Report for Data Science Project 1

Xiaoshuang Chen†, Xu Lin†, Xinpeng Liu†, Yue Xu†,

***Abstract*— The raw data for machine learning are often noisy, redundant and high-dimensional, which largely increases the complexity of the problem and difficulty in classification task. Thus, dimensionality reduction plays an important role in data mining and machine learning as a data pre-processing method. In this paper, we evaluated the performance of some popular dimensionality reduction algorithms on Animals with Attributes (AwA2) data set[1], including variance feature selection, PCA, kernel PCA, LDA, t-SNE, LLE, MDS and VAE. Furthermore, we raised some conjectures from our results and conducted further experiment to explain the reasons of our results.**

I. INTRODUCTION

The raw data for machine learning are often noisy, redundant and high-dimensional, which largely increases the complexity of the problem and difficulty in classification task. Thus, dimensionality reduction plays an important role in data mining and machine learning as a data pre-processing method, aiming at avoiding the curse of dimensionality and reducing the complexity, as well as de-noising, compressing or visualizing the data. In practice, a proper dimensionality reduction can significantly enhance the performance of machine learning algorithms.

With decades of development, there are many approaches to reduce dimensionality and map the features into lower dimensional space. The most naive method is **feature selection**, which reduce the dimensionality by selecting some "important" dimensions and discarding the rest. Feature selection is equivalent to a search problem and can be accelerated by greedy algorithm (e.g. forward or backward feature selection), genetic algorithm and so on. The most popular dimensionality reduction algorithms try to find lower dimensional linear combinations of the original features by learning a projection matrix W. The so-called **feature projection** method includes a large variety of algorithms such as PCA[2], kernel PCA[3], LDA[4], etc. Nowadays, **feature learning** approaches grow rapidly and have taken over the state-of-the-art of some areas like visualization. They manage to retain the data distribution (SNE, t-SNE[5]) or manifold structure (LLE[6], MDS, IsoMap), or use deep learning method (Auto-encoder, VAE[7]), which are suitable under various conditions.

In this paper, we evaluated the performance of some popular dimensionality reduction algorithms on Animals with Attributes (AwA2) data set[1], including variance feature selection, PCA, kernel PCA, LDA, t-SNE, LLE, MDS and VAE. We compared the SVM classification accuracy of the reduced features and the visualization effects. The algorithms are introduced in detail and experimented in section II and section III. Moreover, we raised some conjectures from our results and conducted further experiment to explain the reasons of our results in section IV.

† Equal contribution.

II. METHOD

A. Feature Selection by Variance

Feature selection is the simplest approach for dimensionality reduction. It is accomplished by removing some certain dimensions and keeping the rest.

One of the fastest and most popular methods of feature selection is selecting by variance. It can be regarded as both forward feature selection algorithm and backward algorithm. And it can also be seen as a non-projected version of PCA. Since the dimensions that have larger variance are more likely to contain key information of the data, we can reduce the dimensionality to k by selecting the k largest dimensions by variance.

B. PCA

Principle component analysis is a well-known method of linear feature projection, which was invented by Karl Pearson[2].
The basic idea is to project data to a space of lower dimensionality, maximizing the variance along each projected component to preserve more useful information. Formally, the optimization goal is

$$\max_{v} \; v^T X^T X v, \;\; s.t. \;\; v^T v = 1, \tag{1}$$

where v is the the new axis that X is projected to. To optimize Equation 1, we introduce its Lagrangian form:

$$\begin{aligned} &L_v = v^T X^T X v - \lambda (v^T v - 1), \\ &\frac{\partial L}{\partial v} = X^T X v - \lambda v = 0, \\ &X^T X v = \lambda v. \end{aligned} \tag{2}$$

We can see that v is actually the eigenvector of $X^T X$, and λ is the corresponding eigenvalue. Moreover, if we substitute λ back, we will find that

$$v^T X^T X v = v^T \lambda v = \lambda v^T v = \lambda . \tag{3}$$

Therefore, if we want to project n-dimension data matrix X to k-dimension data matrix X', we can first perform eigenvalue decomposition to the co-variance matrix $X^T X$, then choose the top k eigenvalues and corresponding eigenvectors as v and perform the projection. In this way, we make sure the variance is maximized along the k new axis.

In implementation, we perform singular decomposition to X instead of performing eigenvalue decomposition to X^T to achieve better speed.

In our experiment, we performed PCA algorithm provided in scikit-learn on the data set.

C. Kernel PCA

Kernel PCA[3] is an extension of PCA, using kernel methods, to perform PCA in a non-linear space.

It was inspired by the observation that when N points can't be linearly separated in $d < N$ dimensions, it can almost always separated in $d > N$ dimensions. To exploit the observation, we assume that there is a non-trivial function $\Phi(x)$ that project point x to a higher-dimension space. Then we replace X with $\Phi(X)$ in Equation 2.

$$\begin{aligned} L_v &= v^T \Phi(X)^T \Phi(X) v - \lambda (v^T v - 1) \\ \frac{\partial L}{\partial v} &= \Phi(X)^T \Phi(X) v - \lambda v = 0 \\ \Phi(X)^T \Phi(X) v &= \lambda v \end{aligned} \tag{4}$$

We can see that $\Phi(X)$ always appears in the form of $\Phi(X)^T \Phi(X)$. Considering the cost of computing $\Phi(x)$ for each data point and storing them, we can introduce $K(x,y) = \Phi(x)^T \Phi(y)$, which is called kernel function, to simplify the calculation.

In our experiment, we performed Kernel PCA algorithm provided in scikit-learn on the data set, and compared performance of polynomial, rbf, sigmoid and cosine kernels. The expression of these four kernels is presented in Equation 5

$$\begin{aligned} K_{Poly}(x,y) &= (\alpha x^T y + b)^d \\ K_{RBF}(x,y) &= \exp(-\frac{\|x-y\|^2}{2\sigma^2}) \\ K_{Sigmoid}(x,y) &= tanh(\alpha x^T y + b) \\ K_{Cosine}(x,y) &= \frac{x^T y}{\|x\|\|y\|} \end{aligned} \tag{5}$$

D. LDA

Linear Discriminant Analysis[4] is a method used to find a linear combination of features that characterizes or separates two or more classes of objects. The idea of LDA can be summarized in one sentence, that is, "the intra-class variance is the minimum after projection, and the inter-class variance is the maximum".

Let's start with LDA for two classes. We define μ_j as mean vector of the jth sample while Σ_j as co-variance matrix of the jth sample.

Formally, our goal is:

$$\underset{\omega}{\text{argmax}}\ J(\omega) = \frac{\omega^T (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T \omega}{\omega^T (\Sigma_0 + \Sigma_1) \omega} \tag{6}$$

We define matrix S_ω for the distance between the category centers of different categories of data, S_b for co-variance of projection points of the same sample, so our goal will change to:

$$\underset{\omega}{\text{argmax}}\ J(\omega) = \frac{\omega^T S_b \omega}{\omega^T S_\omega \omega} \tag{7}$$

According to the property of generalized Rayleigh quotient, we can determine the best projection direction ω.

If we need to deal with data with multi-class, the goal should be:

$$\underset{W}{\text{argmax}}\ J(W) = \prod_{i=1}^{d} \frac{\omega_i^T S_b \omega_i}{\omega_i^T S_\omega \omega_i} \tag{8}$$

The maximum value is maximum eigenvalue of the matrix $S_\omega^{-1} S_b$, the product of the largest d values is the product of the largest d eigenvalues of the matrix $S_\omega^{-1} S_b$. In this situation W is the matrix spanned by the eigenvectors of the largest d eigenvalues. Because W is a projection matrix using the category of the sample, the maximum value of d that it reduces to is $k-1$.

In our experiment, we performed LDA algorithm provided in scikit-learn on the data set.

E. t-SNE

The technique t-SNE[5] is a common method for visualizing high-dimensional data by giving each data point a location in a two or three-dimensional space, which is a variation of Stochastic Neighbor Embedding[8]. It produces significantly brilliant visualization by reducing the tendency to crowd points together in the center of the map.

Stochastic Neighbor Embedding (SNE) evaluates the similarity of data point x_j to data point x_i by

$$p_{j|i} = \frac{\exp\left(-\left\|x_i - x_j\right\|^2\right)}{\sum_{k \neq i} \exp\left(-\left\|x_i - x_k\right\|^2\right)} \tag{9}$$

Similarly, in low-dimensional space, the similarity $q_{j|i}$ is defined as

$$q_{j|i} = \frac{\exp\left(-\left\|y_i - y_j\right\|^2\right)}{\sum_{k \neq i} \exp\left(-\left\|y_i - y_k\right\|^2\right)} \tag{10}$$

In t-SNE, however, Gaussian distribution is replaced by t-distribution, so that the similarity of data point x_j to data point x_i is defined as

$$q_{j|i} = \frac{\left(1 + \left\|y_i - y_j\right\|^2\right)^{-1}}{\sum_{k \neq i} \left(1 + \left\|y_k - y_i\right\|^2\right)^{-1}} \tag{11}$$

The t-SNE has two main advantages over SNE. First, the t-SNE gradient strongly repels dissimilar data points that are modeled by a small pairwise distance in the low-dimensional representation. Second, these repulsion does not go to infinity. For these reasons, the t-SNE has considerable performance on data visualization.

F. LLE

Locally Linear Embedding[6] (LLE) is a well-known manifold learning algorithm. As the name has suggested, LLE tries to keep the linear relation of the neighbors of each sample when reducing the dimensionality.

To keep the local linearity, we can firstly learn the linear relation W between each sample and its neighbors, which can be formalized as:

$$W = \underset{W}{\operatorname{argmin}} \sum_{i=1}^{m} \|x_i - \sum_{j \in KNN(x_i)} w_{ij} x_j\|), \tag{12}$$

where $KNN(x_i)$ is the k-nearest neighbors of sample x_i

Then we can reconstruct the low-dimensional features without changing the linear relation W:

$$Y = \underset{y_i}{\operatorname{argmin}} \sum_{i=1}^{m} \|y_i - \sum_{j \in KNN(x_i)} w_{ij} y_j\|). \tag{13}$$

G. MDS

MDS(Multi-dimensional scaling) algorithm requires the distance between samples in the original space to be maintained in the low-dimensional space.

Assuming that the distance between m samples is D in the distance matrix of the original control, we aim to obtain the sample matrix Z with the dimension reduced to d:

$$D \in R^{m \times m} \rightarrow Z \in R^{d \times m} \tag{14}$$

The middle distance between the ith sample and the jth sample in D is $dist_{ij}$, in Z is $\|Z_i - Z_j\|$ then $dist_{ij} = \|Z_i - Z_j\|$. Next we define $B = Z^T Z$ then we will get:

$$dist_{ij}^2 = \|z_i - z_j\|^2 = \|z_i\|^2 + \|z_j\|^2 - 2z_i^T z_j = b_{ii} + b_{jj} - 2b_{ij} \tag{15}$$

For the convenience of discussion, let's centralize the sample matrix Z, that is:

$$\sum_{i=1}^{m} b_{ij} = \sum_{j=1}^{m} b_{ij} = 0 \tag{16}$$

Now we can get:

$$\begin{aligned}
\sum_{i=1}^{m} dist_{ij}^2 &= \sum_{i=1}^{m} \|z_i\|^2 + \|z_j\|^2 - 2z_i^T z_j = \sum_{i=1}^{m} b_{ii} + m b_{jj} \\
\sum_{j=1}^{m} dist_{ij}^2 &= \sum_{i=1}^{m} \|z_i\|^2 + \|z_j\|^2 - 2z_i^T z_j = \sum_{j=1}^{m} b_{jj} + m b_{ii} \\
\sum_{i=1}^{m} \sum_{j=1}^{m} dist_{ij}^2 &= \sum_{i=1}^{m} \sum_{j=1}^{m} \|z_i\|^2 + \|z_j\|^2 - 2z_i^T z_j = 2m \sum_{i=1}^{m} b_{ii}
\end{aligned} \tag{17}$$

Then we get:

$$\begin{aligned}
dist_{i\cdot}^2 &= \frac{1}{m} \sum_{j=1}^{m} dist_{ij}^2 \\
dist_{\cdot j}^2 &= \frac{1}{m} \sum_{i=1}^{m} dist_{ij}^2 \\
dist_{\cdot\cdot}^2 &= \frac{1}{m^2} \sum_{i=1}^{m} \sum_{j=1}^{m} dist_{ij}^2
\end{aligned} \tag{18}$$

According to these equation, we deduce that:

$$b_{ij} = -\frac{1}{2}(dist_{ij}^2 - dist_{i\cdot}^2 - dist_{\cdot j}^2 + dist_{\cdot\cdot}^2) \tag{19}$$

By eigenvalue decomposition of distance matrix $B (B = V \Lambda V^T)$, Z expression can be obtained:

$$Z = \Lambda_*^{\frac{1}{2}} V_*^T \tag{20}$$

H. VAE

Variational Auto-Encoder (VAE)[7] is a classical example of Auto-Encoder Variational Bayes (AEVB) algorithm, which uses a neural network for recognition model. The algorithm allows us to perform very efficient approximate posterior inference using simple ancestral sampling, which in turn allows us to efficiently learn the model parameters, without expensive iterative inference schemes per data point.

VAE uses a neural network for the probabilistic encoder $q_\phi(\mathbf{z}|\mathbf{x})$ (the approximation to the posterior of the generative model $p_\theta(\mathbf{x}, \mathbf{z})$) and where the parameters ϕ and θ are optimized jointly with the AEVB algorithm.

Let the prior over the latent variables be the centered isotropic multivariate Gaussian $p_\theta(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$. We let $p_\theta(\mathbf{x}|\mathbf{z})$ be a multivariate Gaussian whose distribution parameters are computed from $\mathbf{z}$ with a MLP (a fully-connected neural network with one hidden layer). In this case we can let the variational approximate posterior be a multivariate Gaussian with a diagonal co-variance structure:

$$\log q_\phi(\mathbf{z}|\mathbf{x}^{(i)}) = \log \mathcal{N}\left(\mathbf{z}; \mu^{(i)}, \sigma^{2(i)} \mathbf{I}\right) \tag{21}$$

where the mean and s.d. of the approximate posterior, $\mu^{(i)}$ and $\sigma^{2(i)}$, are outputs of the encoding MLP.

We sample from the posterior $\mathbf{z}^{(i,l)} \sim q_\phi(\mathbf{z}|\mathbf{x}^{(i)})$ using $\mathbf{z}^{(i,l)} = g_\phi\left(\mathbf{x}^{(i)}, \varepsilon^{(l)}\right) = \mu^{(i)} + \sigma^{(i)} \odot \varepsilon^{(l)}$ where $\varepsilon^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. With $\odot$ we signify an element-wise product. The resulting estimator for this model and data point $\mathbf{x}^{(i)}$ is:

$$\begin{aligned}
&\mathscr{L}\left(\theta, \phi; \mathbf{x}^{(i)}\right) \\
&\simeq \frac{1}{2} \sum_{j=1}^{J} \left(1 + \log\left(\left(\sigma_j^{(i)}\right)^2\right) - \left(\mu_j^{(i)}\right)^2 - \left(\sigma_j^{(i)}\right)^2\right) \\
&+ \frac{1}{L} \sum_{l=1}^{L} \log p_\theta\left(\mathbf{x}^{(i)}|\mathbf{z}^{(i,l)}\right) \\
&\text{where} \quad \mathbf{z}^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \odot \varepsilon^{(l)} \\
&\text{and} \quad \varepsilon^{(l)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})
\end{aligned} \tag{22}$$

III. Experiment

In this section, we used various popular dimensionality reduction methods described in section II. If not mentioned, the algorithm is implemented with `Python 3` and `sklearn` package.

A. Data Set

We evaluated the chosen dimensionality reduction method on Animals with Attributes (AwA2) data set[1]. This data set consists pre-extracted 2048-dimension deep learning features for 37322 images of 50 animal classes. We split the images in each category into 60% for training and 40% for testing.

B. Baseline

We used linear SVM image classification based on the deep learning features as baseline. To determine the hyper parameters of SVM, we performed 5-fold cross-validation on

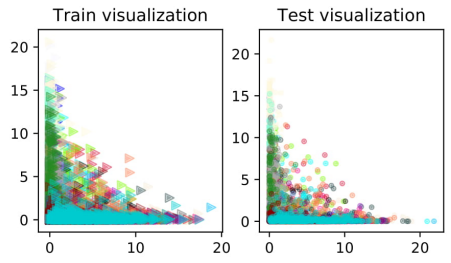


Fig. 1. 2D Visualization of Variance Feature Selection

train set. The validation results is shown in Table I. We finally chose one-vs-rest decision function and set C to 0.8 for the following experiments. The baseline method's accuracy on test set is 93.05%.

C \ Decision function	one vs one	one vs rest
0.2	92.29	92.30%
0.4	92.36	92.46%
0.6	92.34	92.33%
0.8	92.51	**92.53%**
1.0	92.24	92.30%

TABLE I

CLASSIFICATION RESULT OF LINEAR SVM FOR VALIDATION

C. Feature Selection by Variance

We implemented variance feature selection directly with Python `Numpy` and evaluated with SVM described in subsection III-B. The classification results and 2D visualization are shown in Table II and Figure 1.

Dim.	Acc.
Baseline	93.05%
2	9.82%
3	12.07%
10	46.64%
50	81.37%
100	87.07%
250	89.47%
500	91.53%
1000	92.55%
1500	92.88%
2000	**93.06%**

TABLE II

CLASSIFICATION ACCURACY OF VARIANCE FEATURE SELECTION

Overall, more dimensions results in higher accuracy. Since the input data are pre-processed deep learning feature, they have been reconstructed and may have more redundancies. Thus the simple feature selection perform quite well on the data: the 50 dimensions with largest variance achieve appreciable accuracy of 81.37%. And reducing the dimensionality to 2000 can even enhance the performance to 93.06%, which outperforms our baseline model. This may be due to the noise of the original data.

D. PCA and Kernel PCA

In this experiment, five different kernels was evaluated: linear, polynomial, RBF (actually Gaussian in our experiment), sigmoid, and cosine. Noticing the similarity between kernels used in SVM and PCA, we also evaluated performance of SVM using corresponding kernels on raw data as supplementary baselines.

For PCA, we used full svd solver. For poly kernels, we set its degree to 3, coefficient to $1/n_features$ and independent term to 1. The shared parameters were the same for other kernels.

The results are shown in Table III.

Acc. $\mathscr{K}$ / Dim.	Linear	Poly	RBF	Sigmoid	Cosine
Baseline	**93.05%**	91.03%	**93.10%**	**92.87%**	-
2	25.61%	24.28%	19.94%	23.79%	28.34%
3	37.34%	34.48%	34.62%	34.72%	41.93%
10	77.90%	77.98%	75.99%	73.41%	78.94%
50	89.89%	91.65%	90.25%	88.77%	91.20%
100	90.31%	92.52%	91.51%	90.00%	92.19%
200	91.15%	92.99%	92.14%	90.52%	92.79%
500	92.24%	93.35%	92.58%	90.82%	93.16%
750	92.48%	93.38%	92.64%	90.90%	93.20%
1000	92.67%	93.48%	92.71%	90.89%	93.25%
1500	92.92%	**93.52%**	92.81%	90.91%	**93.27%**

TABLE III

CLASSIFICATION RESULT WITH PCA AND KERNEL PCA

We also visualized the data reduced to dimension 2 in Figure 2 and Figure 3.

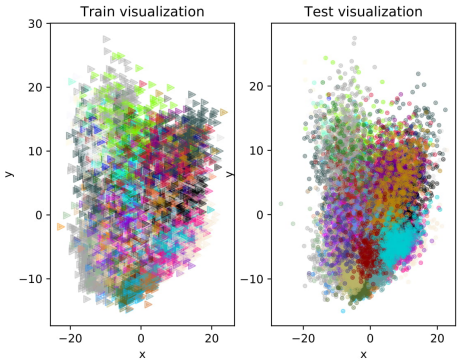


Fig. 2. PCA visualization

Generally, for linear PCA, the more dimensions preserved, the more information is saved, and the higher is the accuracy. We think it may be caused by different factors:

1) From the 2D visualization result, the data seems not separable, which supports the poor performance with 2D reduced data for classification. Since we used linear SVM for classification, the probable non-linearity was never considered when using PCA for dimensionality reduction.
2) PCA is based on the assumption that components with bigger variance contain more information helpful to classification. The assumption may not work

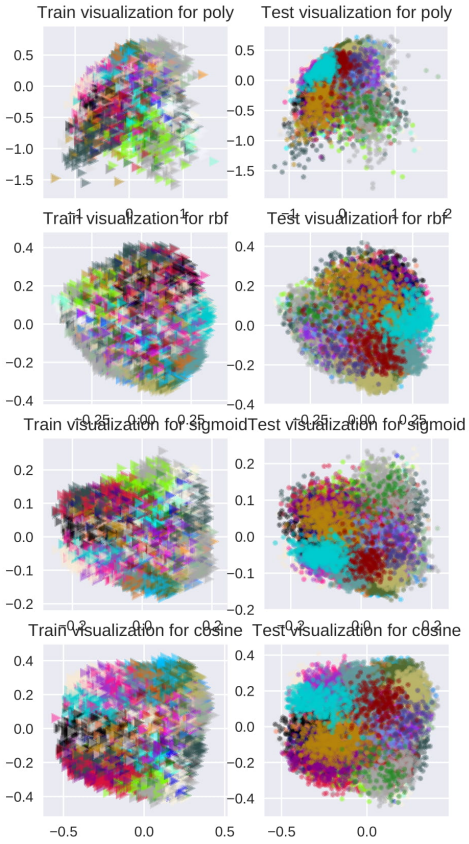


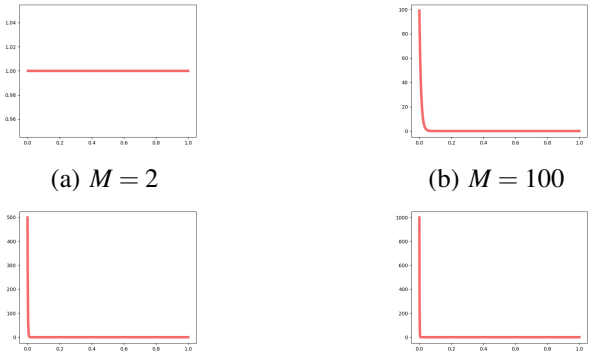
Fig. 3. Kernel PCA visualization

for extracted deep learning features. The components with smaller variance could also be important. PCA simply ignores them, which may result in acceptable performance but will definitely fail to get better.

For PCA with kernel function, we also have some observations and analysis:

1) Polynomial kernel achieved best performance among all. We thought it indicated the effectiveness of considering the combinations of different features for this classification task. However, we can see that polynomial kernel didn't work well on original data, which might indicate that the combination of different features can also bring redundancy or enhance the noises, declining the performance. Thus, dimensionality reduction can result in some performance improvement.

2) Cosine kernel offered better accuracy than linear PCA. To explain this, we can take a look at the expression of cosine kernel, and find that it simply performs L2 normalization to data points before computing inner product. This observation indicates that normalization can still play an important role in making the features' information easier to be exploited.
Moreover, we can see that cosine kernel produced significantly better performance for lower dimensionality comparing to others, which is also interesting. Now suppose we are given two independent unit norm random vectors u and v on M dimensional sphere S^M, and u, v are uniformly distributed on the sphere. The inner product of u and v will follow distribution $Beta(1, M-1)$. Here, we illustrate its probability density function with different M in Figure 4. We can see that the inner products become much more near to each other as the dimensionality grows. Therefore, the improvement of L2 normalization, or cosine kernel, might be less significant for higher dimension situation.



(a) $M = 2$ (b) $M = 100$

(c) $M = 500$ (d) $M = 1000$

Fig. 4. $Beta(1, M-1)$ with different Ms.

3) RBF kernel, or Gaussian kernel in our experiment, provided slightly worse performance comparing to linear, polynomial and cosine kernels. But for original data, it achieved the best performance among all. The performance gap, we think, was caused by the inappropriate variance metric used in PCA. From the expression, we can see that $K_{Gaussian}(x, y) = \exp(-\frac{\|x-y\|^2}{2\sigma^2})$ actually represents the probability that x follows distribution $\mathcal{N}(x, \sigma)$. Therefore, the computed 'co-variance matrix' actually has its probability meanings, and simply selecting the dimension with biggest variance might not be a good choice for this classification task. In other words, the assumption that dimensions with bigger variance contain more useful information could fail for Gaussian kernels.
4) Sigmoid kernel is the only kernel that was defeated by simple linear kernel in all the situations. For some parameters, its 'co-variance matrix' is not semi-positive definitive, the PCA algorithm might not be suitable for its processing. For some other parameters, it behaves like RBF kernel [9]. We think these might influenced the relatively poor performance of sigmoid kernel.

Furthermore, noticing that in subsection III-C, we used similar strategy to select features. Comparing PCA and feature selection results, we can easily see that PCA improved classification accuracy significantly. Why did the same maximizing-variance idea turn out to behave differently? We think the projection process is the key. PCA is an orthogonal transformation, which means data are projected to a space where axes are pairwise orthogonal. Without PCA, simple feature selection ignores correlation between the selected features and others. Therefore, when reducing to the same dimensionality, PCA preserves more useful information than simple feature selection by variance, which results in the performance gap.

E. LDA

The experiment results of LDA are shown in Table IV. We also visualized the data reduced to dimension 2 in Figure 5.

Dimension	Accuracy
Baseline	**93.05%**
2	30.99%
3	41.36%
10	71.67%
49	**91.60%**

TABLE IV
CLASSIFICATION RESULT WITH LDA

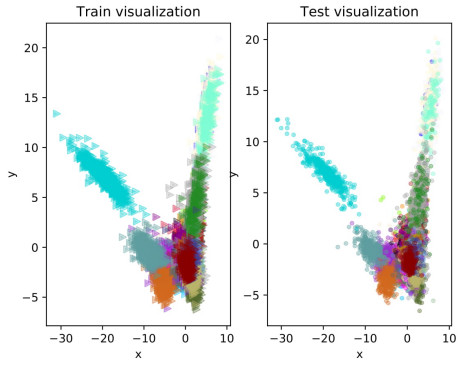


Fig. 5. LDA 2D visualization

Notice that the LDA was only performed on dimensions smaller than 50. The reason has been stated in subsection II-D. Since the projection matrix W in the LDA algorithm is obtained according to the type of the sample, the dimensional d of the dimensional reduction is at most $k-1$ (k is the number of types of samples).

LDA is an algorithm supervised by classification labels. Therefore, its reduction result is effective for classification tasks. We can see its accuracy was the best of all the algorithms when reducing data to 50 dimensions.

F. t-SNE

In our experiment, t-SNE used Barnes-Hut approximation to calculate gradient on dimension 2 and 3, and calculated exact gradient for other dimensions. Due to the computing capacity limit, we did not get the reduction result on dimension higher than 500.
The result is shown in Table V. We also visualized the data reduced to dimension 2 in Figure 6.

Dimension	Accuracy
Baseline	**93.05%**
2(Barnes Hut)	86.58%
3(Barnes Hut)	87.82%
2	86.15%
3	86.83%
10	80.80%
50	90.21%
100	90.46%
200	91.36%
500	**92.12%**

TABLE V
CLASSIFICATION RESULT WITH T-SNE

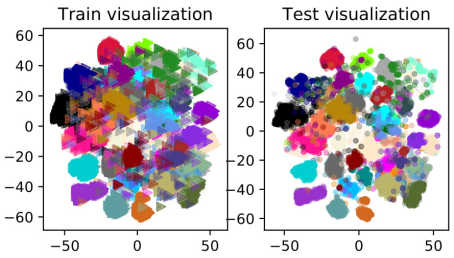


Fig. 6. t-SNE 2D visualization

The t-SNE provided the best performance on less than 10 dimensions among all of our models. We consider its extraordinary performance comes from its choice on how to model relations between data points, which has been stated in subsection II-E. The introduction of Gaussian distribution for high dimension space preserves more useful information than simple distance used in MDS, which is proved ineffective for this task in subsection III-H. And the use of t distribution for lower dimension space enables different data clusters to be distant enough and the data points belonging to the same cluster to be close, which benefits classification.

However, despite its brilliant performance, t-SNE cannot be a good choice for dimension reduction. Because it has no global optimum solution for a certain data distribution, we can't simply perform t-SNE separately on train set and test set, since the algorithm might return totally different data mappings, thus the classifier trained on train set can't work on test set. This is also the obstruction for MDS and LLE. A possible solution is to train a regressor to regress the low-dimension coordinate given the high-dimension data point. However, considering the cost of resource and time, it's not quite a feasible way. Therefore, t-SNE is usually used for data visualization instead of dimensionality reduction for downstream tasks.

G. LLE

Unfortunately, due to the limitations of computing resources (memory), we only conducted LLE experiments on 2, 3 and 5 dimensions. We also tried different numbers of nearest neighbors K. The results are shown in Table VI.

Not surprisingly, higher dimensionality brings higher accuracy. And although the dimensionality is too low to have an accurate performance, LLE can easily achieve higher accuracy than simple feature selection methods and even better than PCA at some choices of K.

Acc. K / Dim.	4	8	16
Baseline		**93.05%**	
2	9.81%	41.10%	35.35%
3	28.11%	45.71%	42.13%
5	36.55%	54.22%	63.27%

TABLE VI
CLASSIFICATION RESULT WITH LLE

We also visualized the 2 dimensional features at different K in Figure 7. The visualization results of LLE is much worse than t-SNE since its performance highly depends on the distribution and manifold structure of the raw data. Besides, the best visualization figure is at $K = 8$, which is consistent with the conclusion about accuracy above.

During the experiments of LLE, we found that normalization after LLE reduction significantly improves the accuracy of SVM classifier since the LLE output feature are usually too small. The LLE results are on a scale of 10^{-4} to 10^{-6} in our experiments and without normalization, the SVM classifier predicted all the data into one class. This may due to the heterogeneity between dimensions, the limitation of float precision or algorithm implementation of `libsvm` itself.

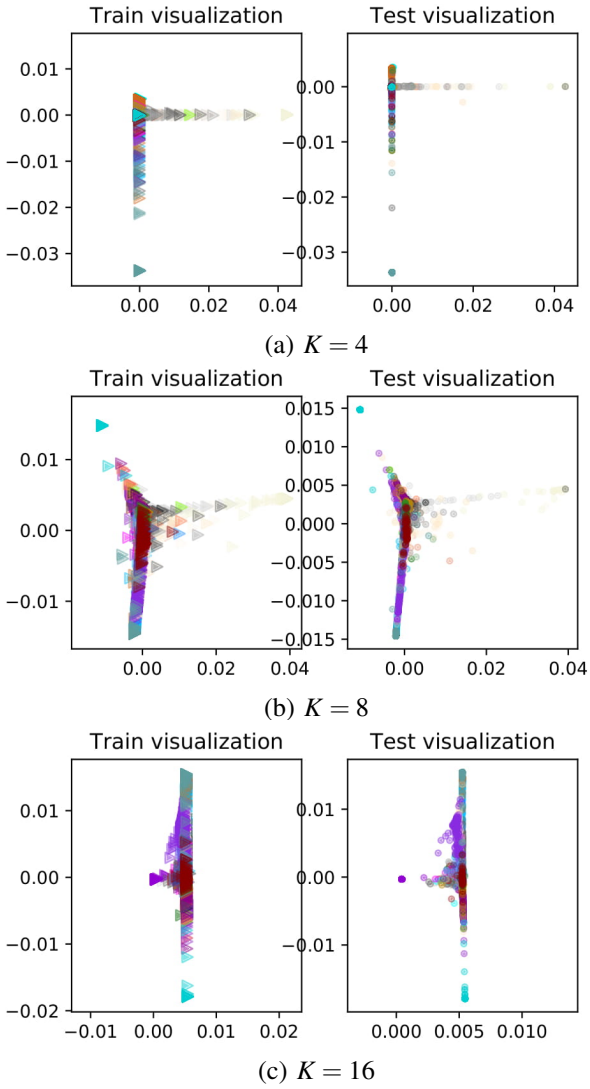
The good performance of LLE is also surprising because the input data are deep learning features and are supposed to be randomly distributed in an area and do not have manifold structure. A reasonable explanation is that in high dimensional space, data are mainly distributed at the surface of the area (proved at the class) and can be approximately regarded as manifold.

H. MDS

Due to the same reason of memory limitation, MDS was only implemented for dimensionality reduction of 2 dim. It turned out to be ineffective and inaccurate on test set. The results are shown in Table VII.

Acc. Method / Dim.	MDS
Baseline	**93.05%**
2	4%

TABLE VII
CLASSIFICATION RESULT WITH MDS



(a) $K = 4$

(b) $K = 8$

(c) $K = 16$

Fig. 7. LLE 2D Visualization at different K

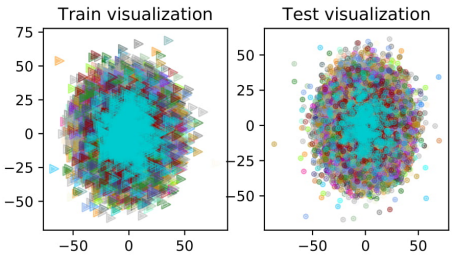


Fig. 8. MDS 2D Visualization

We can see the poor performance of MDS for this task. subsection III-C have shown that even simply selecting two features can reach an accuracy of 9.82%, which is more than twice of that of MDS.
The occurrence of these situations has a certain relationship with the algorithms themselves. The reasons of the bad performance are two-fold. It is mainly due to the low dimensionality, since less key information is contained in the feature. But what key information was lost? Considering

that MDS's core idea is to preserve distance information during the transform, we can tell that the high-dimensional distance information cannot be modeled well in low-dimensional space through MDS. It can be illustrated in Figure 9. In high-dimension space, there can be lots of points that are pairwise equidistant while separable, however, when transformed to lower-dimension space, their separability may not be preserved. Figure 8 also supports our inference, as we can see that the data points were totally mixed up.

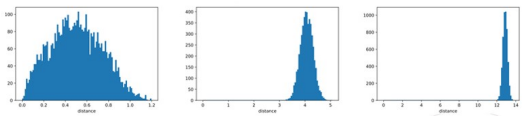


Fig. 9. The histograms of pairwise distances between $n = 100$ points sampled uniformly in the hypercube $[0,1]^d$

I. VAE

We implemented VAE with parameters, in which there were $x_{dim} + y_{dim}$ hidden layer neurons, where x_{dim} and y_{dim} were the number of neurons in input layer and output layer, and learning rate was set to 0.0001.

The visualization of 2-dimensional reduction result is shown in Figure 10, and all results are shown in Table VIII.

Dimension	Accuracy	Accuracy (Reconstruction)
Baseline	93.05%	
2	65.09%	62.78%
3	72.95%	70.15%
10	85.28%	**75.20%**
50	**87.97%**	74.22%
100	87.36%	73.25%
200	85.79%	69.80%
500	84.27%	61.13%
750	83.23%	46.96%
1000	75.34%	40.59%
1500	75.30%	39.80%
2000	78.16%	33.98%

TABLE VIII
CLASSIFICATION RESULT WITH VAE

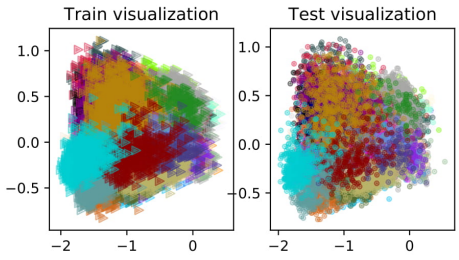


Fig. 10. VAE 2D visualization

From the result of VAE reduction and reconstruction, we come to a conclusion that, for this data set, dimension around 50 performed better than any other dimensions in our experiment. And generally, the reconstruction results were worse than reduction results, which we think the random sample procedure might be the reason. It could be caused by the distribution of the data set. We supposed the data set was composed of 50 Gaussian distributions, and we did some further study on this issue in subsection IV-C. Another observation from the result is that the accuracy after reconstruction is much lower than that before. It could be caused by that fact that in the decoder part of VAE neuron network, more information of distribution is lost.

J. Comprehensive Comparison

We illustrate the performance of all methods used in Figure 11.

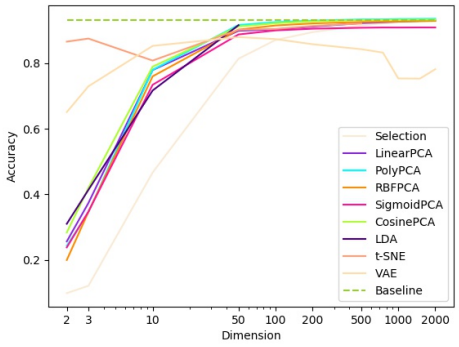


Fig. 11. Performance curve of all the methods used except MDS and LLE

Generally, three types of dimensionality reduction method were all proved effective by the experiments.

The projection and selection methods had similar performance growth as the dimensionality grew, but most of these methods began to work until reducing the data to about 500 dimensions. Besides the algorithms own insufficiency, the crowding problem occurring in low-dimension situation is also an important reason. The crowding problem is introduced by Hinton et al. in [5]. For example, it is possible to have 11 data points that are mutually equidistant in a ten-dimensional manifold but it is not possible to model this faithfully in a two-dimensional map. This problem results the inefficiency of classifying data projected to low-dimension space. LDA solved the problem by introducing the supervision of classification labels, trying to maximize inter-class variance, and achieved great performance in relatively low dimensionality. The feature learning methods, which were quite noteworthy, provided different solutions. LLE, trying to preserve the original manifold structure, achieved a feasible result. MDS tried to preserve distance information in the dimension transforming procedure. However, it failed in this task because the distance information itself could lose a lot during dimensionality reduction. t-SNE, aiming at preserving the probability relationship during the transform procedure, achieved to alleviate the problem with t distribution intro-

duced. And VAE, exploiting the neural network structure, presented competitive result from a generative perspective.

If taking efficiency into account, the projection and selection methods were all efficient with CPU, while most feature learning methods suffered from low efficiency with only CPU support. VAE, accelerated by GPU, is an exception among feature learning methods.

In a word, feature selection and feature projection provided both efficiency and effectiveness, but they also needed to keep more dimensions, where LDA was an exception. Most feature learning methods could provide good results with lower dimensionality if the data was suitable, but also needed more time. And some of them are not suitable for classification because the transform needs to be based on the whole data set. VAE, which benefits from neural network structure, could achieve both efficiency and effectiveness with low dimensionality.

IV. Further Study

In subsection III-H and subsection III-I, we were inspired to some ideas relative to these methods. Trying to evaluate them, here we presents some further study on these ideas.

A. MDS with Normalization

We've seen the poor performance of MDS in subsection III-H. Since it suffered from the loss of distance information, will the normalization before dimensionality reduction help? The inspiration was that normalization might make the distance distribution in high-dimension space smoother, thus it can be modeled better in low-dimension space.

Since MDS used Euclidean distance in our experiment, we performed L2 normalization before performing MDS on our data. The result were shown in Table IX.

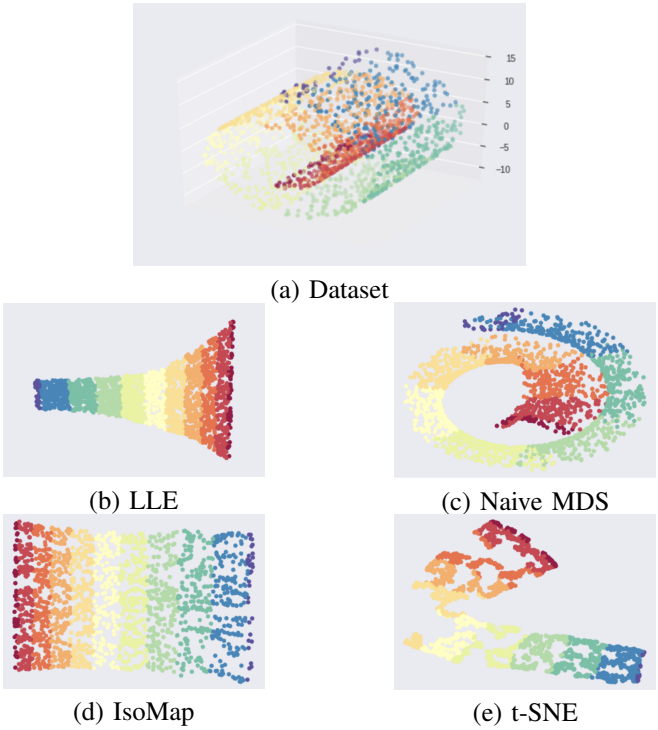
Dimension	Accuracy	Accuracy (Normalization)
Baseline	93.05%	
100	4%	19.39%

TABLE IX
Classification result with MDS before and after normalization

Obviously, the performance of MDS has been improved with normalization. Normalization can make the distance distribution smoother in high-dimension space, so that the distance information can be retained more effectively after dimensionality reduction, though it's still not worthy, especially considering the time and resource it costs.

B. Comparison Between Manifold Learning Algorithms

Since high dimensional data are more likely to distribute on a low dimensional manifold, manifold learning algorithms can be suitable on many circumstances rather than only on some specific situation like Swiss roll. Because of the limitation of computation resources, we can not implement these algorithms on the AwA2 data set. Without loss of generality, we evaluated some popular manifold methods with generated Swiss roll like data, including LLE, MDS(Nave MDS), IsoMap and t-SNE. The 3D samples and its dimensionality reduction results are shown in Figure 12



(a) Dataset

(b) LLE (c) Naive MDS

(d) IsoMap (e) t-SNE

Fig. 12. 3D samples and its dimensionality reduction results

These algorithms have their own characteristics. Naive MDS is not actually a manifold learning method so it didn't learn the manifold structure of the Swiss roll. But it managed to retain the cluters of the data by keeping the relative distance of the samples. LLE and IsoMap successfully exploit the manifold of the data set and stretch the Swiss roll into a 2D plane. But they are slightly different in their results because LLE only considers the local structure information while IsoMap calculates the global pair-wise distance. Thus the visualization of IsoMap is better than LLE. t-SNE try to keep the probabilistic distribution of the data, so it only partly keeps the manifold structure.

C. VAE

Originally in this model, we did not implement experiments on dimension from 20 to 90 except 50. We saw the VAE model reached the best performance when the dimension increased to 50, and then the performance became worse, except the final 2000 dimension. We assumed that the original data is composed of 50 different Gaussian distributions, because there were 50 different classes. Also, as the data set was 2048-dimensional, the 2000-dimensional performance got a bit better than lower dimensions. In this case, we made an assumption that the VAE model reached the best performance when the dimension is the original dimension or the number of clusters. We implemented more experiments between dimension 10 to 100 to confirm our assumption. The result is shown in Table X. The result did not conform to the assumption. However, we could see that

the accuracy with reduced data and reconstructed data did not converge to the same point, which might result from the over-fitting or under-fitting of our model. It could be also caused by other factors. The research on this phenomenon could be interesting.

Dimension	Accuracy	Accuracy (Reconstruction)
Baseline	93.05%	
10	85.28%	75.20%
20	**88.93%**	72.60%
30	88.53%	74.17%
40	88.28%	74.12%
50	87.97%	74.22%
60	87.67%	74.26%
70	87.72%	**74.60%**
80	87.28%	74.18%
90	86.99%	73.62%
100	87.36%	73.25%

TABLE X

CLASSIFICATION RESULT WITH VAE WHEN REDUCING DATA TO 10 TO 100 DIMENSIONS

V. CONCLUSIONS

In this project, we evaluated different dimensionality reduction methods on the given data and analyzed their performance. Finally, we try to give our own observation on how to choose dimensionality reduction methods considering different situations. For general purpose, we think PCA and Kernel PCA is feasible, while VAE is also competitive and efficient. For classification tasks, LDA might be a better choice. For data visualization, LLE, MDS and t-SNE might have their own advantages considering the data's manifold structure, while t-SNE might be more general. And for generative purpose, VAE is the only choice.

REFERENCES

[1] Yongqin Xian, Christoph H. Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning - A comprehensive evaluation of the good, the bad and the ugly. *CoRR*, abs/1707.00600, 2017.

[2] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572, 1901.

[3] Bernhard Schlkopf, Alexander Smola, and Klaus-Robert Mller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[4] Peter N. Belhumeur, João P. Hespanha, and David J. Kriegman. Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. In Bernard Buxton and Roberto Cipolla, editors, *Computer Vision — ECCV '96*, pages 43–58, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[5] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[6] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[7] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. cite arxiv:1312.6114.

[8] Geoffrey E Hinton and Sam T. Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 857–864. MIT Press, 2003.

[9] Hsuan-Tien Lin and Chih-Jen Lin. A study on sigmoid kernels for svm and the training of non-psd kernels by smo-type methods. *Neural Computation*, 06 2003.