-0.447708	-0.033587

On spambase dataset from UCI repository (without divide)



	Sparsness	Components	Normal	BackNormal	BackSparse	Other
0	10.0	3.0	0.882442	0.157364	0.147310	0.998458
1	10.0	4.0	0.882442	0.236635	0.217558	0.996688
2	10.0	5.0	0.882442	0.233029	0.192486	0.991437
3	10.0	6.0	0.882442	0.222292	0.186509	0.993010
4	10.0	7.0	0.882442	0.243007	0.183276	0.980816
5	10.0	8.0	0.882442	0.258467	0.185927	0.972370
6	10.0	9.0	0.882442	0.260500	0.193283	0.980091
7	20.0	3.0	0.895980	0.157364	0.135048	0.993117
8	20.0	4.0	0.895980	0.236635	0.107568	0.869836
9	20.0	5.0	0.895980	0.233029	0.106060	0.888394
10	20.0	6.0	0.895980	0.222292	0.114862	0.917505
11	20.0	7.0	0.895980	0.243007	0.106114	0.891884
12	20.0	8.0	0.895980	0.258467	0.129948	0.891095
13	20.0	9.0	0.895980	0.260500	0.140780	0.878177
14	40.0	3.0	0.889171	0.157364	0.094152	0.972969
15	40.0	4.0	0.889171	0.236635	0.067834	0.804309
16	40.0	5.0	0.889171	0.233029	0.070838	0.817742
17	40.0	6.0	0.889171	0.222292	0.114157	0.860828
18	40.0	7.0	0.889171	0.243007	0.106613	0.832294
19	40.0	8.0	0.889171	0.258467	0.085393	0.814314
20	40.0	9.0	0.889171	0.260500	0.082703	0.849302
21	60.0	3.0	0.645868	0.157364	0.206597	0.851730
22	60.0	4.0	0.645868	0.236635	0.285247	0.695999
23	60.0	5.0	0.645868	0.233029	0.176520	0.701499
24	60.0	6.0	0.645868	0.222292	0.296098	0.710104
25	60.0	7.0	0.645868	0.243007	0.280544	0.776220
26	60.0	8.0	0.645868	0.258467	0.318856	0.735757
27	60.0	9.0	0.645868	0.260500	0.301679	0.704530
28	80.0	3.0	0.504101	0.157364	-0.021000	0.153512
29	80.0	4.0	0.504101	0.236635	-0.101811	0.181252
30	80.0	5.0	0.504101	0.233029	0.023875	0.207287
31	80.0	6.0	0.504101	0.222292	0.049790	0.169431
32	80.0	7.0	0.504101	0.243007	0.090594	0.142402
33	80.0	8.0	0.504101	0.258467	0.041352	0.153894
34	80.0	9.0	0.504101	0.260500	0.103385	0.132101

On QSAR biodegradation Data Set(with divide) from UCI repository

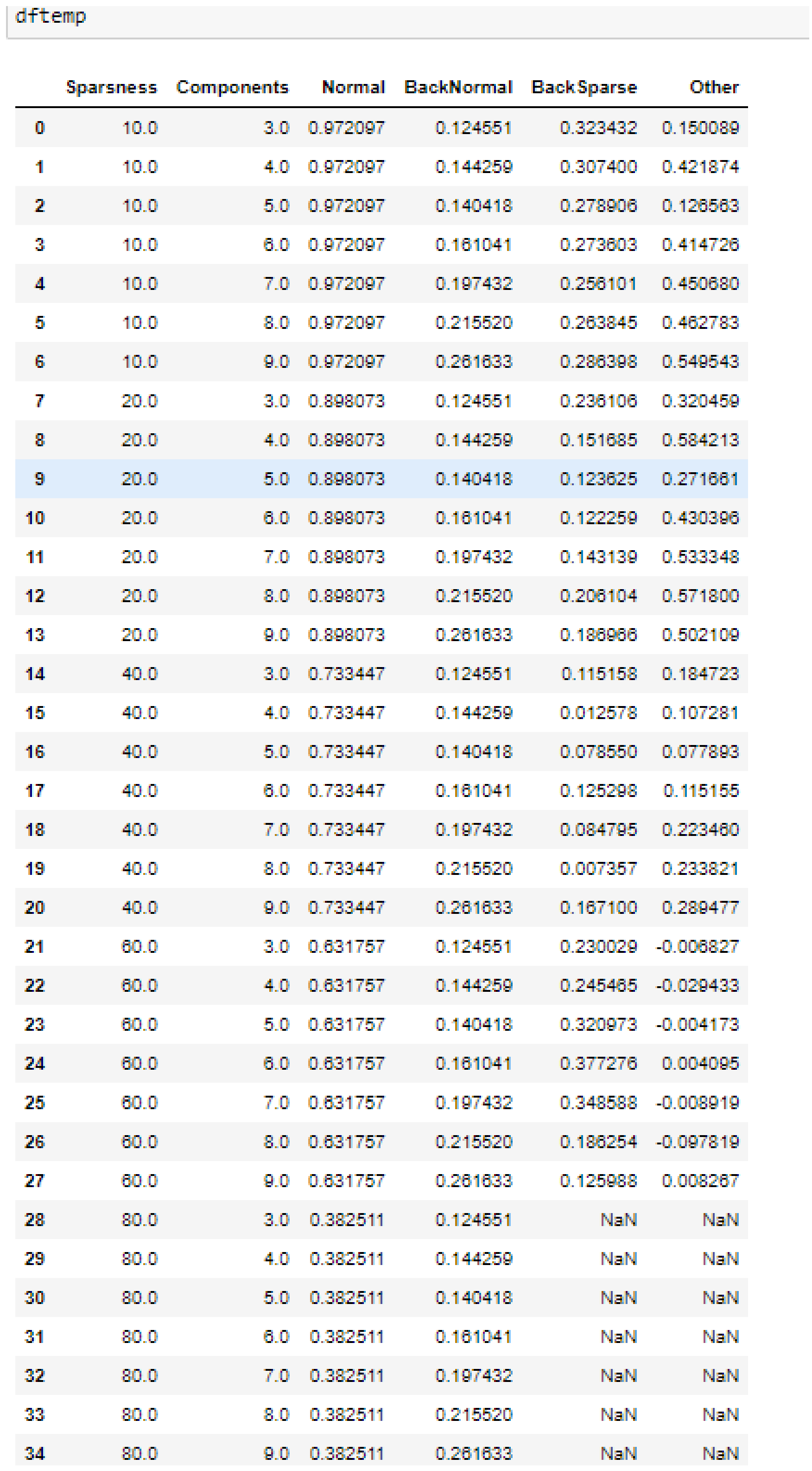


```
]: dftemp
```

```
]:
```

	Sparsness	Components	Normal	BackNormal	BackSparse	Other
0	10.0	3.0	0.959786	0.121412	-0.085609	-0.059819
1	10.0	4.0	0.959786	0.081132	-0.006897	-0.010033
2	10.0	5.0	0.959786	0.106489	0.035698	0.070228
3	10.0	6.0	0.959786	0.242104	-0.006303	0.025650
4	10.0	7.0	0.959786	0.210945	0.126327	-0.011495
5	10.0	8.0	0.959786	0.236196	-0.012755	-0.374751
6	10.0	9.0	0.959786	0.188602	0.200262	0.305463
7	20.0	3.0	0.847202	0.121412	-0.105702	-0.152244
8	20.0	4.0	0.847202	0.081132	0.120380	-0.013621
9	20.0	5.0	0.847202	0.106489	0.266478	-0.119427
10	20.0	6.0	0.847202	0.242104	0.078966	-0.005859
11	20.0	7.0	0.847202	0.210945	0.364456	-0.024729
12	20.0	8.0	0.847202	0.236196	0.176584	-0.047037
13	20.0	9.0	0.847202	0.188602	-0.051606	0.029618
14	40.0	3.0	0.813982	0.121412	-0.033302	-0.003367
15	40.0	4.0	0.813982	0.081132	0.023975	-0.297988
16	40.0	5.0	0.813982	0.106489	-0.086793	-0.115738
17	40.0	6.0	0.813982	0.242104	-0.013489	0.053969
18	40.0	7.0	0.813982	0.210945	-0.069600	-0.051955
19	40.0	8.0	0.813982	0.236196	-0.020440	-0.006574
20	40.0	9.0	0.813982	0.188602	-0.052378	-0.057049
21	60.0	3.0	0.572435	0.121412	-0.088672	0.039083
22	60.0	4.0	0.572435	0.081132	0.258554	0.103220
23	60.0	5.0	0.572435	0.106489	-0.074382	-0.066555
24	60.0	6.0	0.572435	0.242104	-0.087862	-0.194452
25	60.0	7.0	0.572435	0.210945	0.073253	0.027258
26	60.0	8.0	0.572435	0.236196	-0.021382	-0.033798
27	60.0	9.0	0.572435	0.188602	0.111295	0.525962
28	80.0	3.0	0.417909	0.121412	NaN	NaN
29	80.0	4.0	0.417909	0.081132	NaN	NaN
30	80.0	5.0	0.417909	0.106489	NaN	NaN
31	80.0	6.0	0.417909	0.242104	NaN	NaN
32	80.0	7.0	0.417909	0.210945	NaN	NaN
33	80.0	8.0	0.417909	0.236196	NaN	NaN

On QSAR biodegradation Data Set(without divide) from UCI repository



```
dftemp
```

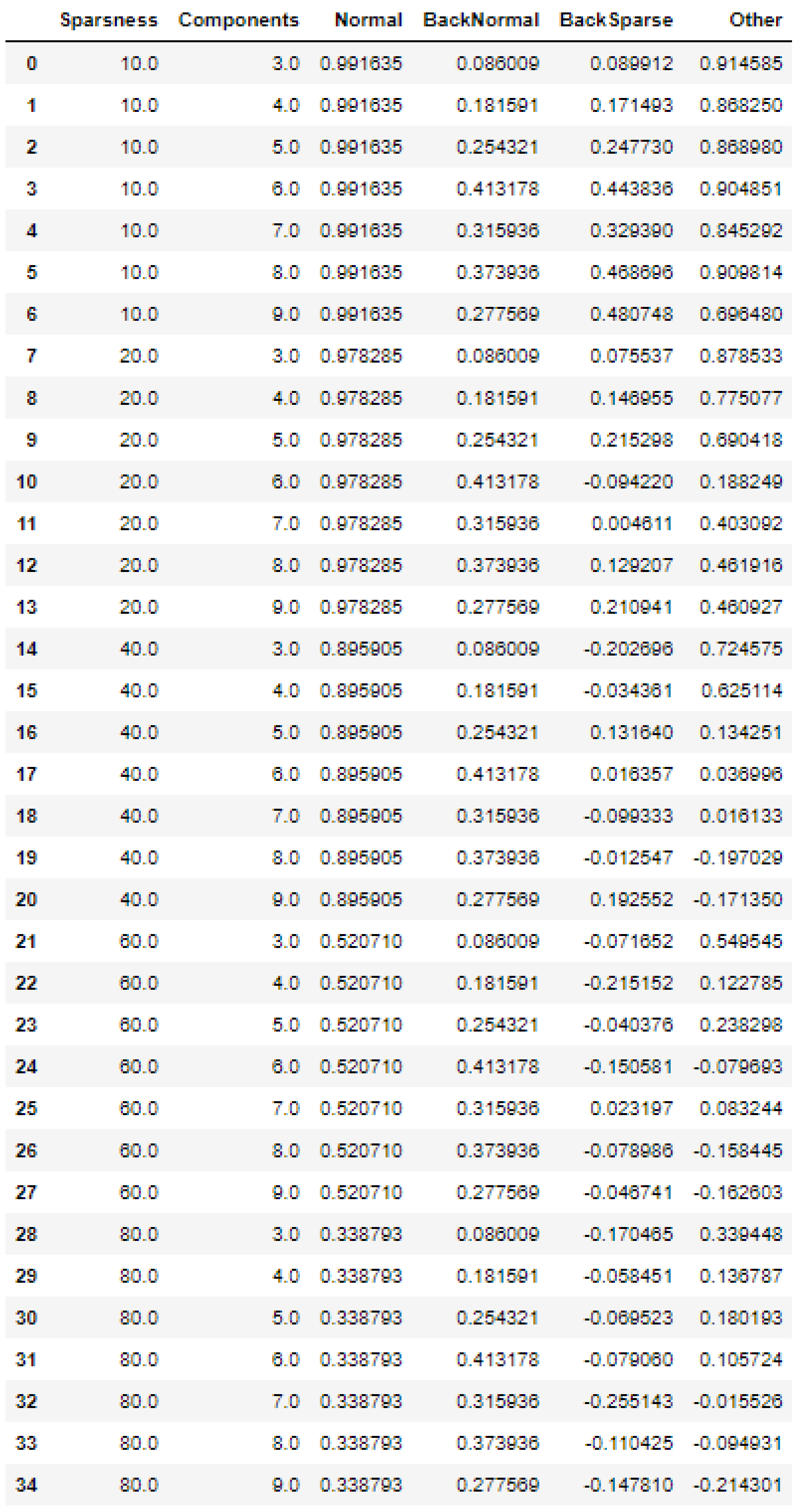
	Sparsness	Components	Normal	BackNormal	BackSparse	Other
0	10.0	3.0	0.972097	0.124551	0.323432	0.150089
1	10.0	4.0	0.972097	0.144259	0.307400	0.421874
2	10.0	5.0	0.972097	0.140418	0.278906	0.126563
3	10.0	6.0	0.972097	0.161041	0.273603	0.414726
4	10.0	7.0	0.972097	0.197432	0.256101	0.450680
5	10.0	8.0	0.972097	0.215520	0.263845	0.462783
6	10.0	9.0	0.972097	0.261633	0.286398	0.549543
7	20.0	3.0	0.898073	0.124551	0.236106	0.320459
8	20.0	4.0	0.898073	0.144259	0.151685	0.584213
9	20.0	5.0	0.898073	0.140418	0.123625	0.271661
10	20.0	6.0	0.898073	0.161041	0.122259	0.430396
11	20.0	7.0	0.898073	0.197432	0.143139	0.533348
12	20.0	8.0	0.898073	0.215520	0.206104	0.571800
13	20.0	9.0	0.898073	0.261633	0.186966	0.502109
14	40.0	3.0	0.733447	0.124551	0.115158	0.184723
15	40.0	4.0	0.733447	0.144259	0.012578	0.107281
16	40.0	5.0	0.733447	0.140418	0.078550	0.077893
17	40.0	6.0	0.733447	0.161041	0.125298	0.115155
18	40.0	7.0	0.733447	0.197432	0.084795	0.223460
19	40.0	8.0	0.733447	0.215520	0.007357	0.233821
20	40.0	9.0	0.733447	0.261633	0.167100	0.289477
21	60.0	3.0	0.631757	0.124551	0.230029	-0.006827
22	60.0	4.0	0.631757	0.144259	0.245465	-0.029433
23	60.0	5.0	0.631757	0.140418	0.320973	-0.004173
24	60.0	6.0	0.631757	0.161041	0.377276	0.004095
25	60.0	7.0	0.631757	0.197432	0.348588	-0.008919
26	60.0	8.0	0.631757	0.215520	0.186254	-0.097819
27	60.0	9.0	0.631757	0.261633	0.125988	0.008267
28	80.0	3.0	0.382511	0.124551	NaN	NaN
29	80.0	4.0	0.382511	0.144259	NaN	NaN
30	80.0	5.0	0.382511	0.140418	NaN	NaN
31	80.0	6.0	0.382511	0.161041	NaN	NaN
32	80.0	7.0	0.382511	0.197432	NaN	NaN
33	80.0	8.0	0.382511	0.215520	NaN	NaN
34	80.0	9.0	0.382511	0.261633	NaN	NaN

On Credit card default with divide(From UCI repository)

```
dftemp
```

	Sparsness	Components	Normal	BackNormal	BackSparse	Other
0	10.0	3.0	0.993550	0.403175	0.823007	0.644029
1	10.0	4.0	0.993550	0.661327	0.835164	0.773542
2	10.0	5.0	0.993550	0.424691	0.798574	0.782150
3	10.0	6.0	0.993550	0.486153	0.811157	0.700032
4	10.0	7.0	0.993550	0.448946	0.774274	0.819702
5	10.0	8.0	0.993550	0.223161	0.463760	0.058786
6	10.0	9.0	0.993550	0.233023	0.214726	0.858616
7	20.0	3.0	0.979948	0.403175	0.792108	0.475784
8	20.0	4.0	0.979948	0.661327	0.782743	0.628076
9	20.0	5.0	0.979948	0.424691	0.823562	0.718530
10	20.0	6.0	0.979948	0.486153	0.820799	0.660147
11	20.0	7.0	0.979948	0.448946	0.812271	0.646775
12	20.0	8.0	0.979948	0.223161	0.597543	0.105388
13	20.0	9.0	0.979948	0.233023	0.283557	-0.164576
14	40.0	3.0	0.899432	0.403175	-0.767530	-0.325560
15	40.0	4.0	0.899432	0.661327	-0.729627	-0.531047
16	40.0	5.0	0.899432	0.424691	-0.547671	-0.247627
17	40.0	6.0	0.899432	0.486153	-0.201031	0.142907
18	40.0	7.0	0.899432	0.448946	-0.137309	0.502935
19	40.0	8.0	0.899432	0.223161	-0.089769	0.055205
20	40.0	9.0	0.899432	0.233023	0.037797	0.082618
21	60.0	3.0	0.480287	0.403175	-0.155780	0.542291
22	60.0	4.0	0.480287	0.661327	0.549634	0.678536
23	60.0	5.0	0.480287	0.424691	0.192653	0.765951
24	60.0	6.0	0.480287	0.486153	0.017236	0.053887
25	60.0	7.0	0.480287	0.448946	0.520723	-0.095513
26	60.0	8.0	0.480287	0.223161	-0.047628	-0.043470
27	60.0	9.0	0.480287	0.233023	-0.056800	-0.436781
28	80.0	3.0	0.263438	0.403175	0.138020	0.224034
29	80.0	4.0	0.263438	0.661327	-0.261902	-0.080406
30	80.0	5.0	0.263438	0.424691	0.182917	0.409963
31	80.0	6.0	0.263438	0.486153	0.170550	0.068546
32	80.0	7.0	0.263438	0.448946	0.408778	0.489700
33	80.0	8.0	0.263438	0.223161	-0.012095	0.183548
34	80.0	9.0	0.263438	0.233023	0.144886	0.042910

On Credit card default without divide(From UCI repository)



	Sparsness	Components	Normal	BackNormal	BackSparse	Other
0	10.0	3.0	0.991635	0.086009	0.089912	0.914585
1	10.0	4.0	0.991635	0.181591	0.171493	0.868250
2	10.0	5.0	0.991635	0.254321	0.247730	0.868980
3	10.0	6.0	0.991635	0.413178	0.443836	0.904851
4	10.0	7.0	0.991635	0.315936	0.329390	0.845292
5	10.0	8.0	0.991635	0.373936	0.468696	0.909814
6	10.0	9.0	0.991635	0.277569	0.480748	0.696480
7	20.0	3.0	0.978285	0.086009	0.075537	0.878533
8	20.0	4.0	0.978285	0.181591	0.146955	0.775077
9	20.0	5.0	0.978285	0.254321	0.215298	0.690418
10	20.0	6.0	0.978285	0.413178	-0.094220	0.188249
11	20.0	7.0	0.978285	0.315936	0.004611	0.403092
12	20.0	8.0	0.978285	0.373936	0.129207	0.461916
13	20.0	9.0	0.978285	0.277569	0.210941	0.460927
14	40.0	3.0	0.895905	0.086009	-0.202696	0.724575
15	40.0	4.0	0.895905	0.181591	-0.034361	0.625114
16	40.0	5.0	0.895905	0.254321	0.131640	0.134251
17	40.0	6.0	0.895905	0.413178	0.016357	0.036996
18	40.0	7.0	0.895905	0.315936	-0.099333	0.016133
19	40.0	8.0	0.895905	0.373936	-0.012547	-0.197029
20	40.0	9.0	0.895905	0.277569	0.192552	-0.171350
21	60.0	3.0	0.520710	0.086009	-0.071652	0.549545
22	60.0	4.0	0.520710	0.181591	-0.215152	0.122785
23	60.0	5.0	0.520710	0.254321	-0.040376	0.238298
24	60.0	6.0	0.520710	0.413178	-0.150581	-0.079693
25	60.0	7.0	0.520710	0.315936	0.023197	0.083244
26	60.0	8.0	0.520710	0.373936	-0.078986	-0.158445
27	60.0	9.0	0.520710	0.277569	-0.046741	-0.162603
28	80.0	3.0	0.338793	0.086009	-0.170465	0.339448
29	80.0	4.0	0.338793	0.181591	-0.058451	0.136787
30	80.0	5.0	0.338793	0.254321	-0.069523	0.180193
31	80.0	6.0	0.338793	0.413178	-0.079060	0.105724
32	80.0	7.0	0.338793	0.315936	-0.255143	-0.015526
33	80.0	8.0	0.338793	0.373936	-0.110425	-0.094931
34	80.0	9.0	0.338793	0.277569	-0.147810	-0.214301