ELL 888 Minor 1

Supervisors: Prof.Sandeep Kumar and Prof.Jayadeva

Report by: Hari Krishnan CK (2021EEY217583)

Code available at: https://github.com/harikuttan7136/ELL888-Minor-Graph-learning

Problem Statement 2:

Code available at: https://github.com/harikuttan7136/ELL888-Minor-Graph-learning

Consider a semi-supervised graph learning problem. Now with the data X sitting at n vertices we also have the label information for k < n vertices. Simply, we have $X \in R^{(n \times d)}$ with label information for k vertices $y \in \{0, 1\}^k$. Explain in detail how can we learn the graph matrix integrating the label information.

Introduction:

In a graph learning setup normally we don't use the label information available to us. Here a new approach to the Semi-Supervised graph learning problem is discussed. We can't use normal graph learning techniques like kNN graph learning,b-matching etc directly since these methods don't use graph labels in their objective function. We modified the kNN graph learning approach to accommodate the label information and account for the unlabelled samples.

APPROACH:

We assume that the neighbourhood of a sample in a particular class is of the same class. Based on this assumption we prevent the between-class edges for the labeled data. For the unlabelled data, we first train a nonlinear neural network classifier that predicts the class of the sample. The last layer of the neural network is a SoftMax layer which gives the probability of the sample to be in a particular class. To get the edges between the unlabelled and labeled data points we multiply the distance between the samples with the inverse of the probability predicted. i.e. Suppose an unlabelled data sample x_i has the probabilities $\{P_i\}_{i=1,2,\dots K}$ (K=total number of classes), then the distance between x_i and samples in labelled class is modified $d_{ij} * 1/P_k$ if x_j belongs to class k. The intuition is that if a particular unlabelled sample has a high probability of belonging to one class then the sample should have more neighbors from that class.

ALGORITHM:

Let n be the total number of data points with k being the number of labeled points.$X_l=\{x_i,y_i\}_{i=1,2,\dots k}$ be the set of labelled datapoints and $X_u= \{x_i\}_{i=k,k+1\dots\dots n}$ be the set of unlabelled datapoints .

Step 1: Normalize the data for zero mean and unit variance. Since kNN graph learning will be used it is better to scale all features.

Step 2: Build a neural network to classify samples in X_l .The output of the neural network gives the probability that a sample belongs to class i (0<=i<=K-1).The number of outputs of neural networks is equal to the number of classes in the dataset.

Step 3: Predict the probabilities for the unlabelled samples.

Step 4: Build a mask that prevents the samples in opposite classes from having an edge which is provided as input to kNN graph learning algorithm[2]. Here we assume that a sample in a particular class will be surrounded by samples of the same class.

Step 5: Update the distance between labelled and unlabelled samples

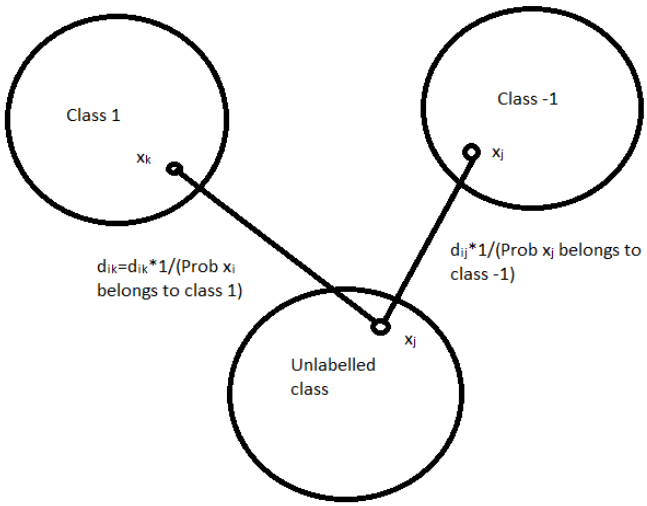
$d_{ij} = d_{ij} * 1/P_k$ if x_i belongs to unlabelled class x_j belongs to class k and vice-versa.

Step 6: Run the kNN graph learning algorithm to get the edge matrix E which find the k nearest neighbours for each node using the distance matrix D calculated in step 3.

Step 7: To find the weight matrix W, we use the below formula

$$W_{ij} = E_{ij} \cdot \exp\left(-d\left(x_i, x_j\right)^2 / 2\sigma^2\right)$$ [2]

Figure[1] shows the edge formation between labeled and unlabelled class. Note that the same sample can form an edge with -1 class and +1 class if the modified distance is still within k smallest values. This situation normally happens when the probabilities of two classes are equal. We will be able to learn a fully connected graph with this setup since the unlabelled points will act as a bridge between the two classes. These bridge points will be the points that the classifier is unable to ascertain with very high confidence as +1/-1 and hence connection will be formed with both the classes.



Figure[1]. The edges between labelled class and unlabelled class

EXPERIMENTS:

Experiment 1:

Dataset description: Banknote authentication dataset[1]. Data consists of 1348 instances with 4 numerical features and no missing values. The specific dataset is chosen because there are no categorical values and all features are continuous.

The classifier was trained using 1078 labelled samples and the probability values were predicted for 270 samples. The predicted probability values are then used to modify the distance with the X_l samples.

To demonstrate the effectiveness of the algorithm , a small sample of 15 samples was taken after training the neural network with 1078 samples. For the following example in Figure[2] only one sample is present in class 1, 9 samples in -1, and 5 samples in class 0(unlabelled class). In the figure . nodes with labels greater than 10 are unlabelled, node 6 is class 1 and the remaining are from class -1.

Explanation of figure[2]:

The probability that unlabelled samples belong to class 1:

Node 10: 0.99

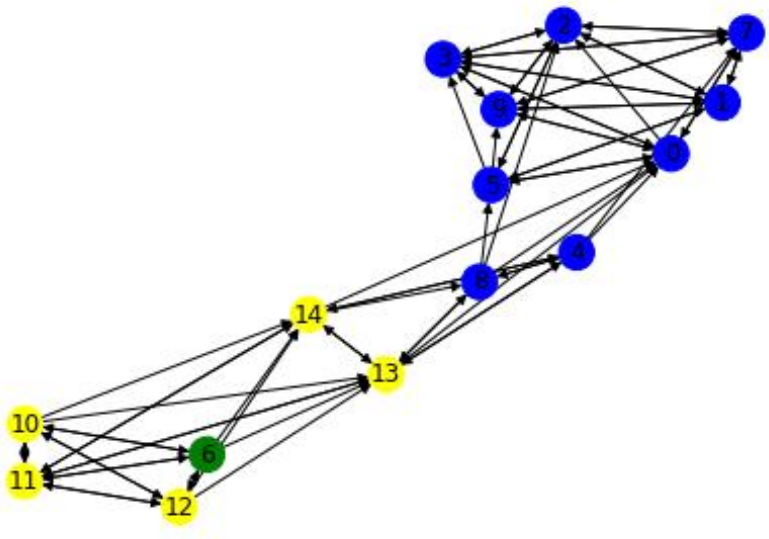
Node 11: 0.99

Node 12: 0.98

Node 13:0.009

Node 14:0.009

As it is evident from the graph, nodes 13 and 14 are connected to class -1 cluster (Nodes 0,1,2,3,4,5,7,8,9) and unlabelled nodes 11,12,13 and hence act as a bridge between class 1 and class -1. Nodes 10,11,12 are connected to node 6 since their probability of being in class 1 is very high nd hence are closer to Node 6.



Figure[2]. Graph formation. Nodes with labels greater than 10 are unlabelled, node 6 is class 1 and the remaining class -1. Yellow is unlabelled Blue is class 1 and green is class -1.

RESTRICTIONS:

The main restrictions of the above-proposed algorithm are:

1.) The algorithm uses kNN graph learning which could be unfeasible for large datsets.
2.) There is the possibility that the graph formed may be disconnected if the unlabelled class samples are far apart from both classes (If the probability is near 0.5 for all the samples), in that case, we will get three components with no edges between different classes and unlabelled class as samples will prefer to form edges with their own class.

CONCLUSION:

The idea of using a classifier to predict the probabilities which are further used to modify the distance seems to give good results as shown in the experiments section. The added advantage is that we will be able to get fully connected graphs for a sufficiently large k value. The ambiguous unlabelled samples according to the classifier can be connected to both classes as well as within themselves. Here in this new algorithm we have made use of the labelling of the samples.

REFERENCES:

[1].Bank note authentication dataset.

[2]. Lishan Qiao, Limei Zhang , Songcan Chen, Dinggang Shen Data driven graph construction and learning

Problem Statement 3:

Code available at: https://github.com/harikuttan7136/ELL888-Minor-Graph-learning

Consider a massive data scenario, where n and d both are very large and typically n >> d. Such that it is not possible to process the entire data every iteration. Explain how we can approach the graph learning problem under such a setting.

Introduction: In graph learning given a set of vertices V we have to find the edge set E or weight matrix W which connects the vertices. The problem statement here deals with the situation of having a very large number of samples in a high dimension (N>>d). The difficulty in using normal graph construction techniques like kNN graph,b-matching, etc here is because of the gigantic amount of memory needed to compute the distance matrix (N*N). Below is an algorithm to tackle the problem of having many samples.

Approach:

Instead of finding the distance between all the points in one iteration which needs huge memory requirements for large samples, the idea of using clustering before graph learning is explored.

We split the data into k clusters using Mini-batch K-means clustering which finds the cluster centroids using a gradient descent approach. The main benefit is since this algorithm uses only mini-batches instead of the whole data, it can be assumed that the mini chunk can be stored in the memory. Once we have the clusters, we first find the intra-cluster edge set (E) and distance matrix(D) for each cluster separately using some graph construction algorithms like kNN graph learning. Since we are only calculating the within-cluster distances the run time will be reduced significantly. We find the edges between the clusters(inter-cluster edges) by finding the nodes farthest from the cluster centroids and finding the edges and weights between the furthest nodes. The assumption here is that the connections between clusters will be only between the nodes at the periphery of the clusters and hence we can ignore the nodes other than the furthest nodes from the cluster centre for finding between cluster edges.

ALGORITHM:

Assume $\{x_i\}_{i=1,2,3....N}$ is the data where each x_i is a 1*d vector, N is the total number of nodes.

Step 1: Normalize the data to have zero mean and unit variance. Since we are using k-NN graph learning approach, normalization is necessary as a means to check that the pairwise distance doesn't explode

Step 2: Fix n_clusters which is the number of clusters and run Mini Batch K-means on the data to get the cluster centroids and cluster for each sample x_i. Let us denote the cluster label for sample x_i as y_i. y_i lies between {0,1,.....n_clusters-1}[1].

Step 3: Within-cluster graph- Run k-NN graph learning on each of these clusters separately to get the Edge set and Distance set.

Edge set for each cluster: $\{E_i\}_{i=1,2,\ldots n\_clusters-1}$ where each E_i is N_i*N_i matrix and N_i is the number of samples in i^{th} cluster.

Distance set for each cluster: $\{D_i\}_{i=1,2,\ldots n\_clusters-1}$ where each D_i is N_i*N_i matrix and N_i is the number of samples in i^{th} cluster.

Step 4: Combining $\{E_i\}_{i=1,2,\ldots n\_clusters-1}$ to form a single within-cluster Edge matrix E and combining $\{D_i\}_{i=1,2,\ldots n\_clusters-1}$ to form a single within-cluster Distance matrix D.

E and D are diagonal matrices with size N*N.

$$E=\begin{pmatrix} E1 & 0 & \cdots & 0 \\ 0 & E2 & \ldots & 0 \\ \vdots & : & \ddots & \vdots \\ 0 & 0 & \cdots & Ez \end{pmatrix}$$

$$D=\begin{pmatrix} D1 & 0 & \cdots & 0 \\ 0 & D2 & \ldots & 0 \\ \vdots & : & \ddots & \vdots \\ 0 & 0 & \cdots & Dz \end{pmatrix}$$

z =n_clusters-1

Step 4: Between cluster graph- For each cluster find k' nodes furthest from the cluster centroid.

Let us denote $X_b=\{x_i'\}$ as the set of samples furthest from their respective cluster centroids. Size of X_b is $(k'*n\_clusters)Xd$ since for each cluster we have the furthest k' samples.

Step 5: Run k-NN graph learning on X_b to get between cluster Edge set E_b and Distance matrix D_b (Here b Is used to denote between cluster). Split E_b and D_b to different matrices E_{ij} where E_{ij} is the edge set between cluster i and cluster j.

Step 6: Update E and D matrices with between cluster edges and distances. The matrices after updating will be

$$E=\begin{pmatrix} E1 & E12 & \cdots & E1z \\ E21 & E2 & \ldots & E2z \\ \vdots & : & \ddots & \vdots \\ Ez1 & Ez2 & \cdots & Ez \end{pmatrix}$$

$$D=\begin{pmatrix} D1 & D12 & \cdots & D1z \\ D21 & D2 & \ldots & D2z \\ \vdots & : & \ddots & \vdots \\ Dz1 & Dz2 & \cdots & Dz \end{pmatrix}$$

z =n_clusters-1

E12 is the edge matrix between cluster 1 and cluster 2

Step 7: Prune the degree of the nodes with degree more than k to get the final edge set E and distance matrix D which can be used further to get the weights.

Step 8: Run the kNN graph learning algorithm to get the edge matrix E which find the k nearest neighbours for each node using the distance matrix D calculated in step 3.

Step 9: To find the weight matrix W, we use the below formula

$$W_{ij} = E_{ij} \cdot \exp\left(-d\left(x_i, x_j\right)^2 / 2\sigma^2\right)$$ [5]

EXPERIMENTS:

Experiment 1:

Dataset description-HTRU_2 dataset[2]. It contains 17989 data samples with 8 attributes. It is a collection of pulsar candidates (pulsar is a type of star which is of scientific interest) collected during the High Time Resolution Universe Survey[2].

Sample size 1= (12528, 8)
Chose n_clusters =9 for running Mini-Batch K-means so that the total samples will be sufficiently distributed to 9 clusters.

Total number of samples in each cluster after Mini-batch K-means :
[679, 1544, 572, 1368, 396, 2694, 1339, 1203, 2733]

Total time taken for the algorithm to run- 19.45s
Total time taken for the k-NN graph learning algorithm to run without clustering – 31.34s

Time taken for the proposed algorithm is significantly less than the direct approach where time increases exponentially with the increase in the total number of samples whereas for the proposed algorithm increases exponentially on the highest number of samples in each cluster. This is evident from the graph given below.

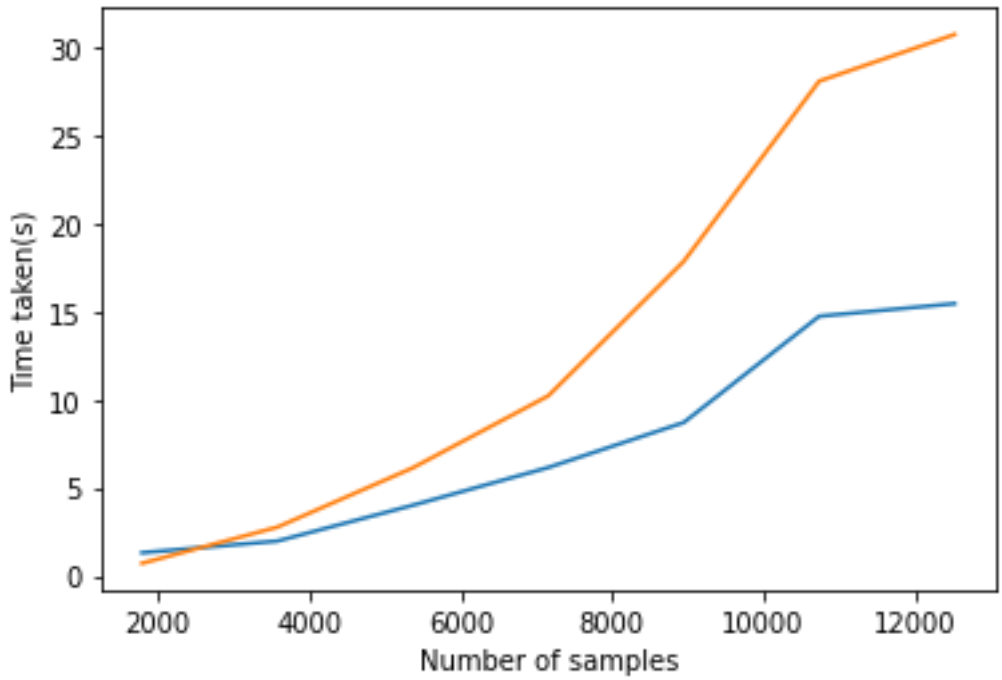


Figure [1]. Time taken vs Number of samples for the the two algorithms. Orange is for direct kmeans and blue is for the algorithm proposed.

Experiment 2:

Dataset Description: Spambase dataset[3] .

Sample size taken for running algorithm: 3368*57

n_clusters=3

Total samples in each cluster=[2478,617,273]

Time taken for clustering based algorithm:2.038

Time taken for naïve k-NN graph learning: 2.05

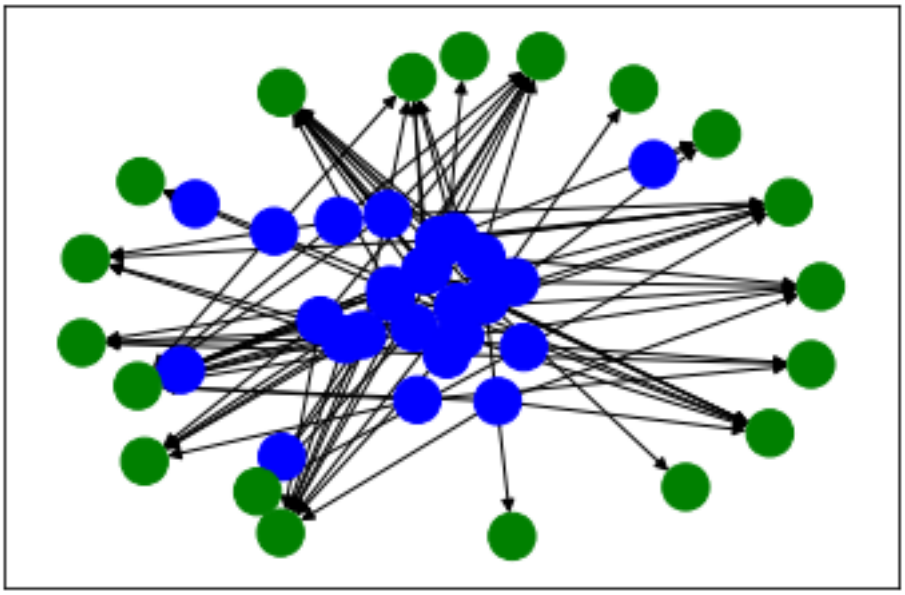


Figure [2]. The figure shows that the inter-cluster edges are formed between clusters. Blue is cluster 0 and green is cluster 1. Note that edges between cluster are not shown and the graph is not drawn with positions.

RESTRICTIONS:

The main restrictions or problems with the proposed algorithm is :

1.) If the sample size is low, then Mini-batch K-means will assign all samples to one cluster with only very few samples in other clusters.
2.) In the algorithm and implementation, combined edge matrix E was made which could lead to space problems but an easy fix is modify the algorithm so we store edges in form of adjacency list using hashmap instead of edge matrix (so only non-zero edges are stored).

ALTERNATE IDEAS:

Alternate approaches which could be useful or could which lead to better results:

1.) Use Autoencoders to get the representation in lower dimension if d is very high.
2.) Use classifiably measure[4] as loss function to increase the distance between the samples so there won't be any between-cluster edges and we get k component graph(k is the number of clusters).

CONCLUSION:

The proposed approach of finding the cluster centroids using Mini-Kmeans algorithm which uses stochastic gradient descent and then finding the within-cluster edges and between-cluster edges is much faster than the naïve approach of k-NN graph learning. We are also able to learn between cluster edges without having to calculate the distance between points for the entire dataset.

REFERENCES:

[1]. Javier Bejar,K-means vs Mini Batch K-means

[2].HTRU_2 dataset

[3].Spambase dataset

[4].Hiulin Xiong,M.N.S Swamy Optimising the kernel in empiracl feature space

[5]. Lishan Qiao, Limei Zhang , Songcan Chen, Dinggang Shen Data driven graph construction and learning

Problem Statement 4:

Code available at: https://github.com/harikuttan7136/ELL888-Minor-Graph-learning

Consider a heterogeneous data scenario, where dataset X associated with vertices may belong to set of real R, set of integers I, and categorical C = {c1, c2, . . . , ck}. More concretely, X ∈ R (n×dr) × I (n×di) × C (n×dc) and d = dr + di + dc. For example x1 = [x11, x12, . . . , x1dr ∈ R, x1dr+1, x1dr+2, . . . , x1dr+di ∈ I, x1dr+di+1, x1dr+di+2, . . . , x1dr+di+dc ∈ C]. Explain in detail how can we learn graph matrix with heterogenous data setting. Put a descriptive detail

INTRODUCTION:

In real-life datasets, we may have the situation of having different types of features like categorical, integer, and real continuous values. The way we treat these different features or columns matters a lot while doing graph learning since we have to find the distance between the nodes which have these different features. This is a big research area in most machine learning problems and in this section, we will see how to tackle this problem from a graph learning perspective.

APPROACH:

We consider kNN graph learning to illustrate the idea presented here. In kNN graph learning it is important that we find the pairwise distances between the nodes which are then used to find the k nearest neighbors for a particular node and then the graph is constructed. Since we have different types of features we consider each type of feature differently. For the continuous real-valued features we calculate the Euclidean distance between them, for integer-valued features we calculate the Minkowski distance and for categorical values, we use the Hamming distance between the values. So we split the features into three subsets i.e for each node feature vector is split into {R,I,C} where R is for real-valued features, I is for integer-valued features and C is for categorical values. Once it is split into three subsets we calculate the distance between a pair of nodes using the distance matrix corresponding to a different type of feature. Finally, the total distance between two nodes is $d_R+d_I+d_c$.

Euclidean distance :

$$d(p,q) = \sqrt{(p_1-q_1)^2 + (p_2-q_2)^2 + \cdots + (p_i-q_i)^2 + \cdots + (p_n-q_n)^2}.$$

.[1]

Here $\{p_1,p_2,\ldots\ldots p_n\}$ are the set of real valued continuous features for one node and $\{q_1,q_2,\ldots\ldots q_n\}$ are the corresponding features for the second node.

Minkowski distance:

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}.$$

.[2]

Here X is set of I i.e integer valued features for one node and Y is set of I for the second node. In this paper we have used p=1

Hamming distance:

$\sum_{k=1}^{d} 1(x_{ik} \neq x_{jk})$
where $1(\cdot) \rightarrow \{0, 1\}$ is an indicator function. [3]

It gives in how many features the categorical values are same or different.

ALGORITHM:

Let x_i is d dimensional, therefore X is (m,d) dimensional matrix where m is the total number of samples. Out of d features, d_r is the dimension of real-valued continuous features, d_i is integer-valued features and d_c is the dimension of categorical valued features.

$d=d_r+d_i+d_c$

Step 1: Split the columns in X to integer, categorical and real-valued features. We decide a feature is categorical if the number of unique values in the data for that column is less than or equal to T(hyperparameter) and then the feature is assigned to categorical set C. For the integer case we check to see if the column has any decimal point or long values in them, if not it is assigned to integer feature set I else to R.

Step 2: Normalize the real-valued and integer-valued features with min-max normalization. Normalizing categorical features are not needed since we will be calculating the hamming distance.

Step 3: Find the pairwise distance matrix D using the Euclidean distance for the feature set R, Minkowski distance for the set I, and Hamming distance for set C.

Step 4: Run the kNN graph learning algorithm to get the edge matrix E which find the k nearest neighbors for each node using the distance matrix D calculated in step 3.

Step 5: To find the weight matrix W, we use the below formula

$$W_{ij} = E_{ij} \cdot \exp\left(-d\left(x_i, x_j\right)^2 / 2\sigma^2\right)$$ [3]

EXPERIMENTS:

Experiment 1:

Dataset Description: Statlog(Australian credit approval dataset)[4]. The dataset consists of 690 instances with 14 features. There are

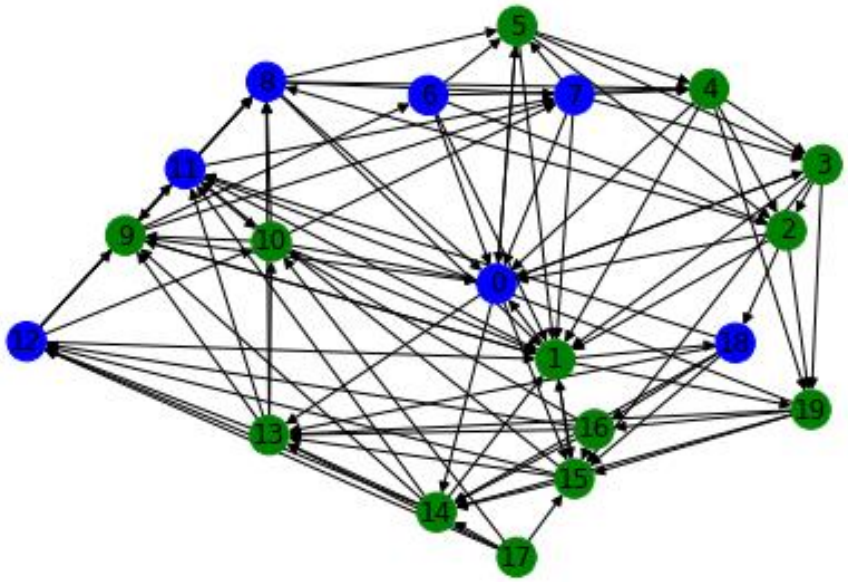
A1: 0,1 CATEGORICAL (formerly: a,b)
A2: continuous.
A3: continuous.
A4: 1,2,3 CATEGORICAL (formerly: p,g,gg)
A5: 1, 2,3,4,5, 6,7,8,9,10,11,12,13,14 CATEGORICAL (formerly: ff,d,i,k,j,aa,m,c,w, e, q, r,cc, x)
A6: 1, 2,3, 4,5,6,7,8,9 CATEGORICAL (formerly: ff,dd,j,bb,v,n,o,h,z)

A7: continuous.
A8: 1, 0 CATEGORICAL (formerly: t, f)
A9: 1, 0 CATEGORICAL (formerly: t, f)
A10: continuous.
A11: 1, 0 CATEGORICAL (formerly t, f)
A12: 1, 2, 3 CATEGORICAL (formerly: s, g, p)
A13: continuous.
A14: continuous.
A15: 1,2 class attribute (formerly: +,-)

The categorical columns returned by the algorithm is [1,4,8,9,11,12]. The set I returned are [5,6,10,13,14] and the real valued set [2,3,7].

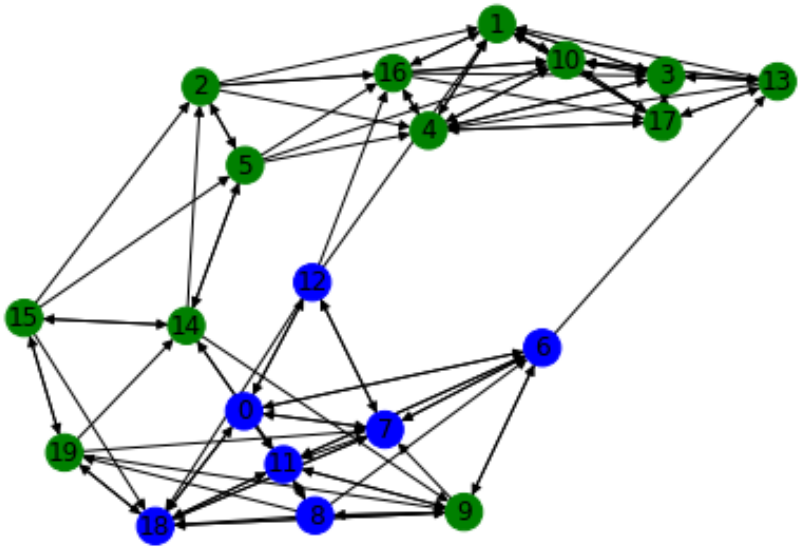
The average distance between the nodes from the proposed algorithm is 0.1621 and while calculating the Euclidean distance between all the features is 1.393.

For comparison purpose two kNN graphs were made, first using the above proposed algorithm and second one using the Euclidean distance for all the features.



Figure[1]. Graph learned using the proposed algorithm which uses different distance matrices for different types of features. Different node colours indicate nodes belong to different classes.

The graph learned using the proposed algorithm is much more well connected than the one learned using Euclidean distance for all features. Also Figure[2] shows that it may be lead to wrong graph structure since node 9 which belongs to class 1 is totally separated from the class 1 nodes and connection to class -1 nodes are made due to the wrong distance values learned. But from Figure[1], the same node 9 is able to form edges with the same class nodes 10 and others. This shows that the new proposed distance metric is better in both connectivity and the significance of the learned graph.



Figure[2]. Graph learned using Euclidean distances for all the features. Different node colours indicate different class.

OTHER IDEAS:

An alternative approach could be to convert the data to a lower-dimensional feature space using dimensionality reduction techniques like PCA,LDA etc and then use the continuous values in the lower dimension to get the distance measure.

CONCLUSION:

Here in the experiment, we were able to show that using different distance metrics for different types of features i.e using Hamming distance for categorical features, Minkowski distance for integer-valued features, and Euclidean distance for real-valued features is better than using Euclidean distances for all the features. We were able to demonstrate that the average distance will be lower in the graph made using the proposed algorithm and will lead to better connectivity in the graphs learned. Also, the graph learned had better significance in the proposed algorithm since in the case where we use Euclidean distance for all features distances might be wrong and could lead to wrong graph structures as shown in the experiment section.

REFERENCES:

[1].Euclidean distance.

[2]. Minkowski distance

[3]. Lishan Qiao, Limei Zhang , Songcan Chen, Dinggang Shen Data driven graph construction and learning

Problem Statement 1:

Code available at: https://github.com/harikuttan7136/ELL888-Minor-Graph-learning

Graph learning with missing data: Consider a dataset some parts of the data are missing at random. Explain in detail how can we learn graph matrix with missing data. Put a descriptive detail.

INTRODUCTION:

In real-life datasets, missing value is a recurring problem and there is a whole research field dedicated to solving this problem of dealing with missing data. Here in this section we investigate how to deal with missing data problem from the graph learning perspective. There is a lot literature around dealing with missing data including mean imputation, mode imputation, Gaussian Mixture Model imputation, etc. Here we consider MCAR (Missing Completely At Random) missing case i.e the data is missing independent of the observed and unobserved data (No difference between participants with missing data and complete data).

APPROACH:

Let X be the total data X is (M,d) shaped matrix with missing data. Let X_c be the complete dataset with no missing values i.e no value is missing in X_c . X_m is the dataset with at least one feature missing in the observation.

$X=X_m+X_m$.

We assume that the data is coming from a Unimodal normal distribution. We find the mean and covariance matrix of the normal distribution using the complete data X_c by the MLE(Maximum Likelihood Estimation). We know that the mean and covariance matrix by MLE is sample mean and sample covariance matrix i.e.

We already know $\hat{\mu} = \overline{x}$, so we use that as the value for μ in the formula for $\hat{\sigma}$. We get the maximum likelihood estimates

$$\begin{aligned} \hat{\mu} &= \overline{x} && = \text{the mean of the data} \\ \hat{\sigma}^2 &= \sum_{i=1}^{n} \frac{1}{n}(x_i - \hat{\mu})^2 = \sum_{i=1}^{n} \frac{1}{n}(x_i - \overline{x})^2 && = \text{the variance of the data.} \end{aligned}$$

[1]

So we find mean matrix μ (1Xd) and covariance matrix ∑ (dXd) using the complete data.

Suppose a random variable X can be split to $X^{(1)}$ (missing values) and $X^{(2)}$ (observed values), then

$$X = \begin{bmatrix} X^{(1)} \\ X^{(2)} \end{bmatrix}, \quad \boldsymbol{\mu} = \begin{bmatrix} \boldsymbol{\mu}^{(1)} \\ \boldsymbol{\mu}^{(2)} \end{bmatrix}, \quad \boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}$$ [2]

$$\boldsymbol{\mu}_1' = \boldsymbol{\mu}^{(1)} + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}^{(2)} - \boldsymbol{\mu}^{(2)})$$ [2]

$$\boldsymbol{\Sigma}_{11}' = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}$$ [2]

Where μ_1' and $\sum_{11}'$ are the conditional distribution of $X^{(1)}$ given $X^{(2)}=x^{(2)}$.

Then we find the pairwise expected square distances between the nodes using the formula :

$$\mathrm{E}\left[\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2\right] = \|\boldsymbol{x}_i' - \boldsymbol{x}_j'\|^2 + s_i^2 + s_j^2, \quad \text{where} \quad s_i^2 = \sum_{l \in M_i} \sigma_{i,l}^2$$ [2]

Where x_i' and x_j' are two samples with missing values and s_i^2 is the sum of the trace of matrix $\sum_{11}'$.

Derivation of the above formulae is elaborated in references [2] and steps to finding the same will be discussed in the algorithm section.

ALGORITHM:

Let X be the total data X is (M,d) shaped matrix with missing data. Let X_c be the complete dataset with no missing values i.e no value is missing in X_c . X_m is the dataset with at least one feature missing in the observation.

$X=X_m+X_m$.

Step 1: Split the data into complete dataset X_c and missing dataset X_m by seeing if there is any missing feature in each observation.

Step 2: Find the sample mean matrix μ (1Xd) and sample covariance matrix ∑ (dXd) using the complete data.

Step 3: For each sample x_i in missing data X_m repeat steps 4 to 7.

Step 4:split x_i to $[x^{(1)}_i, x^{(2)}_i]$, where $x^{(1)}_l$ are the missing values and are to be estimated

Step 5: Find the conditional mean and conditional covariance matrix given by

$$\boldsymbol{\mu}_1' = \boldsymbol{\mu}^{(1)} + \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}(\boldsymbol{x}^{(2)} - \boldsymbol{\mu}^{(2)})$$ [2]

$$\boldsymbol{\Sigma}_{11}' = \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12}\boldsymbol{\Sigma}_{22}^{-1}\boldsymbol{\Sigma}_{21}$$ [2]

Step 6: Find the value of s_i^2 using the formula given below(sum of the trace of conditional covariance matrix).

$$s_i^2 = \sum_{l=1}^{|M_i|} (\boldsymbol{\Sigma}_{11}')_{ll}$$ [2]

Step 7: Update $x^{(2)}_l$ i.e the missing value in x_i with μ_1' and store s_i^2 for each sample in X_m.

Step 8: Calculate the distance matrix D in X. If x_i does not belong to X_m then $s_i^2=0$ else use s_i^2 calculated for each sample in X_m.

Step 9: Run the kNN graph learning algorithm to get the edge matrix E which find the k nearest neighbors for each node using the distance matrix D calculated in step 9.

Step 10: To find the weight matrix W, we use the below formula

$$W_{ij} = E_{ij} \cdot \exp\left(-d\left(x_i, x_j\right)^2 / 2\sigma^2\right)$$ [3]

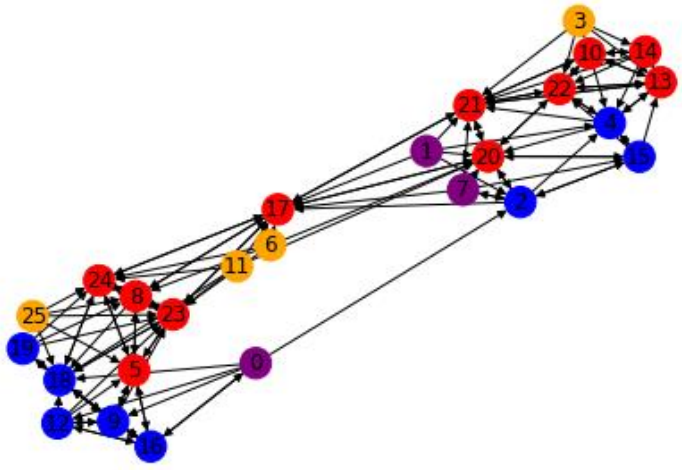
EXPERIMENT:

Dataset Description:

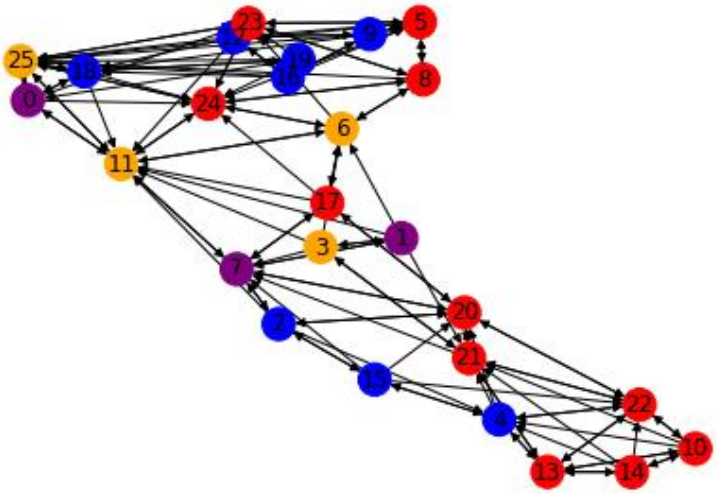
Banknote authentication dataset[4]. Data consists of 1348 instances with 4 numerical features and no missing values. The specific dataset is chosen because there are no categorical values and all features are continuous. For this example, we manually create a random mask which is used to create the missing values in the dataset. Two-thirds of the data is complete and one-third has missing values (one or more than one).

The average distance obtained for the proposed algorithm is more than the average distance obtained by just mean imputation.

Figure[1] below shows the graph constructed using the proposed algorithm and Figure[2] shows a graph constructed on same data using mean imputation on features. By observing the figures it is clear that the orange nodes are closer to red nodes and purple nodes closer to blue nodes in Figure[1] than in Figure[2]. Figure[2] is more mixed and there is no clear pattern.



Figure[1]. Graph constructed using the proposed algorithm. Red nodes are complete and belongs to class1 and orange are nodes with missing values in class1. Blue nodes are complete and belongs to class -1 and purple nodes are nodes with missing values in class -1



Figure[2]. Graph constructed with just feature-wise mean imputation. Red nodes are complete and belongs to class1 and orange are nodes with missing values in class1. Blue nodes are complete and belongs to class -1 and purple nodes are nodes with missing values in class -1

RESTRICTIONS:

1.) Here we have assumed that the data is Unimodal normally distributed which may not be the case. A better approach could have been considering GMM (Gaussian Mixture Model) and using EMM to get the mean and covariance matrix of the complete dataset.
2.) Although it is observed that the missing value nodes were closer to original class nodes without missing values, it couldn't be said with certainty that the graph constructed is much better than by just mean imputation.

CONCLUSION:

The proposed algorithm of using expected squared distances works and gives better results than just using the mean imputation technique but the effectiveness of the algorithm is limited and could have been improved by using GMM and EM algorithm instead of assuming unimodal normal distribution.

REFERENCES:

[1]. Jeremy Orloff and Jonathan Bloom Maximum Likelihood Estimates Class 10, 18.05

[2]. Emil Eirola , Gauthier Doquire , Michel Verleysen , Amaury Lendasse Distance estimation in numerical data sets with missing values

[3]. Lishan Qiao, Limei Zhang , Songcan Chen, Dinggang Shen Data driven graph construction and learning

[4].Bank note authentication dataset.

[5]. Emil Eirola, Amaury Lendasse, Vincent Vandewalle, Christophe Biernacki Mixture of Gaussians for Distance Estimation with Missing Data