CmpE 482 Numerical Linear Algebra and Its Applications Programming Projects

Instructor: A. Taylan Cemgil
Department of Computer Engineering, Boğaziçi University
34342 Bebek, Istanbul, Turkey
taylan.cemgil@boun.edu.tr

as of: May 11, 2016

Abstract

This document defines the programming projects of cmpe482.

1 Project 1: Deadline 3 June, 2016 10:00

In this project, you need to design and implement a method for predicting the amount of sales of several related items, given historical monthly sales figures. More precisely, for the i'th item, you are given the number of sales at the t'th month as

$$x_{i,t}$$

You will write an jupyter notebook (using either the python or octave kernel) that will do the following tasks:

1. Read historical sales data and generates various plots like in Figure 1. The data will be a matrix in the workspace. You should automatically determine the number of items and the number of months in the data.
2. Construct a basis regression model. When constructing your model, assume that there is a constant term, a linear trend and seasonal fluctuations (yearly, two-yearly and every 4 months).
3. For each least-squares method (1-Normal Equations, 2-QR factorization, 3-SVD based), your program will compute a prediction, using a basis regression model. You should generate predictions for an horizon of 6 months. For example, if your data contains 24 columns, you should generate predictions for months $25, 26, \ldots, 30$. You should check if predictions by each computation method are similar by checking the relative error $|x - y|/|y|$. Print a table like

```
Total relative error between predictions generated by different computational methods
NormE: Normal Equations
QR   : QR decomposition
SVD  : SVD Decomposition

          NormE     QR        SVD
NormE     0         0.0009    0.0001
QR        0.0134          0   0.0084
SVD       0.0002    0.0007          0
```

4. When a ground truth test data is given (case 2), you must read the specified file (in the example `'testdata1.txt'`) and compare your predictions in terms of the $\|\cdot\|_2$ norm of the error. Your program must also print the error norm for each item separately as.

```
Errors (Euclidian norms)
Item 1: 2.97
Item 2: 83.91
Item 3: 40.29
Item 4: 88.62
Item 5: 87.44
Total :303.23
```

5. When no ground truth test data is given (case 1), The predictions **must** be given as the output

```
%  salesdata1 and testdata1 are matrices in the workspace
case 1b >> predictions = project1(salesdata1)
case 2b >> predictions = project1(salesdata1, testdata1)
```

6. Instead of considering a basis function, use directly past data (all sales figures in a given time window of length D_τ from the past) as the observed features.

   You need to setup a least squares problem for the following problem and solve by least squares

$$x_{i,t} \approx \sum_{j=1}^{I} \sum_{\tau=1}^{D_\tau} w_{i,j,\tau} x_{j,t-\tau}$$

   where I is the number of items and D_τ is the number of steps you lookback for past data. The output of you program must be the weight array w of size $I \times I \times D_\tau$ and the prediction for the next month (a I vector) only.

7. Read historical sales data. The format of the data file will be ascii text where the figures are separated by a space and each item is located on a separate line. You can use `textread` function of octave/panda library of python. You should automatically determine the number of items and the number of months in the data. The 'depth' of the history D_τ needs to be given as an extra parameter. You must write a function named

   `project2.m`

   that is callable from like below,

```
case 1 >> w, pred = project_part2(D_tau, salesdata1)
case 2 >> w, pred = project_part2(D_tau, salesdata1, testdata1)
```

8. The output of you program must be the weight array w of size $I \times I \times D_\tau$ and the prediction for the next month (a I vector) only.
9. Your program must be able to cope with missing data specified as NaN's. You can use an alternating least squares approach as described in the class.

1.1 How to submit

Use the github and create a pull request.

1.2 Data Set 1

This data contains $i = 1 \ldots 5$ items and $t = 1 \ldots 24$ months.

412	399	371	371	388	386	360	365	381	376	350	354	379	375	361	370	400	402	386	398	420	412	384	385
502	521	498	488	506	515	476	463	482	499	479	486	538	567	559	557	595	607	575	568	596	613	591	595
869	895	893	873	884	942	947	937	946	994	989	970	997	1040	1062	1041	1065	1112	1118	1094	1088	1112	1095	1054
502	535	555	571	592	617	601	586	576	575	556	552	574	596	614	621	645	662	657	652	654	665	668	682
877	890	861	867	913	946	909	914	949	964	915	914	971	984	965	967	1028	1046	1013	1019	1058	1062	1016	1014

Table 1: Data Set 1. Sales for 5 items over a period of 24 months.

1.3 Data Set 2

This data contains $i = 1 \ldots 5$ items and $t = 1 \ldots 24$ months with missing values.

NaN	895	893	873	NaN	942	947	937	946	994	989	970	997	1040	1062	1041	1065	1112	1118	NaN	NaN	1112	1095	1054
502	535	555	571	NaN	617	601	586	576	575	556	552	574	596	614	621	645	662	657	NaN	NaN	665	668	682
412	399	371	371	388	386	360	365	381	376	350	354	379	375	361	370	400	402	386	398	420	412	NaN	385
502	521	498	NaN	506	515	476	463	482	499	479	486	538	567	559	557	595	607	575	568	596	613	591	595
877	890	861	867	913	946	909	914	949	964	915	914	971	984	965	967	1028	NaN	1013	NaN	NaN	NaN	1016	1014

Table 2: Data Set 2. Sales for 5 items over a period of 24 months, with missing values.

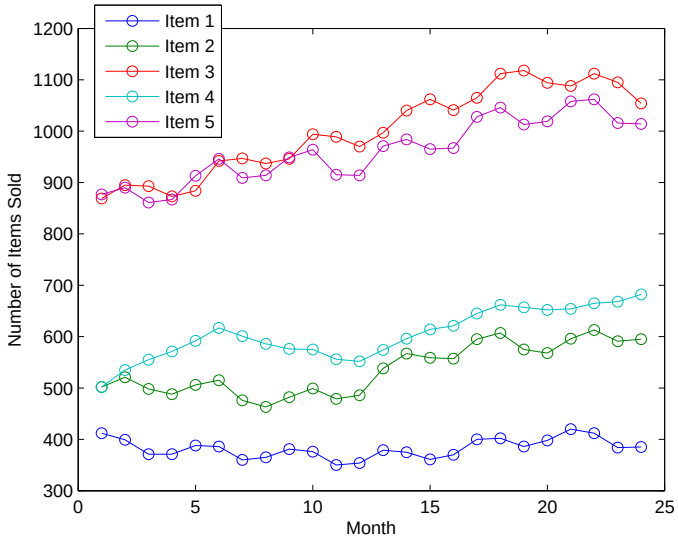


Figure 1: Data Set1

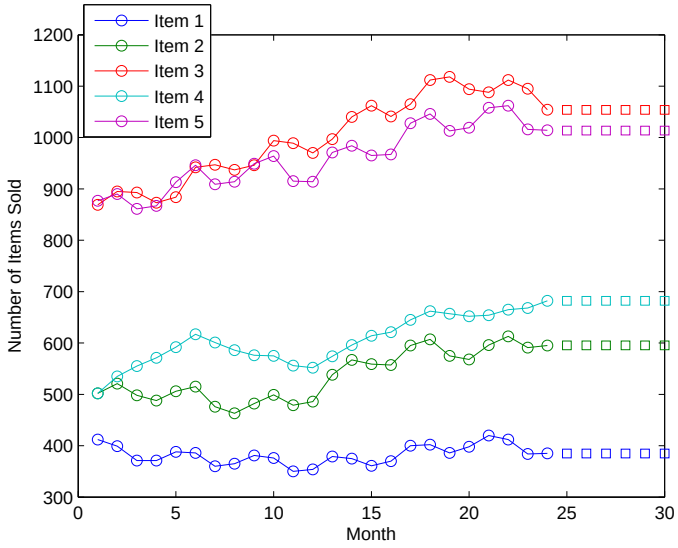


Figure 2: Observed sales (circles) and predictions (squares) on Data Set1, using a naive method that always predicts the next month as the sales of the previous month.

2 Project 2: Spectral Clustering,

Consider an undirected graph $G = (V, E)$ with vertex set $V = \{1, \ldots, N\}$ and edge set $E \subset V \times V$. Each edge $e = (i, j) \in E$ is has a nonnegative weight $w_{i,j} = w_{j,i}$. We will denote the $N \times N$ symmetric weight matrix as W. The degree of vertex i is $d_i = \sum_j w_{i,j}$. The degree matrix D is defined as the diagonal matrix with $D_{i,i} = d_i$.

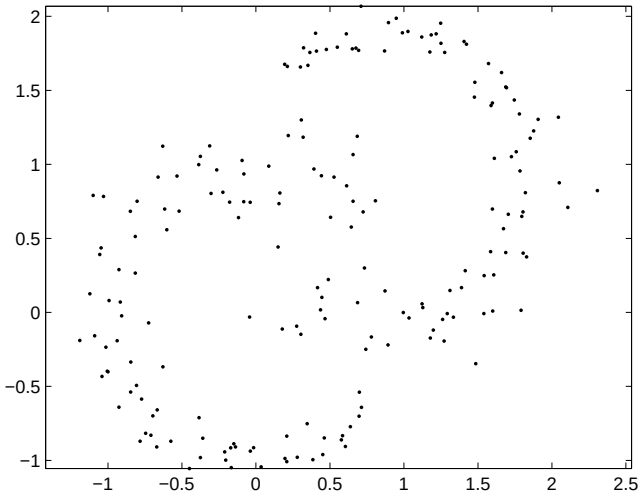
The (random walk) graph Laplacian (of G) is the matrix

$$L = I - WD^{-1}$$

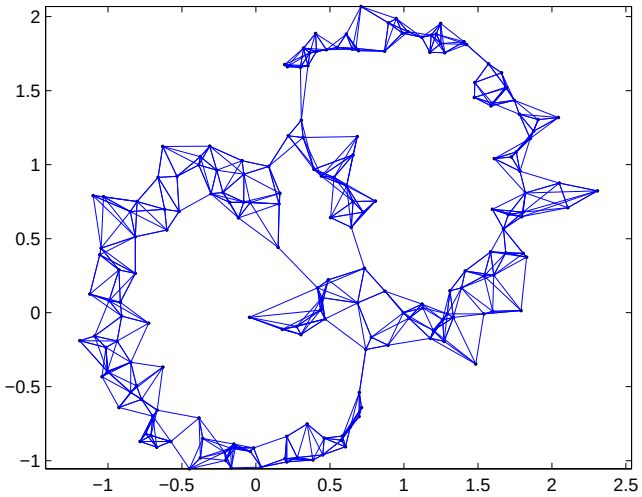
Note that, $T = WD^{-1}$ is a matrix where each column sums to one. Hence, T can be interpreted as the transition probabilities of a random walk. In other words, $T(i, j)$ gives us the probability of the next vertex to be visited is j, when our random walker is at vertex i.

2.1 Things to do

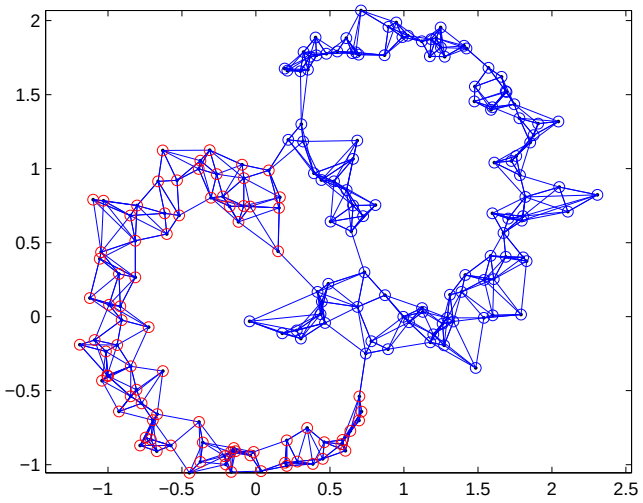
1. Obtain N points fas input, where each row i stores is a point $x_i \in \mathbb{R}^n$. Each point x_i will correspond to the vertex i in a graph G that we will construct.



2. Construct (and plot when $n = 2$) the so called k-nearest neighbor graph. Here, the goal is to connect vertex i with vertex j if the distance $d_{i,j} = \|x_i - x_j\|$ is among the k-nearest neighbors of i. However, this definition leads to a directed graph, as the neighborhood relationship is not symmetric. We will make this graph undirected by simply ignoring the directions of the edges, that is we connect i and j with an undirected edge if i is among the k-nearest neighbors of j or if j is among the k-nearest neighbors of i. Take $k = 5$



3. Construct the weight matrix. If an edge is a member of the k-nearest neighbor graph, take the weight to be $W_{i,j} = \exp(-\frac{1}{2}(d_{i,j}/\sigma)^2)$ with $\sigma = 0.3$.
4. Using an inverse iteration method find the second smallest eigenvalue and the corresponding eigenvector v of the Laplacian.
5. Plot the histogram of the entries of v. Roughly, you should see two 'hills' separated by a shallow 'valley' near the threshold value.
6. For each point $i = 1 \ldots N$, if $v_i < \tau$, assign i to cluster 1, else to cluster 2. Take the default $\tau = 0$. Generate the resulting clustering:



The output of your m file be an array of length N where each entry gives the cluster label.

```
>> [Labels, WeightMatrix] = project3(points, k, sigma, tau)
```

2.2 Example Dataset

```
  1.8041    0.6784
  2.0438    1.3186
  1.7267    1.0516
 -0.3849   -0.7118
  0.6754    1.7853
  0.5495    1.7915
  0.7096    2.0687
  1.5394   -0.0084
 -0.6608    0.9144
  1.7801    1.3404
  0.1777   -0.1121
 -1.0558    0.3911
  1.0338   -0.0374
  1.7965    0.6487
 -1.0000   -0.4010
 -0.7828   -0.8704
 -1.1899   -0.1892
 -1.0406   -0.4323
  1.5998    0.0093
  1.6721    0.5654
 -0.2042   -0.9981
  1.8053    0.3997
 -0.1179    0.6405
  1.8531    1.1769
 -0.7254   -0.0714
  1.1200    1.8603
  1.2711   -0.1937
  1.4852   -0.3463
  0.1551    0.7342
  0.2080   -0.8365
  0.0353   -1.0441
  0.7240    0.6787
  0.1960   -0.9874
  0.6123    0.8548
  1.6875    0.4047
  1.7907    0.0139
  0.4619   -0.8482
 -0.2108   -0.9427
  1.9062    1.3045
  0.6965   -0.7017
 -0.1498   -0.8879
  0.4032    1.8863
  1.1981   -0.1201
  0.5038    0.6427
 -0.8036    0.7512
 -0.3867    0.9984
 -0.9929    0.0796
  0.8957    1.9576
  0.6104    1.8812
  1.8283    0.3746
 -0.0384   -0.9373
 -1.0897   -0.1580
 -0.5749   -0.8707
 -1.0135   -0.2355
  1.5883    1.3974
 -0.1382   -0.9088
  1.6112    1.0404
  1.7054    0.6626
 -0.0826    0.7480
 -0.1706   -0.9147
 -0.8070   -0.4935
 -0.2230    0.8121
  1.0266    1.8979
  1.6877    1.5232
  1.6596    1.6200
  2.1073    0.7092
  0.6942    1.7704
 -0.9167    0.0697
```

```
-1.1005    0.7904
-0.5194    0.6836
-0.4495   -1.0545
 1.7844    0.9555
 0.3844   -0.9948
 0.3000    1.6577
 1.5980    1.4144
-0.9080   -0.0232
-0.6166    0.6981
 1.6930    1.5175
 0.9481    1.9875
 1.5416    0.2478
 0.9958   -0.0008
 0.2080   -1.0085
-0.6968   -0.6995
 0.5853   -0.8335
 0.5270    0.9143
 0.8920   -0.2195
 1.1748    1.7588
 0.0870    0.9883
-0.7729   -0.5863
-0.1664   -1.0489
 0.6559    1.0658
 0.4165    0.1672
 1.3890    0.1669
-0.8468   -0.5381
 0.3040   -0.1485
-0.0159   -0.9139
-1.0051   -0.3978
 0.8693    1.7659
-0.3754    1.0541
 0.6383   -0.7723
 0.6444    0.5768
 0.6842    1.1894
 0.1941    1.6765
 1.7583    1.0854
 0.3221    1.7879
-0.3032    0.8039
-1.1221    0.1256
 1.4760    1.4537
 1.2173    1.8821
 1.7455    1.4349
-1.0465    0.4358
-0.9252   -0.6406
 1.4218    1.8117
-0.8154    0.2656
 2.3067    0.8224
 0.7337    0.2999
 1.1844    1.8742
 1.8210    0.8082
 0.1613    0.8070
 1.4788    1.5544
-0.6026    0.5581
 1.8762    1.2258
 0.3064    1.2996
 0.4089    1.7655
 1.2606   -0.0482
 1.2922   -0.0085
 0.7427   -0.2490
 0.6985   -0.5397
-0.1776    0.7448
 1.4060    1.8302
 0.4664   -0.0426
 0.7799   -0.1661
 1.5858    0.4104
 0.2184    1.1946
 0.3921    0.9686
 0.3187    1.1836
 0.4366    0.0175
-0.7099   -0.8305
 1.5980    0.6986
 0.6562    0.7510
 0.3648    1.7566
 2.0491    0.8758
-0.6675   -0.6591
-0.7451   -0.8168
-0.8486    0.6826
-0.3124    1.1251
-0.0935    1.0268
 0.2788   -0.9795
 0.5770   -0.8618
-0.6706   -0.9095
-0.3759   -0.9815
 1.3104    0.1481
 1.2499    1.8183
 0.3512    1.6688
 0.2118    1.6620
 1.1275    0.0321
 1.2479    1.9534
 0.4427    0.9236
 0.6509    1.7808
-0.8461   -0.3358
 0.6869    0.0662
 1.5716    1.6811
 0.8724    0.1445
 0.6042   -0.9059
-0.9377   -0.1915
 0.1496    0.4422
 0.2753   -0.0930
-0.8153    0.5125
-0.6294    1.1229
 1.3345   -0.0321
 0.8068    0.7538
 1.2756    1.7557
-0.2652    0.9621
-0.9262    0.2884
 0.7134   -0.6414
-1.0300    0.7837
 0.4516   -0.9603
 1.1223    0.0580
 1.1776   -0.1736
 0.4456    0.1014
-0.3597   -0.8503
 0.4753    1.7764
```

```
 0.3456   -0.7520
-0.6290   -0.3676
-0.0431   -0.0311
-0.5333    0.9211
 1.6069    0.2535
 1.4141    0.2811
-0.0813    0.9349
-0.0388    0.7433
 0.4890    0.2214
 0.9895    1.8892
```