Nonlinear Dimensionality Reduction Algorithms "Comparison of LLE and Modified LLE"

Erfan Etesami
School of Engineering (STI)
École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
Email: erfan.etesami@epfl.ch

Hugo Miranda Queiros
School of Engineering (STI)
École Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
Email: hugo.mirandaqueiros@epfl.ch

***Abstract*—Dimensionality reduction is a crucial step used extensively prior to applying machine learning techniques on large multivariate datasets. These algorithms are devised to discover the compact and yet necessary representation of high-dimensional data-i.e., with a significant number of features. Locally linear embedding (LLE) and its modified version (MLLE) are among dimensionality reduction techniques that are meant to learn a compact structure of nonlinear manifolds. These two algorithms are applied to a particular high-dimensional dataset prior to performing a binary classification task by the nonlinear support vector machine (SVM). The results indicate that using LLE and MLLE brings about a significant decrease in computation time as well as a better performance for LLE and equal for MLLE in terms of F1 and accuracy scores over practicing with the raw data. Moreover, LLE exhibits a superior classification behavior compared to MLLE and raw data in this dataset, as suggested by the attributed confusion matrices.**

I. Introduction

In many real-world applications, one often has to analyze datasets containing significant number of features for each sample, e.g. sets of images or text documents. Any attempt of learning a model for these high-dimensional datasets requires large computational time. Moreover, most of the time, these features are highly correlated and thus redundant.

Dimensionality reduction techniques are out there to address these issues. More concretely, these unsupervised algorithms try to squeeze necessary and relevant information out of the data while satisfying the need to learn an accurate model. Thereby, by realizing principal explanatory variables, dimensionality reduction methods avoid curse of dimensionality since they reduce the computational cost as well as the storage space. Furthermore, eliminating irrelevant features brings about a predictive model with higher accuracy.

Locally linear embedding (LLE) is a neighborhood-preserving technique meant to compute low-dimensional embedding of nonlinear high-dimensional manifolds [1]. More specifically, instead of computing pairwise distance among high-dimensional data points, LLE exploits the fact that each data points and its neighbors should lie close to a locally linear patch in a well-sampled manifold. In other words, by examining the local geometry of the manifold, LLE tries to reconstruct each data sample by its neighbors. This reconstruction can be solved in closed form; therefore, LLE avoid the problem of local minima [1]. The corresponding set of neighbors for each sample can be assigned in different ways-e.g., by choosing the k nearest neighbors (KNN) in Euclidean distance.

When the number of neighbors is greater than the number of input features or when the local geometry exploited by weight reconstruction in LLE is not well-determined (e.g.-dataset is not well sampled), the constrained least-squares optimization problems involved in LLE may be ill-conditioned. One approach to solve this issue is regularizing the problem. However, if the regularization parameter is not well-selected, the obtained solution is not a good approximation of the exact solution; and therefore, the optimal solution is not guaranteed to be found [2]. Moreover, LLE can be unstable and produce distorted embeddings when the manifold dimension is larger than one. One of the LLE variants, namely modified LLE (MLLE), introduces multiple linearly-independent weight vectors for each neighborhood on top of the conventional LLE to take care of the described issues [2].

The aim of this article is to explore the advantage of LLE and MLLE dimensionality reduction techniques over utilizing the raw dataset as well as to provide a comparison between these two methods. To have a more sophisticated comparison, these algorithms were tested alongside implementing the nonlinear SVM for binary classification on a chosen dataset provided for predicting survival or death of a number of hospitalized patients given some of their health characteristics. The selected performance metrics to test the hypothesis are execution time, F1 score, accuracy values, and confusion matrices. On top of the reduced execution time obtained by LLE and MLLE, the results indicate that LLE brings about higher F1 and accuracy scores as well as more diagonalized confusion matrix than the raw data and MLLE. MLLE also produce decent results such that they are only slightly worse than those of the raw data. Therefore, LLE exhibits a marginally better performance on this case.

The remainder of this article is organized as follows: Section II provides a more in-depth explanation of LLE and MLLE algorithms-e.g., their computational costs. Section III details the utilized dataset and corresponding preprocessing steps. The tuning procedure of hyperparameters for LLE and MLLE methods as well as for the kernel-SVM is described in section IV. Final results indicating comparison of LLE and MLLE

and the improvements over operating on the raw dataset are presented in Section V. Section VI concludes this paper and summarized the findings.

II. Methods

Both LLE and MLLE are nonlinear methods for reducing number of data features since they nonlinearly map high dimensional samples into lover dimensional embeddings. Prior to proceeding further into how these algorithms work, note that the following nomenclature is used throughout this paper.

TABLE I
NOMENCLATURE

Symbol	Description
N	Number of data points.
D	Original number of features.
d	The dimensionality of output embeddings.
k	Number of neighbors.
X	The data point vector containing all of the features for each sample.
Y	The low-dimensional embedding vector for each sample.
W_{ij}	The weight associated with contribution of *j*th data point to the *i*th construction.
N_{sv}	Number of support vectors for the SVM algorithm.

A. Algorithms

LLE algorithm consists of three steps which can be summarized as follows.

Algorithm 1 LLE

1: Assign neighbors to each data point $\vec{X}_i$.
2: Compute the weights W_{ij} that best reconstruct $\vec{X}_i$ from its neighbors by solving the constrained problem of minimizing the cost

$$\mathcal{E}(W) = \sum_i \left| \vec{X}_i - \sum_j W_{ij} \vec{X}_j \right|^2 . \quad (1)$$

3: Compute the low dimensional $\vec{Y}_i$ embeddings best reconstructed by the fixed weights W_{ij}, minimizing the quadratic cost

$$\Phi(Y) = \sum_i \left| \vec{Y}_i - \sum_j W_{ij} \vec{Y}_j \right|^2 \quad (2)$$

by finding its smallest nonzero eigenvectors.

Eq. 1 is a constrained optimization problem subjected to $\Sigma_j W_{ij} = 1$ which ensures weights invariance to translation of data points. Also note that, W_{ij} is considered to be 0 if $\vec{X}_j$ is not a neighbor of $\vec{X}_i$. The invariance of weights to linear mappings like translation means that same W_{ij} used in reconstructing $\vec{X}_i$ in high D dimensions are also capable of construing its embedded coordinates with low d dimensions as indicated in eq. 2.

Eq. 2 is solved subjected to $\Sigma_i \vec{Y}_i = \vec{0}$ (to center embeddings on the origin) and $\frac{1}{N}\sum_{i=1}^{N} Y_i Y_i^\top = I$ (unit covariance to avoid degenerate solutions). Solving eq. 2 requires dealing with a sparse $N \times N$ eigenvalue problem such that the smallest d non-zero eigenvectors are the ordered set of low-dimensional embeddings (the smallest eigenvalue is 0 which is discarded by the first constraint).

Similar to LLE, MLLE also tries to learn the local geometry of data points and map them nonlinearly to a lower-dimensional space by solving an eigenvalue problem. The difference, however, is that MLLE is more stable than LLE since it uses multiple weight vectors (instead of only one as in LLE) for each data sample to learn its local neighborhood structure and reconstructing its corresponding lower-dimensional embedding. The three main steps of MLLE algorithm are very similar to those of LLE. The only difference is that the eigenvalue problem which has to be solved in MLLE is more complex as it requires finding multiple weight vectors instead of a single optimal one.

B. Computational complexity

The LLE algorithm only requires a single pass among the described three steps, and it finds the global optima of the reconstruction. Given the above discussion, one can conclude that the overall complexity of LLE is $O[D\log(k)N\log(N)] + O\left[DNk^3\right] + O\left[dN^2\right]$. The first term is attributed to the first step of the algorithm which is finding the neighbors using KNN. The second and third terms correspond to the second and third steps of the algorithm respectively.

For MLLE, the extra step of finding multiple weight vectors instead of only one compared to the conventional LLE, increases the cost by $O\left[N(k-D)k^2\right]$ which accounts for constructing the weight matrix from multiple weight vectors. Therefore, the total complexity of MLLE is $O[D\log(k)N\log(N)] + O\left[DNk^3\right] + O\left[N(k-D)k^2\right] + O\left[dN^2\right]$.

Note that the cost associated with eigendecomposition of the weight matrix is linear with respect to number of data points, and also $k \ll N$. Whereas, neighborhood selection (its complexity can be approximated to be $O[DN^2]$) and finding the low dimensional embeddings ($O[dN^2]$) are both quadratic with respect to N. Therefore, the cost of additional step of MLLE is relatively small compared to the complexity of the first and third steps of the algorithm such that the computational cost of MLLE is almost the same as that of LLE.

C. Free parameters

In this project, KNN was used as the algorithm for neighborhood selection for both LLE and MLLE. It is noteworthy that both LLE and MLLE only have the number of neighbors (k) as their free parameter. Furthermore, for a fixed k, the dimensionality of the embeddings (d) that they can recover is strictly less than the number of neighbors [1]. Moreover, d

can be estimated in a reverse manner-i.e, by considering the reciprocal of the described cost functions. This means that the reconstruction weights can be derived from the embeddings $\vec{Y_i}$ in eq. 2 and then applied into eq. 1 to find the associated sample vectors $\vec{X_i}$. However, in this paper, a grid search is done to find both k and d since there was not a way to directly find the right value for d in this case.

III. Dataset

The dataset chosen for this paper concerns mortality state of $91{,}713$ hospitalized patients given 83 explanatory variables. Since the output variable has two states (survival or death), this dataset is suitable for binary classification. The provided features range from general characteristics like gender, age, and height to more health-related ones, such as highest and lowest heart rate, respiratory rate, and blood pressure during first hour and first day of patients' stay in the ICU units. The dataset can be accessed on https://www.kaggle.com/datasets/mitishaagarwal/patient.

A. Preprocessing

Prior to applying the dimensionality reduction techniques, the dataset needs to be cleaned and analyzed. Some meaningless features are deleted at first. For instance, patient, hospital, and ICU identifiers do not contribute any useful information to the classification task. Besides, a few number of features lacks sufficient description, or they are repetitive, or contains almost one unique value throughout the whole data points. All of these features (such as ICU stay type and diagnosis groups for APACHE II and III medical tests) are also removed.

Moreover, all the samples which have at least one missing feature are deleted from the study. Furthermore, among the remaining features, two categorical variables of gender and ethnicity are transformed into discrete numerical features. One-hot encoding is avoided in this case since it produces a significant number of features which reduces the classification performance in the end.

In addition, it is evident that features have different units and range of values; for instance, body temperature and height do not clearly have the same scale. Note that both LLE and MLLE as dimensionality reduction techniques and SVM as the classification method are intensively involved with calculating distances across data points. Therefore, it is necessary to rescale all features to prevent the model from learning large weight values. Since most of the feature distributions are normal in the dataset at disposal, the standardization method is utilized during which each feature is separately scaled by subtracting the mean (centering) and dividing by the standard deviation to shift the distribution to have a mean of zero and a standard deviation of one.

Based on the above discussion, a summary of the dimensionality of the prepossessed dataset is given in table. II. Clearly, the dataset has a large number of samples and significant number of features where dimensionality reduction would be of use.

TABLE II
Summary of the Preprocessed Dataset

#Patients	#Survived	#Died	#Features
56,986	52,091	4895	71

B. Balancing

It is evident from table II that the dataset is severely unbalanced since number of patients who died only constitute 8.59% of the number of data points. This skewed distribution can easily make SVM ignore the minority class-i.e.,patients who died, although it is the minority class on which predictions are most important. To address this issue, three approaches are taken: oversampling the minority class, undersampling the majority class-i.e., patients who survived, and the combination of both.

Oversampling is done with synthetic minority oversampling technique (SMOTE) during which a random example of the minority class is fist taken and then a synthetic example will be created by randomly selecting a point placed within the line connecting that example and one of its neighbors. Undersampling is implemented simply by taking a random proportion of the majority class. The best classification performance is obtained by solely doing the undersampling where all the samples of the minority class are kept and an equal number of data points from the majority class are chosen randomly such that each class constitutes 50% of the data samples. The total population of the sampled dataset used for doing further experiments is presented in table III.

TABLE III
Summary of the Sampled Preprocessed Dataset

#Patients	#Survived	#Died	#Features
9790	4895	4895	71

IV. Hyperparameter tuning

As mentioned previously, the LLE and MLLE are treated as if they have two free parameters, namely number of neighbors k and the dimension of embedding d.

For that purpose, the *sklearn* python library is used to run the LLE and MLLE algorithms from *sklearn.manifold.LocallyLinearEmbedding* [3]. In both methods, the algorithm used by KNN to find the nearest neighbors is chosen as *auto*. This allows the *sklearn* library to choose the best algorithm among *brute*, *kd_tree*, and *ball_tree* for neighborhood selection. Concerning the eigenvalue solver, the *auto* option is again used as the eigen_solver allowing the *sklearn* library to choose the best algorithm to use between *arpack* and *dense*.

Similarly, the implementation of nonlinear SVM (with radial basis function (RBF) kernel) for classification is selected from *sklearn.svm.SVC* [4] to assess the result of the performed dimensionality reductions. Kernel SVM also has two hyperparameters. One is the kernel width σ controlled by the parameter

gamma which is inversely proportional to σ^2. The other free parameter is C which is the regularization parameter that determines the cost of missclassifying data points.

The selection of hyperparameters are done in four steps and based on the best values obtained for F1 and accuracy scores after classification (here only the F1 scores will be plotted as both accuracy and F1 measures are highly correlated). Note that the sampled preprocessed dataset is splitted first into train and test sets where the test represents 1% of the data points. Given the number of data points and features in table III and since dealing with the overfitting problem is not the goal of this article, this test size is logical. A summary of the train and test populations is given in table IV.

TABLE IV
SUMMARY OF THE TRAIN AND TEST SETS

Set	#Patients	#Survived	#Died	#Features
Train	9692	4846	4846	71
Test	98	49	49	71

In the first step, the best value for embedding dimensionality (d) is found by doing a grid search. Afterward, the number of neighbors (k) is chosen in a similar fashion for both LLE and MLLE. Given the required time to perform LLE and MLLE to cover a wide grid as it will be addressed later, these two steps are done first by cutting the number of samples for both train and test sets to be half of what is reported in table IV (Note that the full sets is again utilized to focus the grid search on small regions of interests).

On the other hand, the search for the kernel width and the regularization parameter (C) of SVM is performed on the full sets with the aim of realizing the best classification performance on the raw and dimensionality-reduced datasets.

Finally, it is important to notice that executing all the following grid searches takes around 5 hours of computation time. Evidently, It was not possible to perform cross validation for the hyperparameter search due to time and computational power constraints existing for ordinary computers.

A. Choosing the dimensionality of embeddings

To find the dimensionality of embeddings (d), the hyperparameters of SVM is fixed to be $C = 1.0$ and $gamma = scale$ which means that, for now, *gamma* (inversely proportional to the square of kernel width (σ)) is equal to $\frac{1}{d * var(X)}$.

Note that for the grid search on dimensionality of embeddings, the number of neighbors is always chosen to be $k = d + 2$. This ensures that k is always greater than d as mentioned previously. More concretely, to begin with, a wide grid search on d is done first by only using half of the train and test sets (see fig. 1). The reference line corresponds to F1 score of the raw sets. Comparison with this line reveals that d around 11 and d around 23 seem to provide the best F1 scores for LLE and MLLE respectively.

Accordingly, a second grid search was also performed on a small region around the two peak values of interest for d. This second grid search is done using the full sets and the results are shown in fig. 2 for LLE (search around $d = 11$) and fig. 3 for MLLE (search around $d = 23$). It can be deduced from the figures that the best results are obtained for the following values of d:

- **LLE**: $d = 12$
- **MLLE**: $d = 19$.

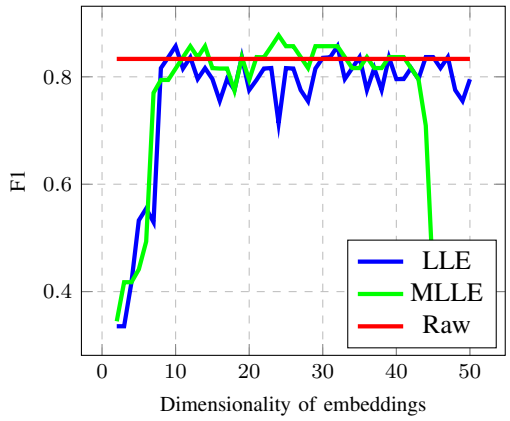


Fig. 1. F1 scores of wide search for d (with half of data)

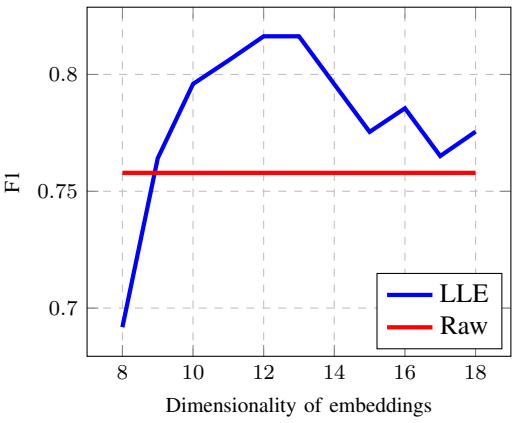


Fig. 2. F1 scores of focused search for d (LLE with full data)

B. Choosing the number of neighbors

Similar to the previous step, to find the number of neighbors (k), the hyperparameters of SVM is fixed to be $C = 1.0$ and $gamma = scale$. Note that, here, for implementing the grid search on the number of neighbors, the best values of d found previously-i.e, $d = 12$ for LLE and $d = 19$ for MLLE, are used. $k \geq 1$ for LLE and $k \geq 19$ for MLLE (the MLLE implantation of *sklearn* does not allow k to be smaller than d) are searched.

As before, first a wide grid search on k is done using only half of the data sets (see fig. 4). This results in value of k to be around 15 for LLE and around 25 for MLLE since they provide the best F1 scores compared to the reference line. Afterward, the second grid search is performed on a restricted

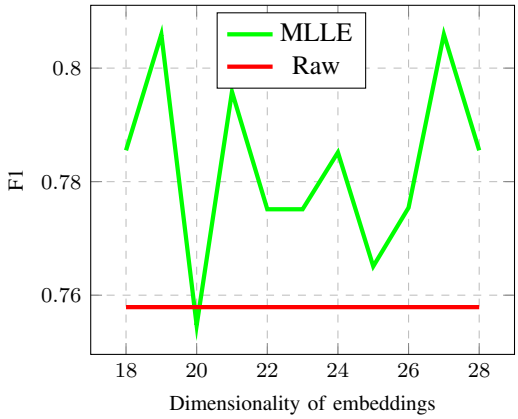


Fig. 3. F1 scores of focused search for *d* (MLLE with full data)

vicinity of those two values of interest for k using the full (see fig. 5 for LLE and fig. 6 for MLLE). The figures suggest the best k values to be:

- **LLE**: $k = 14$
- **MLLE**: $k = 22$.

Note that $k = 14$ is chosen for LLE although $k = 11$ provides a better F1 score as indicated in fig. 5. This is because of the fact that k needs to be greater than $d = 12$ as mentioned previously.

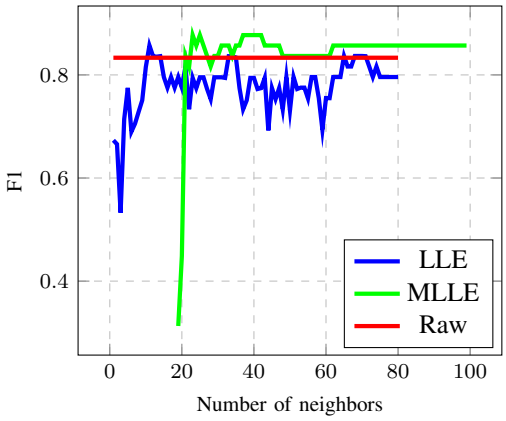


Fig. 4. F1 scores of wide search for *k* (with half of data)

C. Choosing the kernel width

To find the right kernel width (σ) of SVM corresponding to each method, the hyperparameter C of SVM is kept fixed to 1.0. Note that grid search here is done on *gamma* value of *sklearn* implementation which is inversely proportional to σ^2. The grid search is done while maintaining the best values obtained previously for the dimensionality of embeddings (d) and the number of neighbors (k).

The result of the primary wide grid search on *gamma* using the full sets is demonstrated in fig. 7. This graph suggests the *gamma* value of around 10^4 for LLE, around 10^3 for MLLE and approximately 10^{-5} for the raw samples. Similar to the previous two steps, a second grid search is performed on small

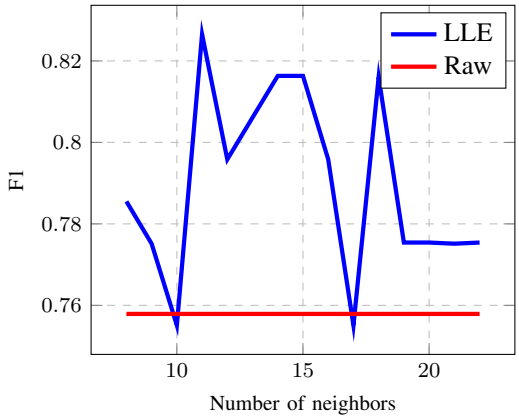


Fig. 5. F1 scores of focused search for *k* (LLE with full data)

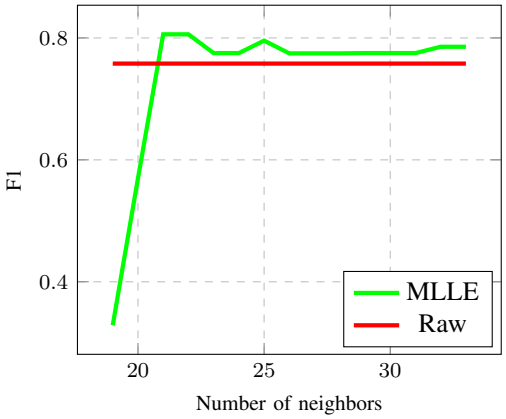


Fig. 6. F1 scores of focused search for *k* (MLLE with full data)

regions around those three values of interest for *gamma* using the full sets (see fig. 8, fig. 9, and fig. 10 for LLE, MLLE, and raw data respectively). The best results found for *gamma* is as follows:

- **LLE**: *gamma* $= 8859$
- **MLLE**: *gamma* $= 695$
- **Raw**: *gamma* $= 9 \times 10^{-5}$.

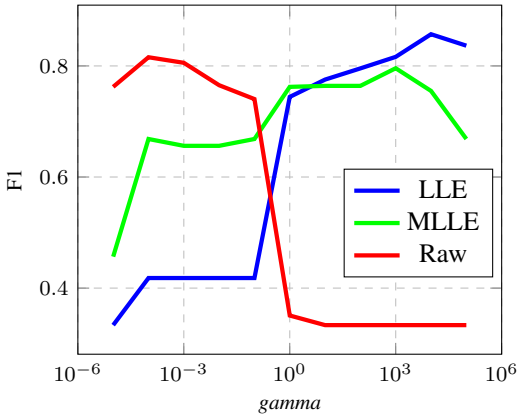


Fig. 7. F1 scores of wide search for *gamma*

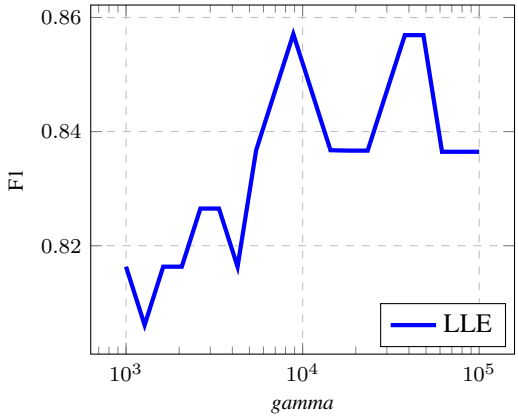


Fig. 8. F1 scores of focused search for *gamma* (LLE)

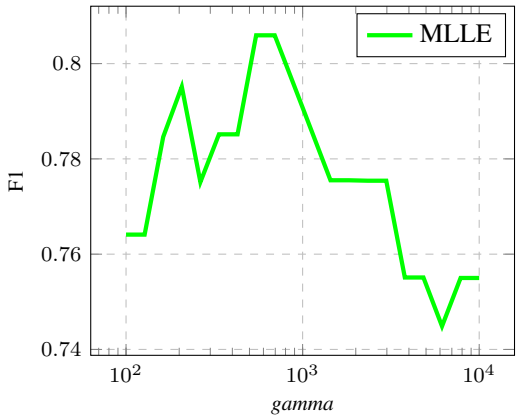


Fig. 9. F1 scores of focused search for *gamma* (MLLE)

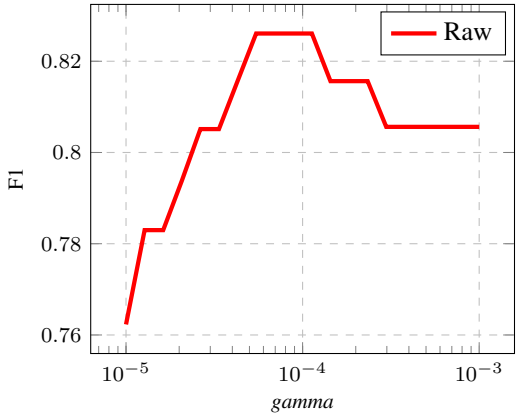


Fig. 10. F1 scores of focused search for *gamma* (raw dataset)

D. Choosing the regularization parameter

Finally, to find the last hyperparameter-i.e., the regularization parameter (C) of SVM, a grid search is done separately for LLE, MLLE, and the raw data while keeping the best values found previously for the dimensionality of embeddings (d), the number of neighbors (k), and the kernel parameter ($gamma$).

Similar to before, first a broad grid search is carried out on C by using the full data sets. The corresponding results are presented in fig. 11. Clearly, $C = 1.0$ seems to provide the best F1 score for LLE, MLLE, and raw data. Accordingly, a second grid search is also done on around $C = 1.0$ vicinity using the full sets. $C = 1.0$ still produces the best results.

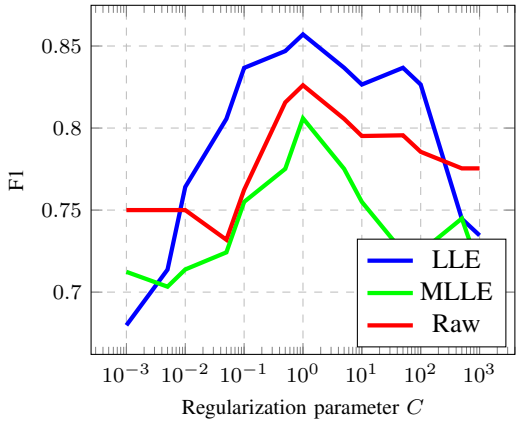


Fig. 11. F1 scores of wide search for regularization parameter

TABLE V
HYPERPARAMETERS

	d	***k***	***gamma***	***C***
LLE	12	14	8859	1.0
MLLE	19	22	695	1.0
Raw	-	-	9×10^{-5}	1.0

V. Results

A. Toy dataset

To demonstrate the functionality of the LLE and MLLE methods, they are applied on the Swiss roll dataset with 1500 samples (see fig. 12). The 2D embeddings are shown in fig. 13. Evidently, LLE very capably unrolls the manifold of the Swiss role. On the other hand, while preserving the general topology of original data, MLLE seems to densely clump some of the data points together-i.e. those who belong to the center of the Swiss roll. This can be justified by noting that actually there is a higher density of points near the center of the roll.

B. Real dataset

The results of the extensive hyperparameter grid search on the full sets of table. IV described previously is summarized in table V. Using this parameters for LLE and MLLE techniques as well as nonlinear SVM leads to the results demonstrated in table VI.

It can clearly be seen from table VI that SVM has already done a decent job using the raw full data since both of its F1 and accuracy scores are above 0.8. Moreover, LLE brings

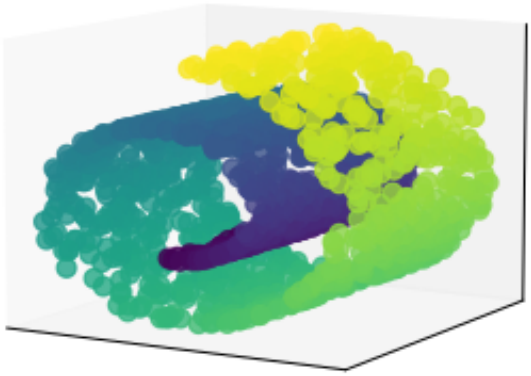
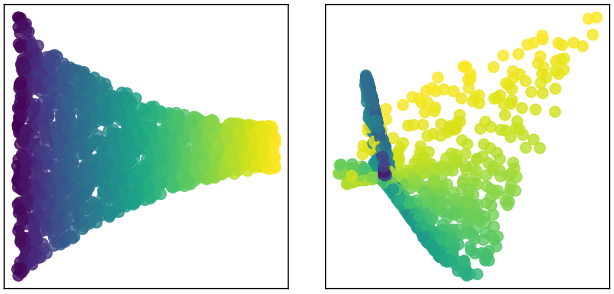


Fig. 12. Swiss roll data points



(a) LLE (b) MLLE

Fig. 13. The 2D embeddings of Swiss roll dataset obtained by (a) LLE and (b) MLLE.

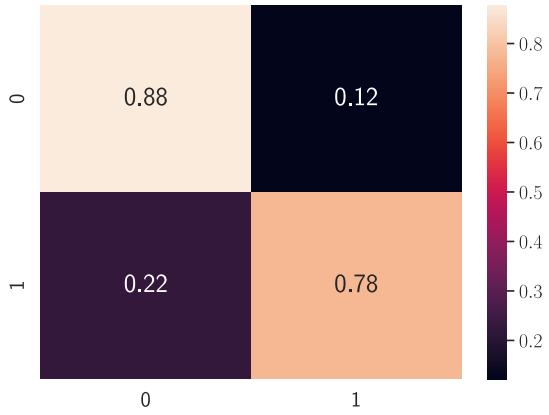
about a better results in terms of both F1 and accuracy values compared to raw data and MLLE. Besides, classifying with raw sets only slightly outperforms MLLE.

TABLE VI
RESULTS

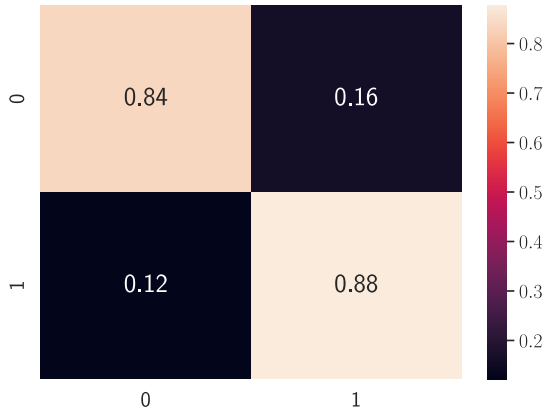
	F1 score	Accuracy	Classification time (s)	Dimensionality reduction time (s)
Raw	0.826	0.827	4.14	-
LLE	0.857	0.857	3.36	71.79
MLLE	0.806	0.806	2.72	74.49

Note that the computational complexity of kernel SVM is $O\left(N^2\ D(or\ d) + N^3\right)$ and $O\left(N_{sv}\ D(or\ d)\right)$ in training and testing respectively [5]. Clearly, SVM is sensitive to the number of data points (N); however, as the dimension of embeddings is reduced by LLE and MLLE, the time required required for classification by kernel SVM is also clearly diminished (18.8% by LLE and 34.3% by MLLE). This amount of time saving would be more apparent if the classification were applied on larger datasets. Besides, as pointed out previously, the time required for performing MLLE is almost the same as the that of LLE.

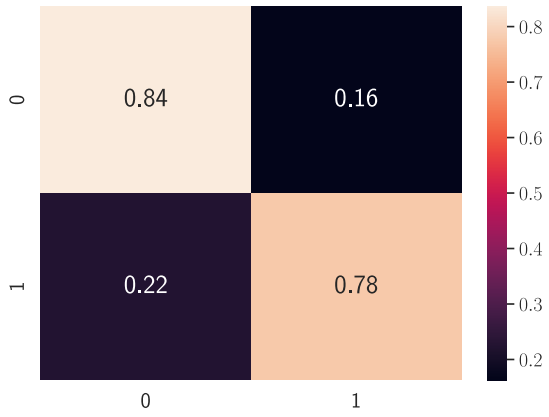
The corresponding confusion matrices are presented in fig. 14. Since the bottom row of each confusion matrix is



(a) Raw



(b) LLE



(c) MLLE

Fig. 14. Confusion matrices obtained by learning kernel SVM on the full train and test sets: (a) Raw dataset, (b) and (c) Dimensionality-reduced data by LLE and MLLE respectively.

concerned with the original minority class-i.e., the patients who died, a careful inspection reveals that LLE has a superior performance over both raw data and MLLE. This is because the model should not afford to predict death incorrectly. On the other hand, both LLE and MLLE demonstrates a slightly worse performance in predicting the true positives-i.e., the survived patients.

All in all, both LLE and MLLE bring about a faster and yet accurate enough classification performance as expected. Besides, MLLE tends to make SVM performs faster than on the LLE although the MLLE dimensionality reduction itself is slightly slower than LLE. Moreover, LLE produces higher F1 and accuracy scores and also exhibits a better performance on the minority class compared to both MLLE and raw data. In addition, MLLE also provides decent results as described. Therefore, one can conclude that LLE is a superior choice over MLLE on this dataset.

VI. Conclusion

In this project, we have attempted to explore the manifold learning by examining two dimensionality reduction techniques, namely locally linear embedding (LLE) and its modified variant called MLLE. The dataset chosen for this project aims at predicting death or survival of hospitalized patients in ICU units given some of their body and health characteristics. Prior to performing any machine learning modelling, the dataset was cleaned to get rid of missing values and irrelevant features. Furthermore, as both LLE and MLLE as well as kernel SVM (used to perform binary classification) relies heavily on measuring the distance between the samples, the remaining features are separately scaled such that all have zero mean and unit standard deviation.

Afterward, to solve the severely unbalanced issue existing in the original dataset, the majority class-i.e., the patients who survived, are undersampled randomly such that we ended up having a perfectly balanced dataset where both minority and majority class constituted 50% of data points. Thereafter, we carried out an extensive grid search to find the right choice of hyperparameters (dimensionality of the embeddings and number of neighbors for LLE and MLLE as well as RBF kernel width and regularization parameter C for nonlinear SVM) for the raw sets and those dimensionality-reduced by LLE and MLLE.

Using the obtained free parameters, we have successfully demonstrated the time saving during classification by both LLE and MLLE compared to the raw data as a result of dimensionality reduction. We also showed that LLE outperforms both MLLE and raw data since it exhibited larger F1 and accuracy scores as well as a superior performance on the minority class suggested by its associated confusion matrix. Besides, the results indicated that MLLE performs almost equally to the raw data in terms of all the utilized metrics.

References

[1] S. T. Roweis and L. K. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[2] Z. Zhang and J. Wang, "Mlle: Modified locally linear embedding using multiple weights," *Advances in neural information processing systems*, vol. 19, 2006.

[3] "sklearn.manifold.LocallyLinearEmbedding." [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.manifold.LocallyLinearEmbedding.html

[4] "sklearn.svm.SVC." [Online]. Available: https://scikit-learn/stable/modules/generated/sklearn.svm.SVC.html

[5] M. Claesen, F. De Smet, J. A. Suykens, and B. De Moor, "Fast prediction with svm models containing rbf kernels," *arXiv preprint arXiv:1403.0736*, 2014.