Data Mining Project 2: Frequent Pattern Analysis

Report

Kiran Nanjundaswamy
U00833551

The input file for this report is BANK_MARKET.txt

Preprocessing: Before frequent pattern mining, we need to transform D using discretization.

To discretize the data I have used weka's discretization packages. Here I have introduced a function which helps us to choose between Equi-Density and Equi-Width Binning Methods. For the purposes of this report I have used Equi-Width Binning containing 5 bins.

The output of this task can be found in 2 file "DiscretizationMap.csv" & "DiscretizedD.csv"

Below are the sample outputs of the 2 files

DiscretizationMap.csv

	A	B	C	D	E	F	G	H	I
1	G1	'(-inf-33.2]'	1						
2	G1	'(33.2-49.4]'	2						
3	G1	'(49.4-65.6]'	3						
4	G1	'(65.6-81.8]'	4						
5	G1	'(81.8-inf)'	5						
6	G2	'(-inf-2.2]'	6						
7	G2	'(2.2-4.4]'	7						
8	G2	'(4.4-6.6]'	8						
9	G2	'(6.6-8.8]'	9						
10	G2	'(8.8-inf)'	10						
11	G3	'(-inf-0.6]'	11						
12	G3	'(0.6-1.2]'	12						
13	G3	'(1.2-1.8]'	13						
14	G3	'(1.8-2.4]'	14						
15	G3	'(2.4-inf)'	15						
16	G4	'(-inf-1.4]'	16						
17	G4	'(1.4-2.8]'	17						
18	G4	'(2.8-4.2]'	18						
19	G4	'(4.2-5.6]'	19						
20	G4	'(5.6-inf)'	20						
21	G5	'(-inf-0.4]'	21						

DiscretizationD.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	3	6	12	16	23	28	33	36	41	46	51	56	61	66	71
2	3	6	12	17	21	28	33	36	41	46	51	56	61	66	71
3	2	6	12	17	23	30	33	36	41	46	51	56	61	66	71
4	2	7	12	18	23	28	33	36	41	46	51	56	61	66	71
5	3	6	12	17	23	28	35	36	41	46	51	56	61	66	71
6	2	6	12	19	21	28	33	36	41	46	51	56	61	66	71
7	3	7	12	18	23	28	33	36	41	46	51	56	61	66	71
8	2	7	12	16	21	28	33	36	41	46	51	56	61	66	71
9	1	8	14	18	23	30	33	36	41	46	51	56	61	66	71
10	1	6	14	17	23	30	33	36	41	46	51	56	61	66	71
11	2	7	12	16	21	28	33	36	41	46	51	56	61	66	71
12	1	6	14	17	23	30	33	36	41	46	51	56	61	66	71
13	1	7	14	17	23	28	35	36	41	46	51	56	61	66	71
14	3	6	15	16	23	30	33	36	41	46	51	56	61	66	71
15	2	7	12	18	23	30	33	36	41	46	51	56	61	66	71
16	3	8	12	19	21	30	35	36	41	46	51	56	61	66	71
17	2	7	12	18	23	30	33	36	41	46	51	56	61	66	71
18	2	7	12	18	21	30	35	36	41	46	51	56	61	66	71
19	3	7	12	19	23	30	35	36	41	46	51	56	61	66	71

<u>Task 1:</u> Implement a function to compute the closed patterns and their minimal generators

To perform this task I have used the FPClose algorithm from SPMF data mining package which I found online (http://www.philippe-fournier-viger.com/spmf/).

This package helped me to find the frequent patterns and making some futher changes to this algorithm, I was able to generate the closed patterns and their support. These details have been stored in the file "ClosedAndMG.csv"

Below is a sample output of the file

	A	B	C	D	E	F	G	H	I	J
1	61 30 #SUP: 50.36%									
2	66 30 #SUP: 50.99%									
3	56 66 30 #SUP: 50.35%									
4	51 30 #SUP: 51.14%									
5	56 51 30 #SUP: 50.5%									
6	56 30 #SUP: 51.73%									
7	30 #SUP: 52.38%									
8	51 66 61 71 100 95 80 #SUP: 56.79%									
9	56 51 66 61 71 100 95 80 #SUP: 55.48%									
10	56 66 61 71 100 95 80 #SUP: 56.93%									
11	66 61 71 100 95 80 #SUP: 58.26%									
12	51 66 61 71 100 95 #SUP: 58.21%									
13	56 51 66 61 71 100 95 #SUP: 56.89%									
14	56 66 61 71 100 95 #SUP: 58.38%									
15	66 61 71 100 95 #SUP: 59.72%									
16	51 66 61 100 95 #SUP: 58.52%									
17	56 51 66 61 100 95 #SUP: 57.2%									
18	56 66 61 100 95 #SUP: 58.7%									
19	66 61 100 95 #SUP: 60.03%									

<u>Task 2:</u> Function to compute bit-set representations of the matching datasets and the Jaccard Similarity

For this task I have designed a function which will build the bit set representations of the closed patterns and the data sets and also find the jaccard similarity for the Closed patterns. This function will be used in task 4.

Below are the sample output for the above task which show us how the data is computed

BitSet Representation

BitSetOperation.txt


```
{61 30} 00100000110101111111001111010000101010111110010101100000101110110001111011011011010010001011100010000100000011110011
000111100110011110110000111111110111011011101011000010010010001010011100000101001110110010101001100011110010110000001111001
0000001000010001110110100000000100010001000000101101110110110010000001110111100110000111001010001110111110110111011000011000
0101111010100011001001010010011100000000010001101010100001110111000100110001101100100010000000101101100001101011001010000000
0101100101100000111100011001110011111000000100101000100001000000111001000010000010110010000000110000000000000010001000101011010
001000000100111001110001100100000111011011101001010101000100100101010011010010101010111111011100000111011011110000000011010
10011010110100010111011101010101101100100100000110010010110010101001000101011000001101001000111100010101111011101111101111
01111011101101101111110111110111111111101000100101100110000000001000011111101111111111110101111000000111110000010100001011
001111110100101001010001000011110000100011100000101010000010101001001011000100011000011010100100100111001000001011011111110
100110110100111111000100011010010000100110001110110001011100011000001001100010011000000000000010100010101011111001001010010
01111110100010000000000000110000000000011011011111101110000001001111100110001111110001110011100101001100011101000101001011110
10011101001100110000111011101010001110000101000010000100000000000000001000000001111010100110100000000110000011111101111010010
010001100110111100100011110101011101011000001100010010101101110111110100110100110101100101000110011111100100011110001011110
10010010010010010100010111000000100100110111100011000010011001100010001010000001000011111100110000100101100100110110111000000
0000000100000110100000001100000101110100101001011110011100100110101110111010010001010011110001010110000000111101100010101111
1101111011010110111000101111101100001000101110111111101011101000000000110001110111001101100010100110110110001100001101000110
100000001010100010111100100100011000101110011010101111111101110111011000110010101111111110000100100000001000100110000011000
10000000000100000001000101100011011100100010111111011001000010010100100001100000111111111000111111101110001010110000000111
000001001100110110011001011100011000000011100000011101100111001110100001010011011001101001011100100101001100001011000010000
01110110001001000000111101010101111111000010010110101111010000000100010000000111010000000101100000001010010010011010110000001110
```


Jaccard Similarity

BitSetJaccardSimilarity.txt

```
{51 30} & {56 51 30} : 1
{56 51 30} & {56 30} : 1
{56 30} & {30} : 1
{30} & {51 66 61 71 100 95 80} : 0.49
{51 66 61 71 100 95 80} & {56 51 66 61 71 100 95 80} : 1
{56 51 66 61 71 100 95 80} & {56 66 61 71 100 95 80} : 1
{56 66 61 71 100 95 80} & {66 61 71 100 95 80} : 1
{66 61 71 100 95 80} & {51 66 61 71 100 95} : 0.97
{51 66 61 71 100 95} & {56 51 66 61 71 100 95} : 1
{56 51 66 61 71 100 95} & {56 66 61 71 100 95} : 1
{56 66 61 71 100 95} & {66 61 71 100 95} : 1
{66 61 71 100 95} & {51 66 61 100 95} : 1
{51 66 61 100 95} & {56 51 66 61 100 95} : 1
{56 51 66 61 100 95} & {56 66 61 100 95} : 1
{56 66 61 100 95} & {66 61 100 95} : 1
{66 61 100 95} & {51 66 100 95} : 1
{51 66 100 95} & {56 51 66 100 95} : 1
{56 51 66 100 95} & {56 66 100 95} : 1
```

Task 3: Function to compute the closed Emerging Patterns (EP) with high growthRate

In this task I have designed a function which will compute the emerging patterns and determine their growth rate. The growth rate is calculated by using the given equation max(sup(P,C1)/sup(P,C2), sup(P,C2)/sup(P,C1))

This function will be used in task 4.

Below is the sample output for the task

GrowthRate.txt

```
{51 66 61 71 100 95 80} Growth Rate :2.628
{56 51 66 61 71 100 95 80} Growth Rate :2.628
{56 66 61 71 100 95 80} Growth Rate :2.628
{66 61 71 100 95 80} Growth Rate :2.628
{51 66 61 71 100 95} Growth Rate :2.514
{56 51 66 61 71 100 95} Growth Rate :2.514
{56 66 61 71 100 95} Growth Rate :2.514
{66 61 71 100 95} Growth Rate :2.514
{51 66 61 100 95} Growth Rate :2.514
{56 51 66 61 100 95} Growth Rate :2.514
{56 66 61 100 95} Growth Rate :2.514
{66 61 100 95} Growth Rate :2.514
{51 66 100 95} Growth Rate :2.514
{56 51 66 100 95} Growth Rate :2.514
{56 66 100 95} Growth Rate :2.514
{66 100 95} Growth Rate :2.514
{51 66 61 71 33 100} Growth Rate :2.499
{56 51 66 61 71 33 100} Growth Rate :2.499
{56 66 61 71 33 100} Growth Rate :2.499
{66 61 71 33 100} Growth Rate :2.499
{51 66 61 33 100} Growth Rate :2.499
{56 51 66 61 33 100} Growth Rate :2.499
{56 66 61 33 100} Growth Rate :2.499
```

Task 4: Implement a function to compute a diversified set PS of K closed EP

In this task we compute the objective function using the given formula and the functions which were designed in task 2 and task 3.

This is a heavy computation and takes a considerable amount of time for computation.

To help in speeding up the process of computation I have computed the emerging closed patterns and their growth rate as well as the bit set representation and sorted them in a hash map for easy access. This has reduced the computation time of my function to less than 5 min.

We have generated 2 files to present the output of the function. "PSkEPs.csv" which consists of the closed emerging patterns and their growth rate along with their supports in C1 and C2 and "PSkEPJaccard.csv" which consists of the jaccard similarity of the K emerging patterns.

Below are the sample outputs of the files.

PSkEPs.csv

1	Objective Function Result: 0.15561449941278102			
2	{51 66 61 71 100 95 80}	GrowthRate: 2.628	SuppC1: 0.626	SuppC2: 0.238
3	{56 51 66 61 71 100 95 80}	GrowthRate: 2.628	SuppC1: 0.626	SuppC2: 0.238
4	{56 66 61 71 100 95 80}	GrowthRate: 2.628	SuppC1: 0.626	SuppC2: 0.238
5	{66 61 71 100 95 80}	GrowthRate: 2.628	SuppC1: 0.626	SuppC2: 0.238
6	{51 66 61 71 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
7	{56 51 66 61 71 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
8	{56 66 61 71 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
9	{66 61 71 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
10	{51 66 61 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
11	{56 51 66 61 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
12	{56 66 61 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
13	{66 61 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
14	{51 66 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
15	{56 51 66 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
16	{56 66 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
17	{66 100 95}	GrowthRate: 2.514	SuppC1: 0.644	SuppC2: 0.256
18	{51 66 61 71 33 100}	GrowthRate: 2.499	SuppC1: 0.721	SuppC2: 0.289
19	{56 51 66 61 71 33 100}	GrowthRate: 2.499	SuppC1: 0.721	SuppC2: 0.289
20	{56 66 61 71 33 100}	GrowthRate: 2.499	SuppC1: 0.721	SuppC2: 0.289
21	{66 61 71 33 100}	GrowthRate: 2.499	SuppC1: 0.721	SuppC2: 0.289
22	{51 66 61 33 100}	GrowthRate: 2.499	SuppC1: 0.721	SuppC2: 0.289
23	{56 51 66 61 33 100}	GrowthRate: 2.499	SuppC1: 0.721	SuppC2: 0.289

PSkEPJaccard.csv

1	{51 66 61 71 100 95 80}	{56 51 66 61 71 100 95 80}	Jaccard Similarity: 1
2	{56 51 66 61 71 100 95 80}	{56 66 61 71 100 95 80}	Jaccard Similarity: 1
3	{56 66 61 71 100 95 80}	{66 61 71 100 95 80}	Jaccard Similarity: 1
4	{66 61 71 100 95 80}	{51 66 61 71 100 95}	Jaccard Similarity: 0.97
5	{51 66 61 71 100 95}	{56 51 66 61 71 100 95}	Jaccard Similarity: 1
6	{56 51 66 61 71 100 95}	{56 66 61 71 100 95}	Jaccard Similarity: 1
7	{56 66 61 71 100 95}	{66 61 71 100 95}	Jaccard Similarity: 1
8	{66 61 71 100 95}	{51 66 61 100 95}	Jaccard Similarity: 1
9	{51 66 61 100 95}	{56 51 66 61 100 95}	Jaccard Similarity: 1
10	{56 51 66 61 100 95}	{56 66 61 100 95}	Jaccard Similarity: 1
11	{56 66 61 100 95}	{66 61 100 95}	Jaccard Similarity: 1
12	{66 61 100 95}	{51 66 100 95}	Jaccard Similarity: 1
13	{51 66 100 95}	{56 51 66 100 95}	Jaccard Similarity: 1
14	{56 51 66 100 95}	{56 66 100 95}	Jaccard Similarity: 1
15	{56 66 100 95}	{66 100 95}	Jaccard Similarity: 1
16	{66 100 95}	{51 66 61 71 33 100}	Jaccard Similarity: 0.893
17	{51 66 61 71 33 100}	{56 51 66 61 71 33 100}	Jaccard Similarity: 1
18	{56 51 66 61 71 33 100}	{56 66 61 71 33 100}	Jaccard Similarity: 1
19	{56 66 61 71 33 100}	{66 61 71 33 100}	Jaccard Similarity: 1
20	{66 61 71 33 100}	{51 66 61 33 100}	Jaccard Similarity: 1
21	{51 66 61 33 100}	{56 51 66 61 33 100}	Jaccard Similarity: 1
22	{56 51 66 61 33 100}	{56 66 61 33 100}	Jaccard Similarity: 1
23	{56 66 61 33 100}	{66 61 33 100}	Jaccard Similarity: 1
24	{66 61 33 100}	{51 66 33 100}	Jaccard Similarity: 1

Extra Credit

Task 5: Closed pattern where a new local classifier built for mds(P) can significantly reduce the classification error on mds(P) made by a fixed global classifier

Here I have used implementations of NBC to compute the error rate for the closed patterns and used that to compute the objective function for task 5
This is a heavy computation function which takes a lot of time to process the results

The results are stored in the files "PSkCPsWithError.csv" which consists of the closed patterns and their error rate & "PSkCPwithErrorJaccard.csv" which consists of the Jaccard similarity of the K closed patterns.

PSkCPsWithError.csv

Objective Function Result: 0.042489836613814023	
{61 30}	ErrorRate: 0.13479187911374804
{66 30}	ErrorRate: 0.13479187911374804
{56 66 30}	ErrorRate: 0.13479187911374804
{51 30}	ErrorRate: 0.13479187911374804
{56 51 30}	ErrorRate: 0.13479187911374804
{56 30}	ErrorRate: 0.13479187911374804
{30}	ErrorRate: 0.13479187911374804
{51 66 61 71 100 95 80}	ErrorRate: 0.041840306717786296
{56 51 66 61 71 100 95 80}	ErrorRate: 0.041840306717786296
{56 66 61 71 100 95 80}	ErrorRate: 0.041840306717786296
{66 61 71 100 95 80}	ErrorRate: 0.041840306717786296
{51 66 61 71 100 95}	ErrorRate: 0.0676074562290243
{56 51 66 61 71 100 95}	ErrorRate: 0.0676074562290243
{56 66 61 71 100 95}	ErrorRate: 0.0676074562290243

PSkCPwithErrorJaccard.csv

{61 30}	{66 30}	Jaccard Similarity: 1
{66 30}	{56 66 30}	Jaccard Similarity: 1
{56 66 30}	{51 30}	Jaccard Similarity: 1
{51 30}	{56 51 30}	Jaccard Similarity: 1
{56 51 30}	{56 30}	Jaccard Similarity: 1
{56 30}	{30}	Jaccard Similarity: 1
{30}	{51 66 61 71 100 95 80}	Jaccard Similarity: 0.493
{51 66 61 71 100 95 80}	{56 51 66 61 71 100 95 80}	Jaccard Similarity: 1
{56 51 66 61 71 100 95 80}	{56 66 61 71 100 95 80}	Jaccard Similarity: 1
{56 66 61 71 100 95 80}	{66 61 71 100 95 80}	Jaccard Similarity: 1
{66 61 71 100 95 80}	{51 66 61 71 100 95}	Jaccard Similarity: 0.97
{51 66 61 71 100 95}	{56 51 66 61 71 100 95}	Jaccard Similarity: 1
{56 51 66 61 71 100 95}	{56 66 61 71 100 95}	Jaccard Similarity: 1
{56 66 61 71 100 95}	{66 61 71 100 95}	Jaccard Similarity: 1
{66 61 71 100 95}	{51 66 61 100 95}	Jaccard Similarity: 1
{51 66 61 100 95}	{56 51 66 61 100 95}	Jaccard Similarity: 1
{56 51 66 61 100 95}	{56 66 61 100 95}	Jaccard Similarity: 1
{56 66 61 100 95}	{66 61 100 95}	Jaccard Similarity: 1
{66 61 100 95}	{51 66 100 95}	Jaccard Similarity: 1
{51 66 100 95}	{56 51 66 100 95}	Jaccard Similarity: 1
{56 51 66 100 95}	{56 66 100 95}	Jaccard Similarity: 1

<u>Task 6:</u> Handle another data mining task such as regression, clustering, and outlier detection

<u>Linear Regression</u>
I have used weka packages to perform data mining task Linear Regression which has the output generated in “LinerRegression.txt”

LinerRegression.txt

```
LinerRegression.txt
      0.1804 * G10 +
     -0.0011 * G11 +
      0.3525 * G12 +
      1.7698 * G13 +
     -5.6672 * G14 +
      2.3879 * G15 +
      4.316  * G16 +
    -41.3482 * G17 +
     -3.2042 * G18 +
     47.1571 * G19 +
   8743.2017

Regression Analysis:

Variable     Coefficient     SE of Coef        t-Stat
G2               -0.0598        0.0265        -2.2537
G5               -0.701         0.156         -4.4946
G6               -0.5982        0.1135        -5.2717
G8               -1.4501        0.2141        -6.7732
G9               -1.1306        0.0354       -31.9165
G10               0.1804        0.0437         4.1254
G11              -0.0011        0.0002        -4.3957
G12               0.3525        0.0226        15.5731
G13               1.7698        0.465          3.8061
G14              -5.6672        0.2993       -18.9352
G15               2.3879        0.2019        11.8273
G16               4.316         0.2661        16.2204
G17             -41.3482        0.2521      -164.0393
G18              -3.2042        0.016       -199.9852
G19              47.1571        0.2217       212.7244
const          8743.2017       24.0412       363.6756

Degrees of freedom = 41172
R^2 value = 0.97
Adjusted R^2 = 0.97001
F-statistic = 88806.1604
```

<u>Outlier Detection</u>
The next task I have implemented is the outlier detection. I have used weka’s interquartile range method to help in this function.

The function I have designed outputs 2 files. OutlierDetection.txt which contains the result in the arff format and you can see 2 new columns as outlier and extreme values which contains nominal values yes or no which indicate outlier and extreme values for the instances.

Below is the sample output of the file “Outlier.txt”

OutlierDetection.txt

```
@attribute G8 numeric
@attribute G9 numeric
@attribute G10 numeric
@attribute G11 numeric
@attribute G12 numeric
@attribute G13 numeric
@attribute G14 numeric
@attribute G15 numeric
@attribute G16 numeric
@attribute G17 numeric
@attribute G18 numeric
@attribute G19 numeric
@attribute G20 numeric
@attribute Outlier {no,yes}
@attribute ExtremeValue {no,yes}

@data
0,56,1,1,1,1,1,1,1,2,2,261,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,no
0,57,2,1,2,0,1,1,1,2,2,149,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,yes
0,37,2,1,2,1,2,1,1,2,2,226,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,no
0,40,3,1,3,1,1,1,1,2,2,151,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,no
0,56,2,1,2,1,1,2,1,2,2,307,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,yes
0,45,2,1,5,0,1,1,1,2,2,198,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,yes
0,59,3,1,4,1,1,1,1,2,2,139,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,no
0,41,4,1,0,0,1,1,1,2,2,217,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,yes
0,24,6,2,4,1,2,1,1,2,2,380,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,no
0,25,2,2,2,1,2,1,1,2,2,50,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,no
0,41,4,1,0,0,1,1,1,2,2,55,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,yes
0,25,2,2,2,1,2,1,1,2,2,222,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,no
0,29,4,2,2,1,1,2,1,2,2,137,1,0,0,0,1.1,93.994,-36.4,4.857,5191,no,yes
```

Clustering

For implementing clustering I have used Simple K Means method of Weka which helped in clustering the data. The output can be found in "Clustering.txt "

Clustering.txt

```

kMeans
======

Number of iterations: 16
Within cluster sum of squared errors: 32084.723357240804

Initial starting points (random):

Cluster 0: 0,31,7,1,6,1,2,1,2,9,2,37,1,0,0,0,-0.1,93.2,-42,4.191
Cluster 1: 0,42,3,2,6,1,1,1,2,5,2,401,2,0,0,0,1.4,93.918,-42.7,4.96
Cluster 2: 0,54,4,2,1,0,2,1,1,2,6,1171,1,0,0,0,1.1,93.994,-36.4,4.857
Cluster 3: 0,40,4,1,3,1,2,1,1,4,2,104,4,0,0,0,1.4,94.465,-41.8,4.96
Cluster 4: 0,33,2,1,2,0,1,2,1,2,2,189,2,0,0,0,1.1,93.994,-36.4,4.857

Missing values globally replaced with mean/mode

Final cluster centroids:
                         Cluster#
Attribute   Full Data          0          1          2          3          4
            (41188.0)  (13225.0)  (13972.0)   (5550.0)   (4314.0)   (4127.0)
============================================================================
class          0.1127     0.2495      0.057     0.0416      0.042      0.032
G1            40.0241    39.7144    40.2026    40.1569    39.4379    40.8459
G2             4.6589     4.8056     4.6441     4.5681     5.1748     3.8221
G3             1.5029     1.5525     1.5046     1.4477     1.4826     1.4332
G4              3.749     3.7724     3.9619     3.5737     5.1949     1.6775
G5             0.7913      0.895     0.7823     0.7072     0.7594     0.6363
G6             1.4998      1.542     1.5301     1.4227     1.4335     1.4349
G7             1.1277     1.1293      1.138     1.1124      1.118      1.118
G8             1.6347     1.9204          2          1          1          1
G9             4.2362     3.2281     6.2898     3.1932     3.1813     3.0199
```