ELL888 Minor

Mridul Gupta

Monday 21 February 2022 IST

1 Q1

This problem has a straight forward approach. First generate the missing data and then use the data matrix as if it was a normal matrix to learn graph structure.

The data generation process itself, however, is non-trivial. And the exact method used depends a lot on the data being used as well as the mechanics of data being missing. Since in this case the data is missing at random, one can completely ignore the missing data and just use complete features [1].

Note that this only works when the fraction of data missing is not huge. For example if $70 - 80\%$ of data is missing, then we cannot ignore all the data. However, in such cases the best remedy is to collect more data properly.

Now, imputing new data can be done in several ways. One method is to just replace the missing data points with a default value. A little improvement over this is calculating the mean (or median) for each feature and replacing missing value in a feature with it its mean.

The mean uses the distribution information, and we can go one step further by learning the distribution/density on the features and sample random variables from the distribution. This works for iid samples and iid features.

This requires assumption of the form of distribution of the features. One popular missing data generating method is the EM-algorithm. The recursive EM-algorithm is more popularly used with the assumption that the density is a mixture of Gaussians.

Another method can be to use can be to learn d regression models, each for a feature. The k^{th} model uses features $\{i\}_{i \neq k}$ to predict the value of feature k. This method will inherently incorporate dependencies between features as well as the learn the density of the feature we are predicting. One problem might be some other feature $j \neq k$ might also have missing value; a solution to this is to use mean (or median) as the placeholder of feature j during prediction of feature k.

1.1 Experimentation

For this problem, I work with two datasets: seeds [2, 3] (from here) and wine [2, 4] (from here). Both of these are complete datasets with real valued features. Seeds dataset has 7 real valued features. The wine clustering dataset has 11 real valued and 2 integer valued features but I just drop them.

Since we need missing data, I randomly delete 1% of values within the dataset. I then create k GaussianMixture models trained using Expectation Maximization algorithm to learn densities of the k features. I then use the learned means and covariances to sample a random value to be used as an estimate of given feature. Graph is learned using the imputed feature matrix.

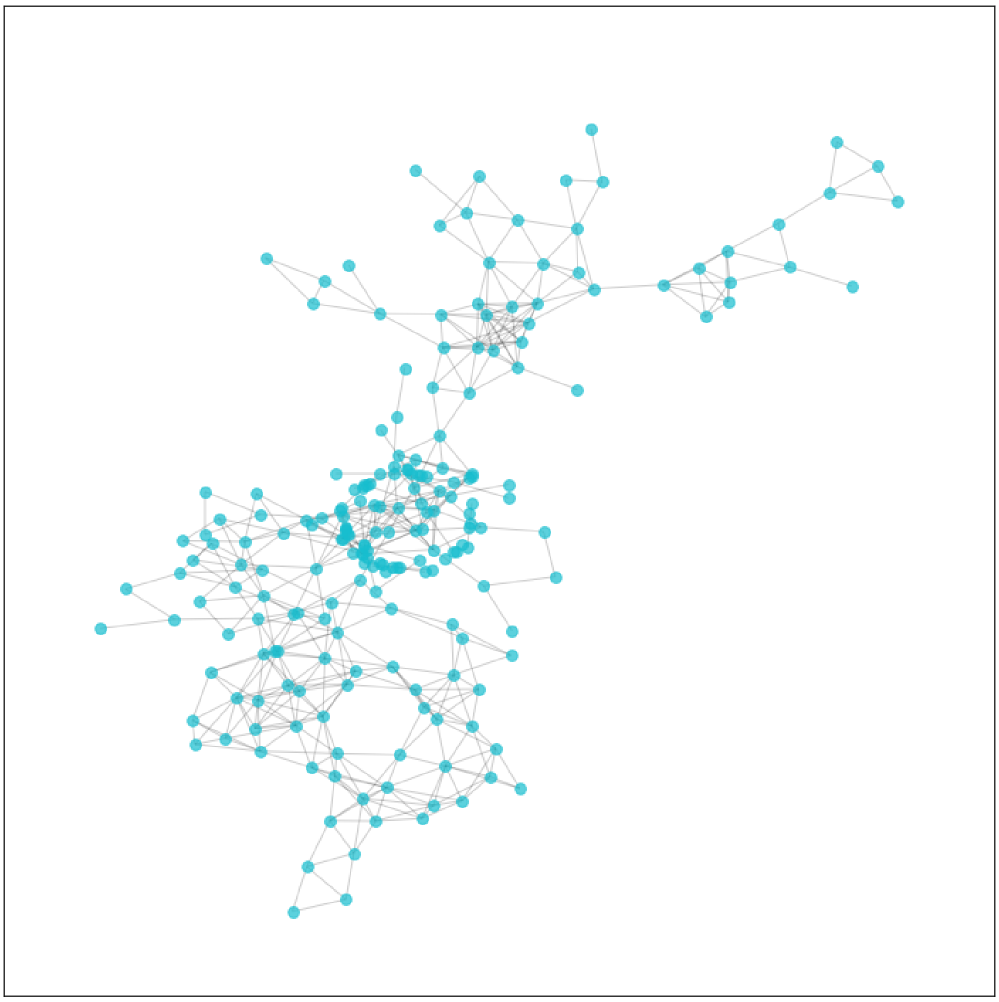


Figure 1: Graph learned on Seeds dataset

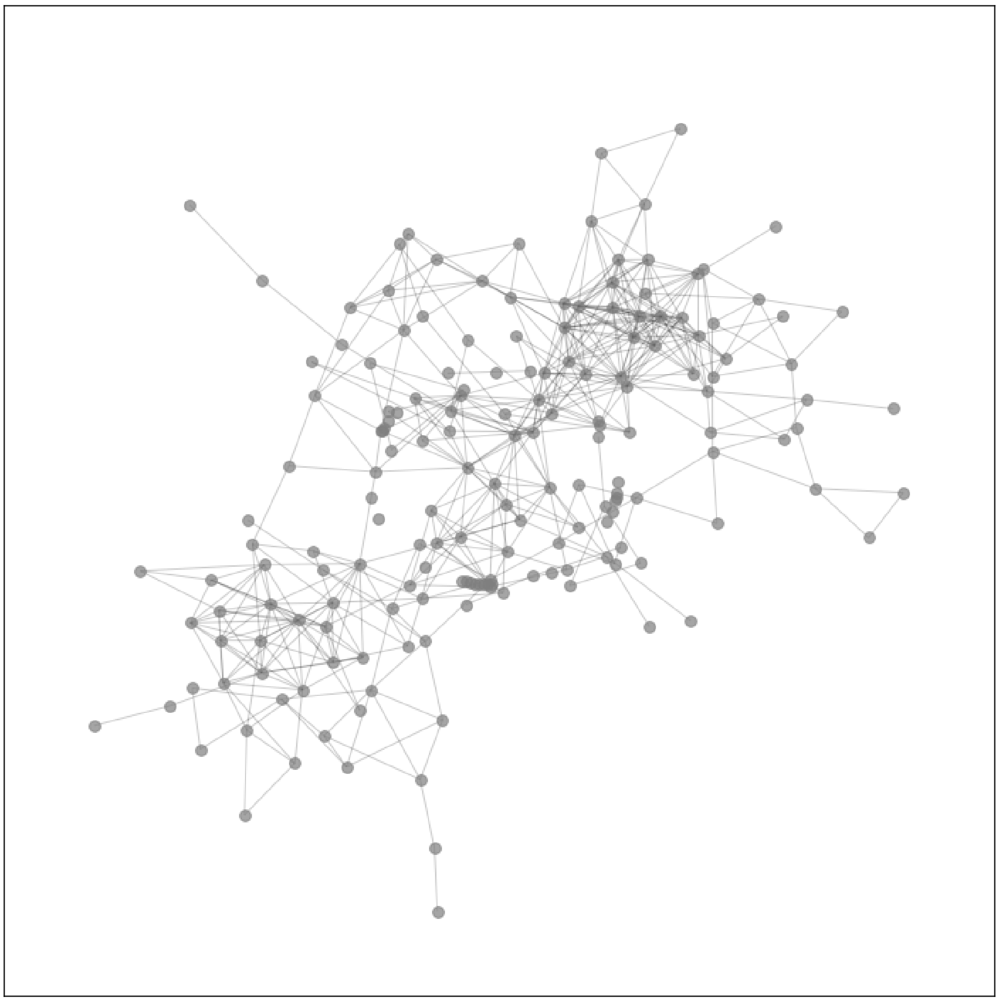


Figure 2: Graph learned on Wine clustering dataset

I also try with larger missing value percentages (10%), and perform missing value imputation using k-regression models. Calculating the Euclidean distance between generated versus original data in both kinds of imputation reveals reduction in distance by almost a factor of $\frac{1}{2}$. Here are the graphs generated in the two settings:

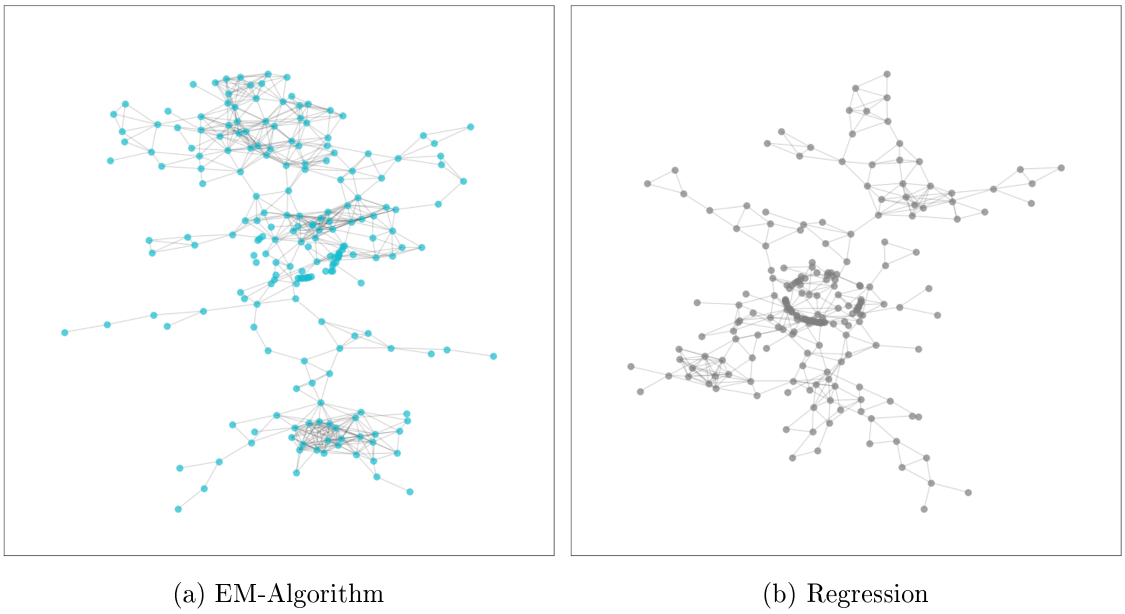

(a) EM-Algorithm (b) Regression

Figure 3: Seeds dataset

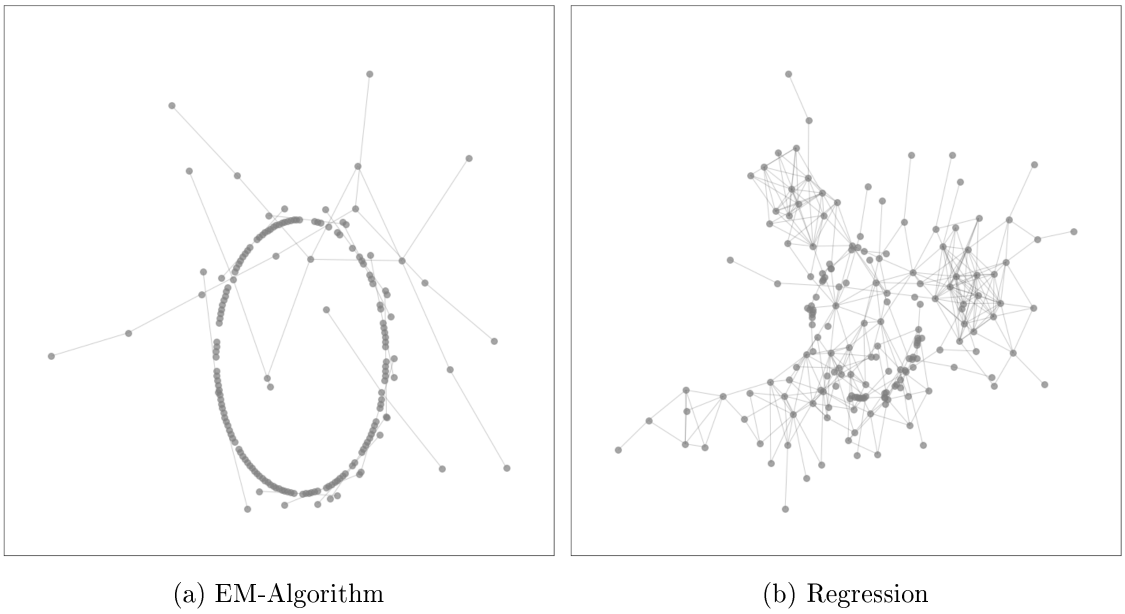

(a) EM-Algorithm (b) Regression

Figure 4: Wine clustering dataset

2 Q2

Since $y_i \in \{0, 1\}$, a solution that incorporates label information is as follows:
Choose a distance metric $D(x_1, x_2)$. Update distance metric to be $\tilde{D}(x_1, x_2) = D(x_1, x_2) + k(1\{y_1 = 1 - y_2\} - 1\{y_1 = y_2\})$ if both y_1 and y_2 are available, else just use $D(x_1, x_2)$ without the label information. Calculate second order statistics matrix and learn graph.

So, taking this approach, for experimentation I chose MNIST handwritten digits dataset [5].

The MNIST dataset is really big for the purposes ($\mathbf{X} \in \mathbb{R}^{60000\times784}$) of this question, so I select $[10, 50, 100, 500]$ images and project this smaller subset using PCA onto $\mathbb{R}^15$. Also, since the question uses the assumption of $y \in \{0, 1\}$, I make this selection such that all images are of class 0 or 1 (MNIST has 10 classes from 0 to 9).

Next, I randomly choose 40% of the labels to be ignored. And just construct the graph from the data (figure 5). I also learn a graph without using label information (figure 6). One thing to notice is that even though the graph structure isn't much different in the

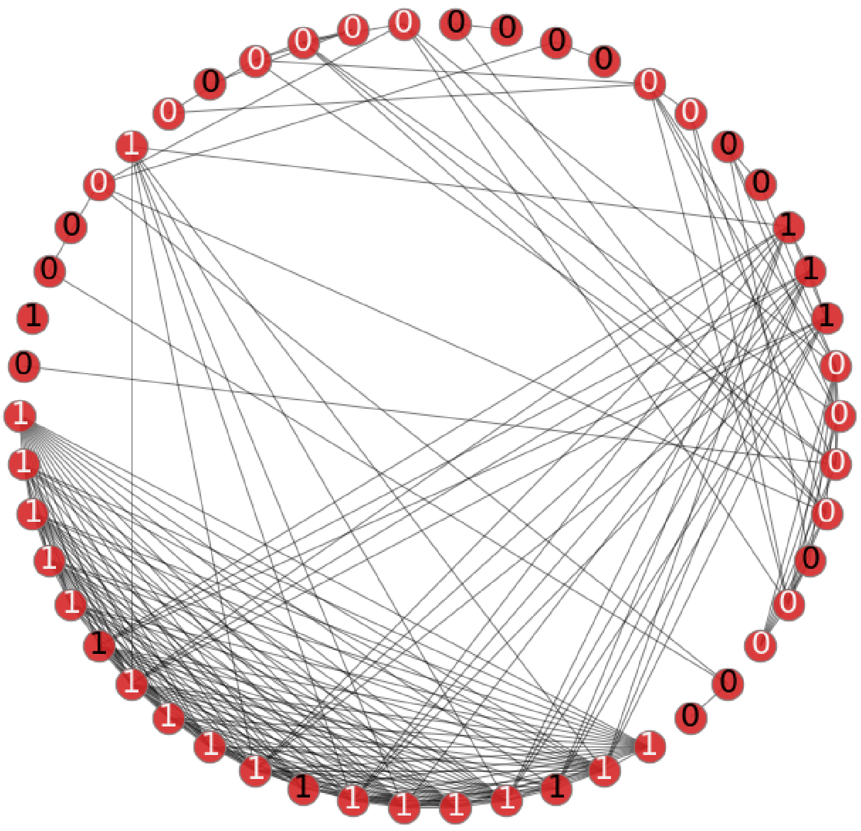


Figure 5: Graph with partially seen labels (white labels are observed, black are unobserved)

latter case there are two 0's connected to a 1 as seen in figure 7. so it can be said that the label information is being used, even if to a small extent.

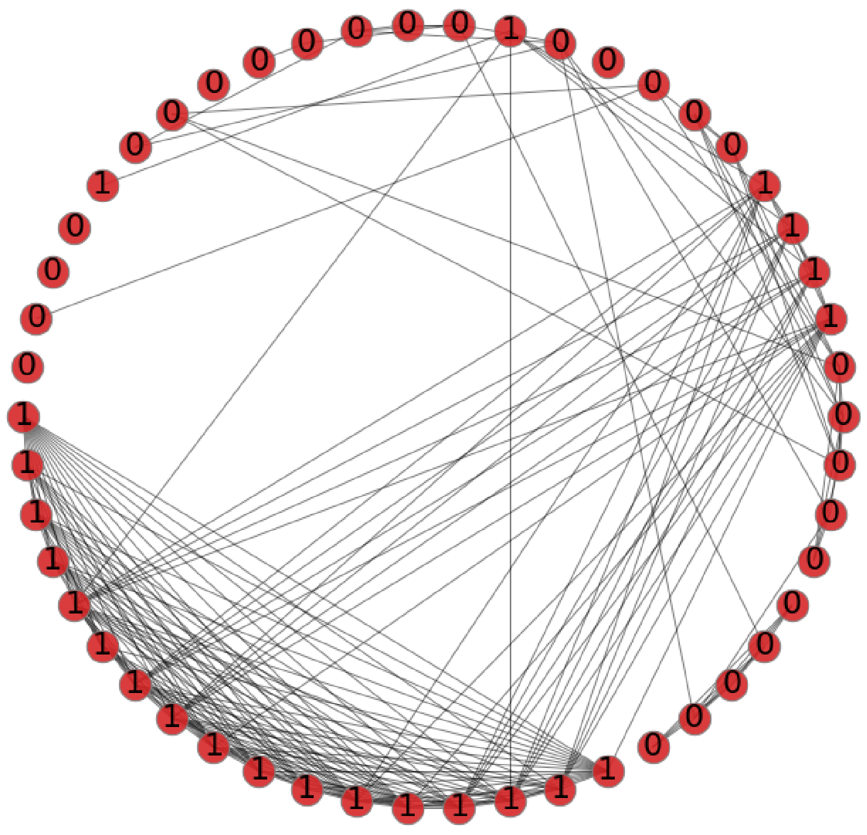


Figure 6: Graph with no seen labels

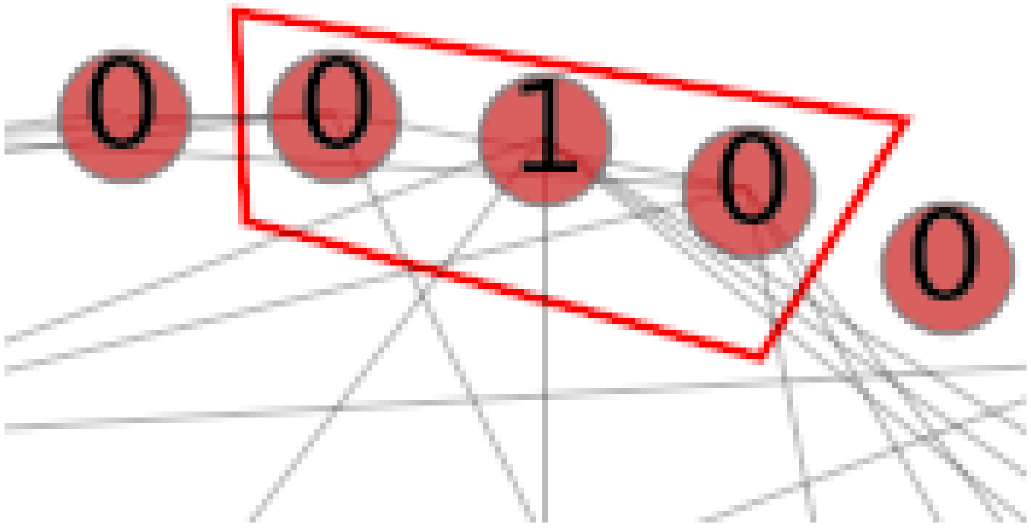


Figure 7: Graph with no seen labels

3 Q3

In this problem we have the data matrix $X \in \mathbb{R}^{n\times d}$ where both n and d are large, typically with $n \gg d$. The problem here is in calculating the second order statistics matrix. The computation cost will be of the order $O(n^2 d)$. Since we need a final adjacency and/or a weight matrix $\in \mathbb{R}^{n\times n}$ we cannot avoid the n^2 in the cost term, but we can try to reduce the d component.

Another problem is of fitting the big data matrix in memory to perform computations. For really large matrices, they need to be stored in hard drive which makes the computation difficult, unless one has access to powerful compute resources.

3.1 Developing the idea

One of the key things that makes what follows work is the assumption (which would generally be true) that the data is quite sparse. This means we can compress the information to a lower dimensional space.

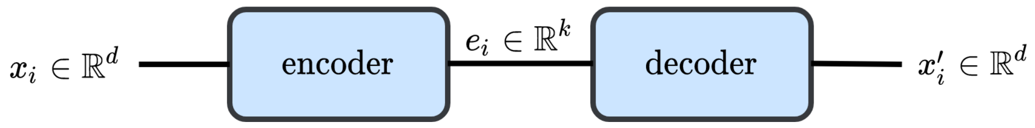
Now, there are several ways to go about this. One way could be to do PCA (or SVD) on the data $X \in \mathbb{R}^{n\times d}$ to get $X^{PCA} \in \mathbb{R}^{n\times k}$ for some $\mathbb{Z}^+ \ni k \ll d$. This new data can then be used to learn the graph. The computation complexity of PCA is $O(\min(d^3, n^3)$ which is $O(d^3)$ in our case. And with the second order statistics calculation the complexity becomes $O(d^3 + n^2 k)$.

Another problem remains however: the closed form PCA algorithm makes use of the full data matrix, which might not fit into the memory for really large n and d.

3.2 Better solution (?)

As mentioned in the hint, it would be good if we could use stochastic gradient descent based approaches. And the straight forward solution is to use the extension of PCA that makes use of stochastic gradient descent: a deep learning architecture called **AutoEncoder**.

An AutoEncoder consists of two neural networks: an *encoder* and a *decoder*.



The encoder takes in the data item $x_i \in \mathbb{R}^d$ and outputs an encoded embedding $e_i \in \mathbb{R}^k$ while the decoder takes this embedding and tries to reproduce the data from this compressed representation. This can then be trained end-to-end using stochastic gradient descent in minibatched fashion with mean squared error loss calculated on x and x'. The idea is that if the embedding is useful and retains information, then original data should be reproducible given the embeddings.

Once the training is completed, the decoder is discarded and just the encoder is used to generate embeddings from data. And once we have this, we can learn the weight matrix using a kernel that looks at pairs of data points at a time. For example a Gaussian Kernel can be used as shown in equation 1.

$$w_{ij} = \exp\left[-\frac{d(e_i, e_j)}{2\sigma^2}\right] \tag{1}$$

Note: an autoencoder without non-linearities learns the same embeddings as PCA.

3.3 Advantages

- Easily parallelizable.
- Reduces data dimension that mitigates "curse of dimensionality".
- Works on part of data that can be loaded into memory from hard drive in advance by efficient data loader making it faster after the first few data loads.
- Can be trained on part of the data. It is generally required to have more data than the number of parameters to avoid overfitting. Using a validation set to keep a check on the generalization error provides further improvement.

3.4 Disadvantages and challenges

- Fine tuning the parameter k is a challenge.
- Since this works like a look up (that is take a data point and generate its embedding), this can only be used with methods that generate w_{ij} by looking only at x_i and x_j. Algorithms that use something like k-nearest neighbor information are still a challenge.

3.5 Experimentation

I use the MNIST handwritten digit dataset [5] for this task. This consists of 60000 images, each of size 28×28, so the data matrix $\mathbf{X} \in \mathbb{R}^{60000\times 28\times 28}$ or equivalently $\mathbf{X} \in \mathbb{R}^{60000\times 784}$ after converting the image matrices into vectors. This data is still small, but it'll work for this purpose.

I choose two projection size 15 and 5 and test on both. Note this is a lot of compression and it is expected a lot of information will be lost.

The 60000 sized dataset is still too large for my machine, and the naive but unavoidable $\frac{n(n-1)}{2}$ calculations take a lot of time so I chose a still smaller subset of size 3000 to do my weight matrix calculations. The only way to go beyond this is to parallelize these calculations using GPU.

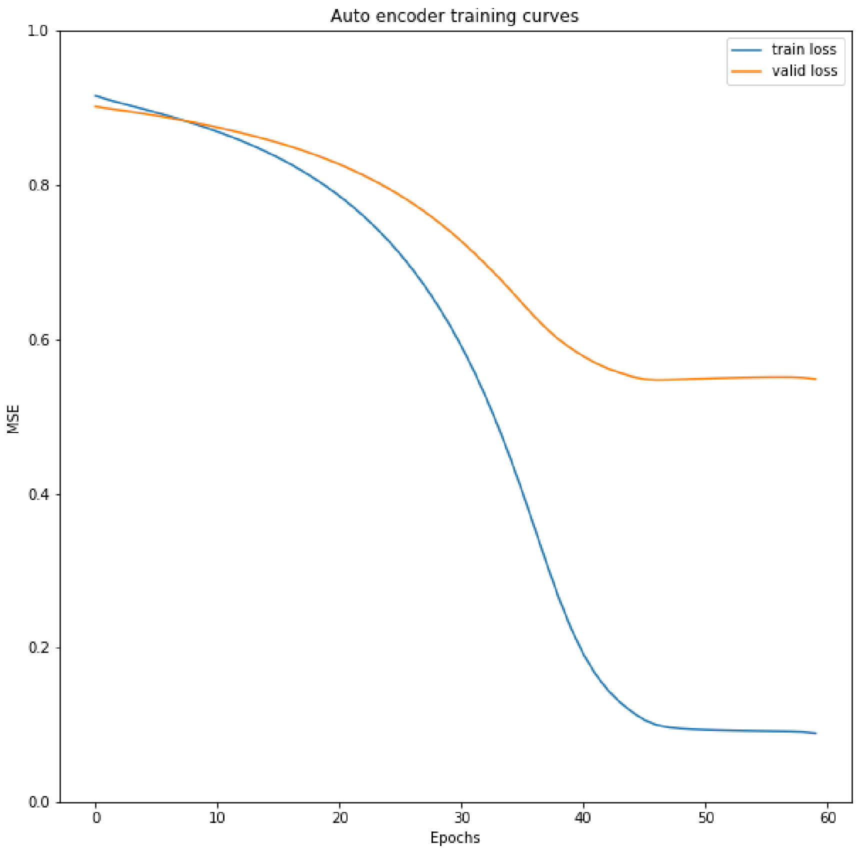


Figure 8: Projection size 15

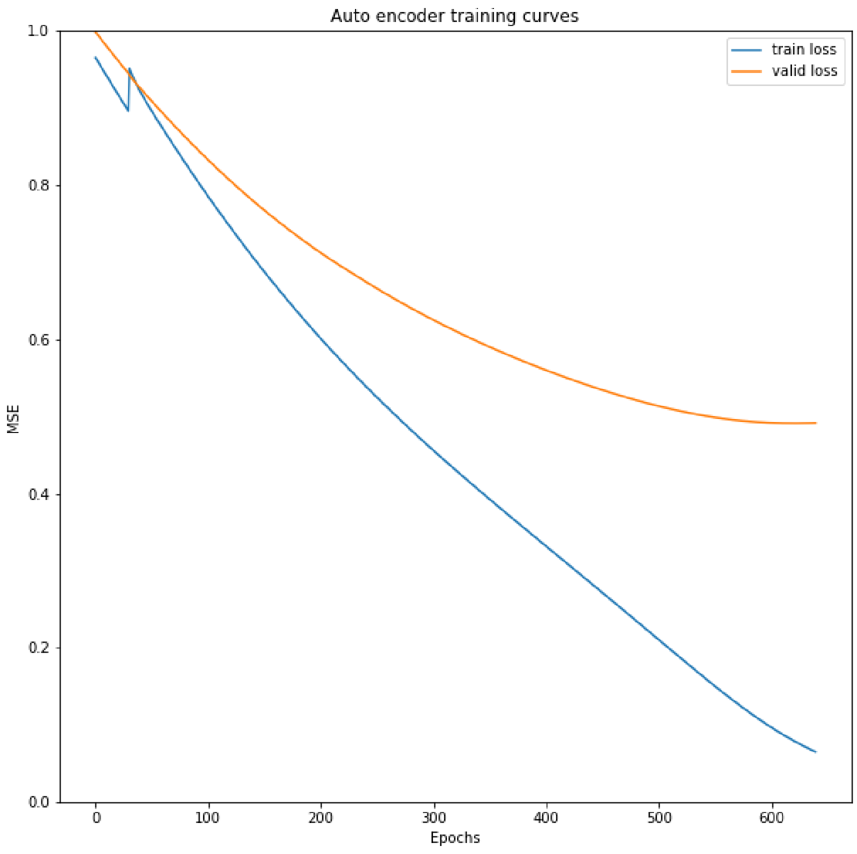


Figure 9: Projection size 5

4 Q4

The problem with hetergeneous data can be solved by embedding the data points into a real vector using something similar to the previous question. Another much simpler approach is to simply define a distance function that works with heterogeneous data to generate the second order statistics matrix.

4.1 Approach

The assumption here is that unlike the previous question, the data is not of high volume, but that the challenge is that graph learning frameworks usually work with numerical data.

So, we need a function that can calculate the elements of the second order statistics matrix. And this can be done by defining a distance function for various different kinds of data. Say we partion $\mathbf{X} \in \mathbb{R}^{n \times d_r} \times \mathbb{I}^{n \times d_i} \times \mathbb{C}^{n \times d_c}$ as $[\mathbf{X_R}, \mathbf{X_I}, \mathbf{X_C}]$. Then we can define a suitable distance function on each of the partition.

For example, say we use the Euclidean distance (equation 2) on both real and integer partitions.

$$D_{\text{eucl}}(x^i, x^j) = \left(\sum_{k=1}^{d_{\text{partition}}} (x_k^i - x_k^j)^2 \right)^{\frac{1}{2}} \tag{2}$$

And use the Hamming distance after one-hot encoding the categorical data (equation 3).

$$D_{\text{hamm}}(x^i, x^j) = \sum_{k=1}^{d_c} x^i \oplus x^j \tag{3}$$

With these two, the final distance can then be defined as a weighted sum as shown in equation 4.

$$\begin{aligned} N(x^i, x^j) =& W_r \cdot D_{\text{eucl}}(x^i_{1:d_r}, x^j_{1:d_r}) \\ &+ W_i \cdot_{\text{eucl}} (x^i_{d_r:d_r+d_i}, x^j_{d_r:d_r+d_i}) \\ &+ W_c \cdot D_{\text{hamm}}(x^i_{d_r+d_i:d_r+d_i+d_c}, x^j_{d_r+d_i:d_r+d_i+d_c}) \end{aligned}$$

$$D(x^i, x^j) = \frac{N(x^i, x^j)}{W_r + W_i + W_c} \tag{4}$$

4.2 Next steps

Once we have the second order matrix weights can be generated from it using kernel methods, linear embeddings, nearest neighbor methods, laplacian based methods for structural constraints, etc.

4.3 Advantages

- Works with heterogeneous data
- Little modification required in anything else

4.4 Disadvantages

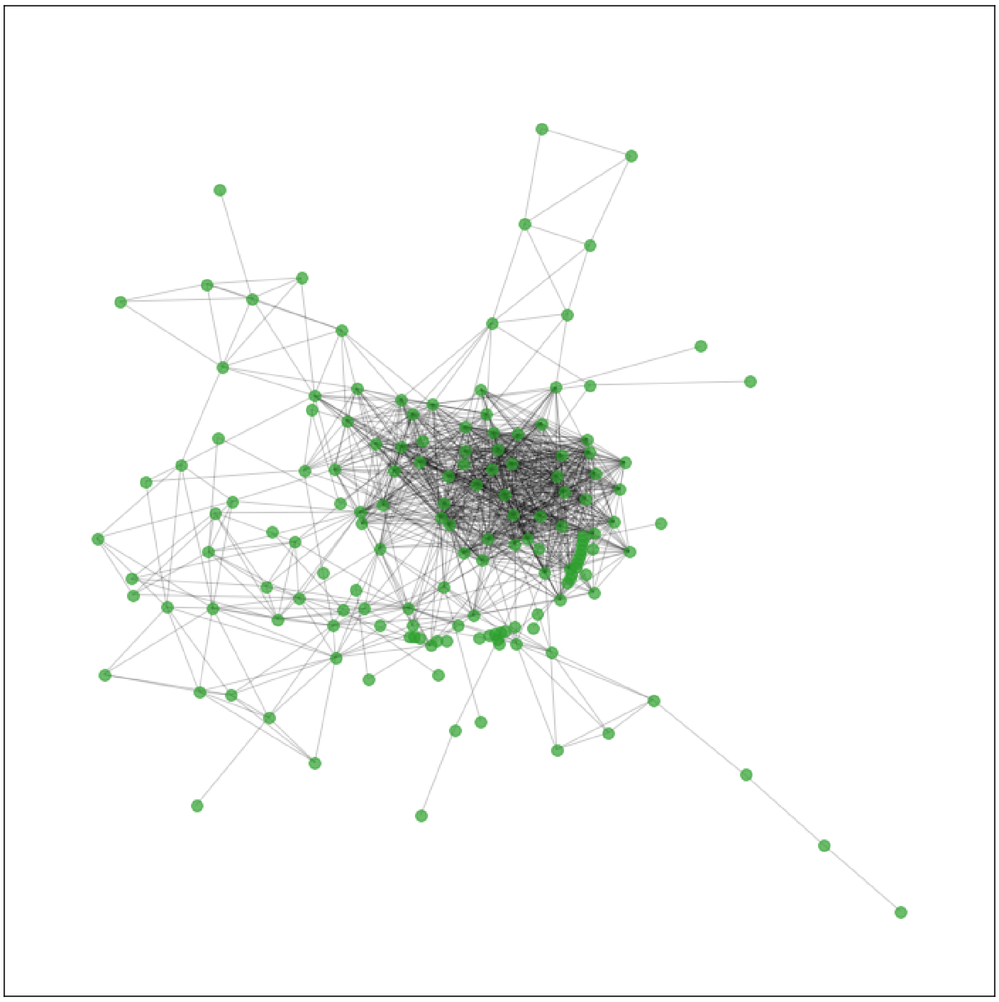
- Works when data is small. If data is large however, then one must consider embedding to lower dimension similar to previous question since autoencoders can also be used with categorical (after one-hot encoding) and integer data.

4.5 Other possible approaches

One can also consider using other data structures such as trees (and metrics they use, like gini index or mutual information) since they work well with heterogeneous data.

4.6 Experimentation

The dataset selected was hepatitis dataset [2, 6, 7] found here. It has 20 features, 14 of which were categorical, 4 integers and 2 are real valued. Using the above method, the following graph was generated



References

[1] Y. Dong and C.-Y. J. Peng, "Principled missing data methods for researchers," *Springer-Plus*, vol. 2, no. 1, pp. 1–17, 2013.

[2] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[3] M. Charytanowicz, J. Niewczas, P. Kulczycki, P. A. Kowalski, S. Łukasik, and S. Żak, "Complete gradient clustering algorithm for features analysis of x-ray images," in *Information technologies in biomedicine*, pp. 15–24, Springer, 2010.

[4] S. Aeberhard, D. Coomans, and O. De Vel, "Comparative analysis of statistical pattern recognition methods in high dimensional settings," *Pattern Recognition*, vol. 27, no. 8, pp. 1065–1077, 1994.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] B. Cestnik, I. Kononenko, and I. Bratko, "Assistant 86: A knowledge-elicitation tool for sophisticated users," in *Proceedings of the 2nd European conference on European working session on learning*, pp. 31–45, 1987.

[7] P. Diaconis and B. Efron, "Computer-intensive methods in statistics," *Scientific American*, vol. 248, no. 5, pp. 116–131, 1983.