SHANGHAI JIAO TONG UNIVERSITY
PRINCIPLES OF DATA SCIENCE
(CS245)

Project 1 Report

Dimensionality Reduction

Yuru Liu	518021910652
Luyuan Jin	518030910293
Xuecheng Li	518030910423

Group: 21

Date: April 9, 2021

1 Introduction

In this report, we mainly conducted three approaches to reduce feature dimensionality reduction: feature selection, feature projection, and feature learning. For each feature reduction approach, we apply several algorithms and compare their performances. Meanwhile, we utilize Support Vector Machine (SVM) to classify images as the baseline.

Our data is from the Animals with Attributes (AwA2) dataset, which consists of 37322 images of 2048-dim features belonging to 50 categories. For each category, we split 60% images into the training set and 40% into the test set.

Furthermore, we find out the optimal feature reduction method and the optimal dimensionality.

2 Support Vector Machine (SVM)

SVM is supervised learning model that analyze data for classification and regression analysis. The primal form of SVM with *soft margin* is:

$$\begin{aligned} \min_{\mathbf{w},b,\xi_i} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & y_i\left(\mathbf{w}^T\mathbf{x}_i + b\right) \geq 1 - \xi_i, \quad \forall i \\ & \xi_i \geq 0, \quad \forall i \end{aligned}$$

And if we apply kernel trick for SVM, the objective becomes:

$$\begin{aligned} \min_{\mathbf{w},b,\xi_i} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & y_i\left(\mathbf{w}^T\phi(\mathbf{x}_i) + b\right) \geq 1 - \xi_i, \quad \forall i \\ & \xi_i \geq 0, \quad \forall i \end{aligned}$$

We feed the training set without dimension reduction into SVM and test the model on testing set.The optimal hyper-parameters of linear SVM on AwA2 is down here.

C	Accuracy
0.0001	90.84%
0.001	92.83%
0.01	91.52%
0.1	92.40%
1	93.46%
10	93.42%

3 Feature Selection

3.1 Univariate Feature

Select-K-Best is one of the methods of univariate feature selection, which works by selecting the best features based on univariate statistical tests. It removes all but the k highest scoring features. There're different functions to score features:

3.1.1 χ^2 value

Testing χ-squared statistic can get χ^2 value. If a statistic is dependent upon another one, the chi-square value χ^2 between them will be very large. Then we select the features with larger chi-square value and remove other features.

We use it on different number of aim component [20, 50, 200, 400, 600, 800 ,1000 , 1200 , 1400, 1600, 1800. 2000],And we give our results on linear kernel SVM. We adopt classifiction accuracy as metric in this section.We summarize the experimental results of Select-K-Best in Tab. 1 and Fig. 1.

	Linear kernel
#comp=20	69.27%
#comp=50	85.01%
#comp=200	91.64%
#comp=400	92.85%
#comp=600	93.16%
#comp=800	93.15%
#comp=1000	93.19%
#comp=1400	93.29%
#comp=1600	93.26%
#comp=1800	93.27%
#comp=2000	93.22%

Table 1: Test accuracy of Select-K-Best χ^2 value

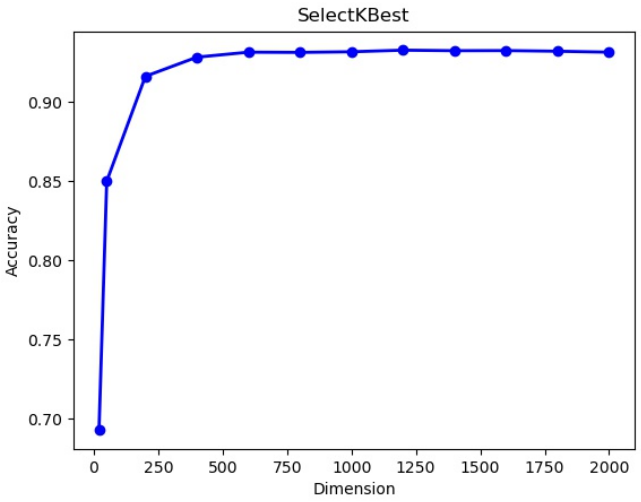


Figure 1: Test accuracy of Select-K-Best χ^2 value

3.1.2 ANOVA F value

F value = variance of the group means (Mean Square Between) / mean of the within group variances (Mean Squared Error). When the variance between the groups is much larger than the variance between the groups, that is, the larger the gap between the groups, the larger the F value.

Same as the previous section, We use it on different number of aim component [20, 50, 200, 400, 600, 800 ,1000 , 1200 , 1400, 1600, 1800. 2000],And we give our results on linear kernel SVM. We summarize the experimental results of Select-K-Best in Tab. 2 and Fig. 2.

	Linear kernel
#comp=20	71.16%
#comp=50	85.06%
#comp=200	91.45%
#comp=400	92.61%
#comp=600	93.16%
#comp=800	93.22%
#comp=1000	93.34%
#comp=1400	93.31%
#comp=1600	93.34%
#comp=1800	93.36%
#comp=2000	93.24%

Table 2: Test accuracy of Select-K-Best ANOVA-F value

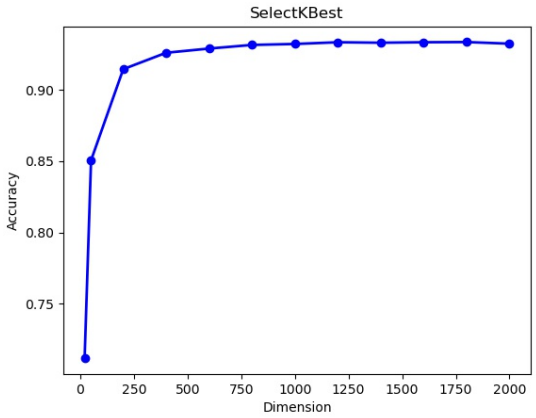


Figure 2: Test accuracy of Select-K-Best ANOVA-F value

3.2 Variance Threshold

Variance Threshold is a simple approach to feature selection. It removes all features whose variance doesn't meet some threshold. The principle is that features with small variance often contain less data information.

Same as the previous section, We use it on different number of aim component [0.1,0.3, 0.5, 0.7, 0.9, 1.1 , 1.3, 1.5 ,1.7, 1.9],And we give our results on linear kernel SVM. We summarize the experimental results of Select-K-Best in Tab. 3 and Fig. 3.

	Linear kernel
Threshold=0.1	93.19%
Threshold=0.3	93.14%
Threshold=0.5	93.02%
Threshold=0.7	92.54%
Threshold=0.9	92.25%
Threshold=1.1	91.45%
Threshold=1.3	90.61%
Threshold=1.5	89.63%
Threshold=1.7	88.41%
Threshold=1.9	85.72%

Table 3: Test accuracy of Variance Threshold method

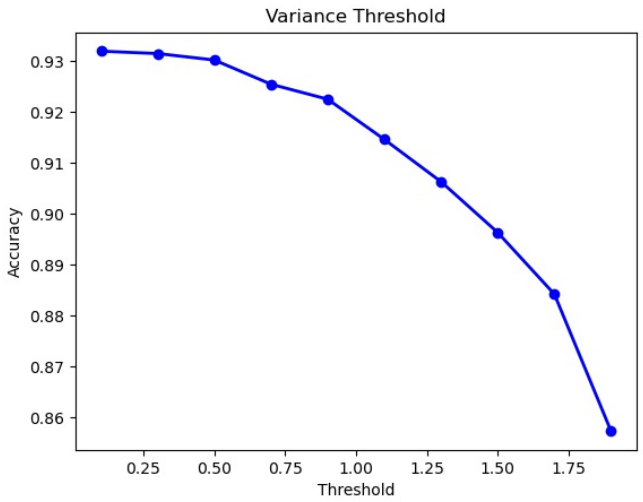


Figure 3: Test accuracy of Variance Threshold method

3.3 Tree-based selection

Tree-based selection is a kind of embedded method. Embedded methods can learn which features contribute most to the accuracy of the model. Then, we can pick up important features and remove irrelevant features. Here, tree-based estimator is a number of randomized decision trees. The estimator will fit those trees on various sub-samples of the dataset and use averaging to improve accuracy and control over-fitting problem.

Same as the previous section, We use it on different number of aim component [2, 4, 8, 16 ,32, 64, 128],And we give our results on linear kernel SVM. We summarize the experimental results of Select-K-Best in Tab. 3 and Fig. 3.

We can find that the best classification accuracy of tree-based selection is not as good as Select-K-Best and Variance Threshold. According to the decision tree, the reason may be that for data with inconsistent sample sizes in each category, the information gain is biased towards those features with more values.

	Linear kernel
#estimator=2	93.19%
#estimator=4	93.14%
#estimator=8	93.02%
#estimator=16	92.54%
#estimator=32	92.25%
#estimator=64	91.45%
#estimator=128	90.61%

Table 4: Test accuracy of Tree-based selection method

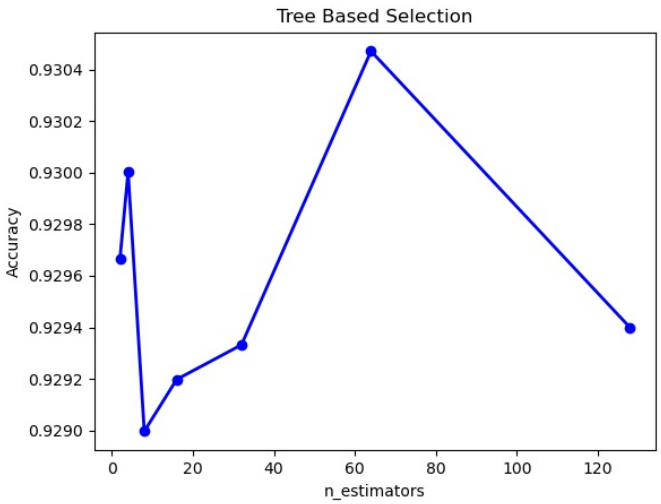


Figure 4: Test accuracy of Tree-based selection method

4 Feature Projection

4.1 Principal Component Analysis (PCA)

4.1.1 The principle of PCA

The *principal components* of a collection of points in a real p-space are a sequence of p direction vectors, where the i^{th} vector is the direction of a line that best fits the data while being orthogonal to the first $i-1$ vectors. PCA is commonly used for dimensionality reduction by projecting each data point onto the only first few principal components to obtain lower-dimensional data while preserving as much of the data's variation as possible. The idea of PCA is to find the principal directions that maximize variance of the data, as shown in Figure 7(a). Following this intuition, we can write the objective of PCA as:

$$\begin{aligned} \max_{\mathbf{v}} \quad & \mathbf{v}^T\mathbf{X}\mathbf{X}^T\mathbf{v} \\ \text{s.t.} \quad & \mathbf{v}^T\mathbf{v} = 1 \end{aligned}$$

where $\mathbf{v}$ is the principal component, $\mathbf{X}$ is the decentralized matrix of data, and $\frac{1}{n}\sum_{i=1}^{n}(\mathbf{v}^T\mathbf{x}_i)^2 = \frac{1}{n}\mathbf{v}^T\mathbf{X}\mathbf{X}^T\mathbf{v}$. Using Lagrangian, we can get:

$$\begin{aligned} & \mathcal{L}_{\mathbf{v}} = \mathbf{v}^T\mathbf{X}\mathbf{X}^T\mathbf{v} + \lambda\left(1 - \mathbf{v}^T\mathbf{v}\right) \\ \Rightarrow \quad & \frac{\partial \mathcal{L}_{\mathrm{v}}}{\partial \mathbf{v}} = \mathbf{X}\mathbf{X}^T\mathbf{v} - \lambda\mathbf{v} = \mathbf{0} \\ \Rightarrow \quad & \mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{v} \\ \Rightarrow \quad & \mathbf{v}^T\mathbf{X}\mathbf{X}^T\mathbf{v} = \mathbf{v}^T\lambda\mathbf{v} = \lambda \end{aligned}$$

We can see from above that the principal components $\mathbf{v}$ are eigenvectors of the data's covariance matrix $\mathbf{X}^T\mathbf{X}$, and thus the principal components are often computed by eigen decomposition of the data covariance matrix or singular value decomposition (SVD) of the data matrix. If we want to reduce the dimentionality to d', the eigenvectors corresponding to the d' largest eigenvalues are wanted. The algorithm of PCA is described in Algorithm 1.

Algorithm 1 Principal Component Analysis (PCA)

Input: The data $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\}$; the lower dimensionality d'.
Output: The projection matrix $\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}'_d)$.

1: Decentralize all samples: $\mathbf{x}_i \leftarrow \mathbf{x}_i - \frac{1}{m}\sum_{i=1}^{m}\mathbf{x}_i$
2: Compute the covariance matrix $\mathbf{X}\mathbf{X}^T$
3: Do eigen decomposition on the covariance matrix $\mathbf{X}\mathbf{X}^T$
4: Take the eigenvectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{d'}$ corresponding to the largest d' eigenvectors.

4.1.2 Kernel PCA

Kernel PCA (KPCA) is an extension of PCA using kernel methods. PCA uses linear projection function from the original feature space to a lower-dimisional space and can only be applied to linearly separable datasets, while KPCA can perform both linear and non-linear dimensionality reduction through the use of kernels, as shown in Figure 5.

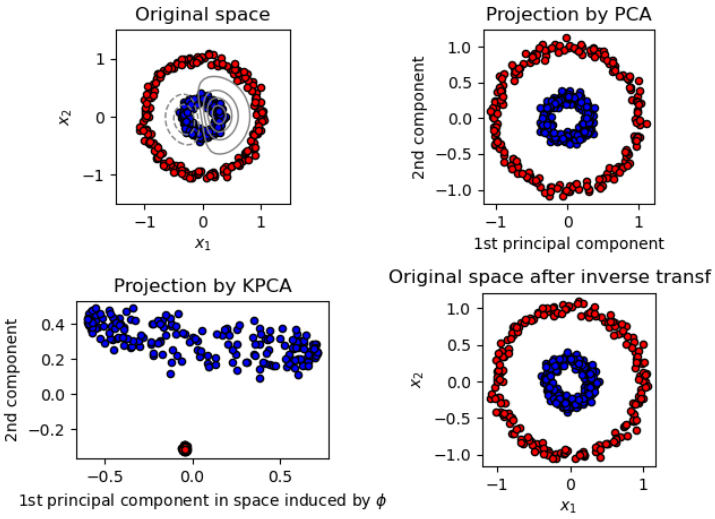


Figure 5: Illustration for Kernel PCA

The kernel function $K = k(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle = \Phi(\mathbf{x})^T \Phi(\mathbf{y})$, where$\Phi : \mathbb{R}^d \to \mathbb{R}^N (d \geq N)$ to project dataset onto a higher dimensional feature space where it is linearly separable. In comparison to PCA, we have:

$$\phi(\mathbf{X}) = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \ldots, \phi(\mathbf{x}_n)]$$
$$\mathbf{v} = \phi(\mathbf{X})\boldsymbol{\alpha}$$

And thus, the engenvalue problem becomes:

$$\begin{aligned} & \mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{v} \\ \Rightarrow \quad & \phi(\mathbf{X})\phi(\mathbf{X})^T\phi(\mathbf{X})\boldsymbol{\alpha} = \lambda\phi(\mathbf{X})\boldsymbol{\alpha} \\ \Rightarrow \quad & \phi(\mathbf{X})^T\phi(\mathbf{X})\phi(\mathbf{X})^T\phi(\mathbf{X})\boldsymbol{\alpha} = \lambda\phi(\mathbf{X})^T\phi(\mathbf{X})\boldsymbol{\alpha} \\ \Rightarrow \quad & \mathbf{K}\mathbf{K}\boldsymbol{\alpha} = \lambda\mathbf{K}\boldsymbol{\alpha} \\ \Rightarrow \quad & \mathbf{K}\boldsymbol{\alpha} = \lambda\boldsymbol{\alpha} \end{aligned}$$

Therefore, λ and $\boldsymbol{\alpha}$ are eigenvalues and eigenvectors of $\mathbf{K}$. The commonly used kernel functions include:

Linear kernel:	$K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$
Polynomial kernel:	$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y})^d$
Sigmoid kernel:	$K(\mathbf{x}, \mathbf{y}) = \tanh(\alpha\mathbf{x}^T\mathbf{y} + c)$
RBF kernel (Gaussian kernel):	$K(\mathbf{x}, \mathbf{y}) = \exp(-\frcc{\|\mathbf{x} - \mathbf{y}\|^2}{2\sigma^2})$
Cosine similarity kernel:	$K(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x}^T\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|}$

4.1.3 Experiment results

To run PCA on the AwA2 dataset, we make use of sklearn.decomposition.KernelPCA, and use two kinds of kernels: linear kernel and RBF kernel (Gaussian kernel). The classifier is implemented by sklearn.svm.SVC, where the regularization parameter C is set to be 1, and the kernel is consistent with the kernel for PCA. The kernel coefficient γ for RBF kernel is automatically set as 1/n_features. The results are shown in Table 5 and Figure 5. From the results we can see that the test accuracy increases as the intended number of components increases, although that means longer running time and larger memory space.

	Linear kernel	RBF kernel
#comp=2	25.07%	21.64%
#comp=4	49.17%	52.82%
#comp=8	74.56%	76.19%
#comp=16	85.10%	86.57%
#comp=32	89.60%	90.54%
#comp=64	90.25%	92.11%
#comp=128	90.41%	92.75%
#comp=256	90.79%	93.07%
#comp=512	91.48%	93.08%
#comp=1024	91.39%	92.61%
#comp=2048	91.15%	91.33%

Table 5: Test accuracy of KPCA

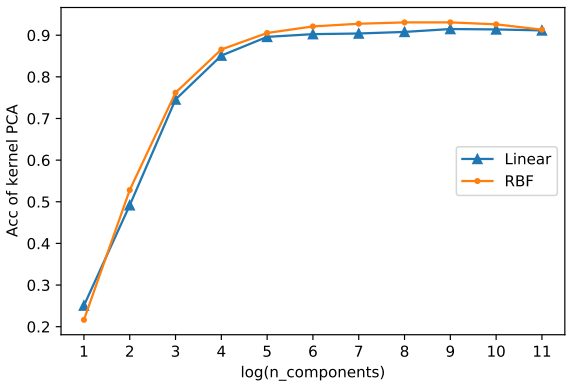


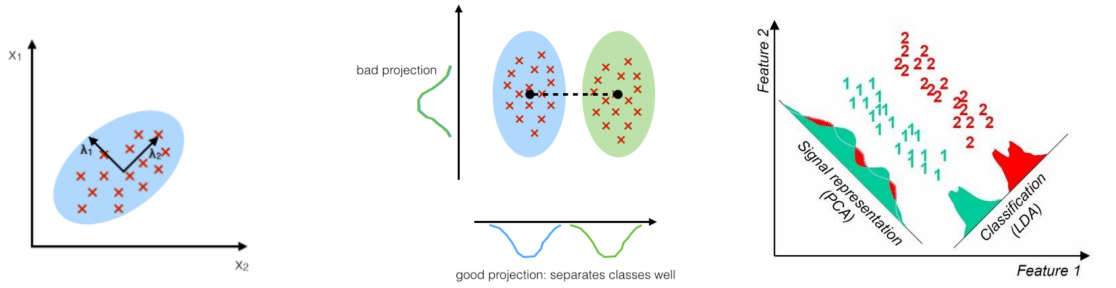
Figure 6: Test accuracy of KPCA

4.2 Linear Discriminant Analysis (LDA)

4.2.1 The principle of LDA

LDA is a generalization of Fisher's linear discriminant, and is commonly used as a dimensionality reduction technique in the pre-processing step for classification. The idea is to project

the given dataset onto a lower-dimensional space with good class-separability, so as to minimize the intra-class variance and maximize the inter-class variance, as shown in Figure 7(b). The task is to learn a linear function that projects $\mathbf{X}$ to $\mathbf{X}_i^T\mathbf{v}$, and the objective is to maximize $J(\mathbf{v}) = \sigma^2_{\text{between}}/\sigma^2_{within}$.



(a) PCA (b) LDA (c) PCA & LDA

Figure 7: Comparison of PCA and LDA

4.3 LDA for two classes

Suppose the two classes of observations have number of samples C_1, C_2, means $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$, and covariances $\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2$, then we can write the objective as:

$$\begin{aligned}
\max_{\mathbf{v}} J(\mathbf{v}), \quad J(\mathbf{v}) &= \frac{\left(\mathbf{v}^T\boldsymbol{\mu}_1 - \mathbf{v}^T\boldsymbol{\mu}_2\right)^2}{\boldsymbol{\sigma}_1^2 + \boldsymbol{\sigma}_2^2} \\
&= \frac{\left(\mathbf{v}^T\boldsymbol{\mu}_1 - \mathbf{v}^T\boldsymbol{\mu}_2\right)^2}{\sum_{i=1}^{C_1}\left(\mathbf{v}^T\mathbf{x}_{1,i} - \mathbf{v}^T\boldsymbol{\mu}_1\right)^2 + \sum_{i=1}^{C_2}\left(\mathbf{v}^T\mathbf{x}_{2,i} - \mathbf{v}^T\boldsymbol{\mu}_2\right)^2} \\
&= \frac{\mathbf{v}^T\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right)\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right)^T\mathbf{v}}{\mathbf{v}^T\left(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2\right)\mathbf{v}} \\
&= \frac{\mathbf{v}^T\mathbf{S}_B\mathbf{v}}{\mathbf{v}^T\mathbf{S}_W\mathbf{v}},
\end{aligned}$$

where S_B is the between-class scatter matrix, and S_W is the within-class scatter matrix:

$$\begin{aligned}
S_B &= \left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right)\left(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2\right)^T \\
S_W &= \boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2 = \sum_{i=1}^{C_1}(\mathbf{x}_{1,i} - \boldsymbol{\mu}_1)(\mathbf{x}_{1,i} - \boldsymbol{\mu}_1)^T + \sum_{i=1}^{C_2}(\mathbf{x}_{2,i} - \boldsymbol{\mu}_2)(\mathbf{x}_{2,i} - \boldsymbol{\mu}_2)^T
\end{aligned}$$

We require $\mathbf{v}^T\mathbf{S}_W\mathbf{v} = 1$, and then the objective is modified as:

$$\max_{\mathbf{v}} \mathbf{v}^T\mathbf{S}_B\mathbf{v}, \quad \text{s.t.} \quad \mathbf{v}^T\mathbf{S}_W\mathbf{v} = 1$$

Use Lagrange multipliers for this constraint problem, and we have:

$$\begin{aligned}
& \mathcal{L}_{\mathbf{v}} = \mathbf{v}^T\mathbf{S}_B\mathbf{v} - \lambda(\mathbf{v}^T\mathbf{S}_W\mathbf{v} - 1) \\
\Rightarrow \quad & \frac{\partial\mathcal{L}_{\mathbf{v}}}{\partial\mathbf{v}} = 2S_B\mathbf{v} - 2\lambda S_W\mathbf{v} = 0 \\
\Rightarrow \quad & S_B\mathbf{v} = \lambda S_W\mathbf{v} \\
\Rightarrow \quad & S_W^{-1}S_B\mathbf{v} = \lambda\mathbf{v}
\end{aligned}$$

Thus, $\mathbf{v}$ is the eigen vector of the matrix $S_W^{-1}S_B$. Since $S_B\mathbf{v} = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T\mathbf{v}$, the orientation of $S_B\mathbf{v}$ is always $\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2$, so the result is:

$$\mathbf{v} \parallel S_W^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$$

Also, we can derive that $S_B\mathbf{v} = \lambda S_W\mathbf{v}$, and thus

$$\mathcal{L}_\mathbf{v} = \frac{\mathbf{v}^T\mathbf{S}_B\mathbf{v}}{\mathbf{v}^T\mathbf{S}_W\mathbf{v}} = \frac{\mathbf{v}^T\lambda\mathbf{S}_W\mathbf{v}}{\mathbf{v}^T\mathbf{S}_W\mathbf{v}} = \lambda$$

So our objective becomes:

$$\max_\mathbf{v} \lambda, \quad \text{s.t.} \quad \mathbf{v}^T\mathbf{S}_W\mathbf{v} = 1$$

And our goal is to find the eigenvector $\mathbf{v}$ corresponding to the maximum eigenvalue λ of the matrix $S_W^{-1}S_B$.

4.3.1 Multiclass LDA

Suppose each class has a mean $\boldsymbol{\mu}_i$ and the same covariance $\boldsymbol{\Sigma}$, then the between-class scatter matrix $\boldsymbol{\Sigma}_B = \frac{1}{C}\sum_{i=1}^{C}(\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^T$ where $\boldsymbol{\mu}$ is the mean of the class means. Then the class separation in a direction $\mathbf{v}$ is given by $\mathcal{L}_\mathbf{v} = \frac{\mathbf{v}^T\boldsymbol{\Sigma}_B\mathbf{v}}{\mathbf{v}^T\boldsymbol{\Sigma}\,\mathbf{v}}$. And this means that when $\mathbf{v}$ is an eigenvector of $\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}_B$ the separation $\mathcal{L}_\mathbf{v}$ will be equal to the corresponding eigenvalue.

If $\boldsymbol{\Sigma}^{-1}\boldsymbol{\Sigma}_B$ is diagonalizable, the variability between features will be contained in the subspace spanned by the eigenvectors corresponding to the $C-1$ largest eigenvalues. These eigenvectors are primarily used in feature reduction, as in PCA. Since $rank(\boldsymbol{\Sigma}_B) \leq C - 1, rank(\boldsymbol{\Sigma}) \leq$ n_features, the number of components after LDA should be less than or equal to $\min(C-1,$ n_features$)$ for dimensionality reduction.

4.3.2 Experiment results

To run LDA, we make use of sklearn.discriminant_analysis.LinearDiscriminantAnalysis, and set the solver to be "svd", i.e. singular value decomposition, which does not compute the covariance matrix and therefore is suitable for data with a large number of features. We also use SVM for classification, and here two kinds of kernels are used, which are also linear kernel and RBF kernels. The regularization parameter for SVM is $C = 1$, and the kernel coefficient for RBF kernel is $\gamma =$ n_features.

	Linear SVM	RBF SVM
#comp=1	19.63%	19.45%
#comp=2	30.34%	30.52%
#comp=3	44.70%	44.85%
#comp=4	52.42%	52.42%
#comp=5	57.94%	58.15%
#comp=10	72.59%	72.96%
#comp=20	83.93%	84.47%
#comp=30	89.33%	89.76%
#comp=40	90.70%	91.55%
#comp=49	91.83%	92.86%

Table 6: Test accuracy of LDA

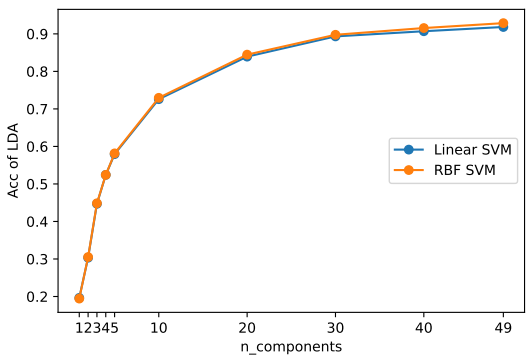


Figure 8: Test accuracy of LDA

5 Feature Learning

5.1 t-Distribution Stochastic Neighbor Embedding (t-SNE)

5.1.1 Background Theory

T-distributed stochastic neighbor embedding (t-SNE) algorithm is utilized to reduce and visualize high-dimensional data. It is composed of two main stages:

First, construct similar probability distributions of high-dimensional and low-dimensional data. High-dimensional data follow **Gaussian distribution**:

$$p_{j|i} = \frac{\exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\left\|\mathbf{x}_i - \mathbf{x}_k\right\|^2 / 2\sigma_i^2\right)} \tag{1}$$

where σ_i is up to the chosen perplexity.

Low-dimensional data follow **student t-distribution**:

$$q_{i|j} = \frac{\left(1 + \left\|\mathbf{y}_i - \mathbf{y}_j\right\|^2\right)^{-1}}{\sum_k \sum_{l \neq k} \left(1 + \left\|\mathbf{y}_k - \mathbf{y}_l\right\|^2\right)^{-1}} \tag{2}$$

Student t-distribution behaves better than normal distribution in terms of crowding problem because student t-distribution has thick tails.

Second, minimize the Kullback-Leibler divergence between the two probability distributions.

$$\mathrm{KL}(P\|Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{3}$$

The goal is to minimize the KL divergence, and the minimization of KL divergence is performed by gradient descent.

5.1.2 Comparison between t-SNE methods

There are 2 methods in t-SNE algorithm: 'exact' and 'barnes_hut', to compute the gradient. The 'exact' method computes the real gradient, with time complexity $O(DN^2)$, where D represents number of dimensions, and N represents number of samples. In our case, $N = 37322, D = 2048$. While the 'barnes_hut' method approximates the gradient with Barnes-Hut tree, with time complexity $O(DNlog(N))$. However, 'barnes_hut' algorithm only applies for $n\_component = 2$ and $n\_component = 3$.

$$\frac{Nlog(N)}{N^2} = \frac{log(37322)}{37322} = 0.0001 \tag{4}$$

In theory, Barnes-Hut algorithm takes 0.0001 time compared to the exact algorithm.

In practice, Barnes-Hut algorithm consumes much less time and RAM space. Barnes-Hut algorithm only takes one hour and 3.2G RAM, while the exact algorithm takes more than 10 hours and 11.5G RAM.

The precision of both barnes_hut and exact algorithms for $n\_component = 2$ and $n\_component = 3$ are shown in the following table.

Although Barnes-Hut improves time and space by much, in terms of precision, it has similar performances with the exact algorithm.

	t-SNE Accuracy	
	barnes_hut	exact
n_component = 2	86.81%	87.30%
n_component = 3	88.80%	88.70%

5.1.3 Comparison among PCA/LDA pre-preocessing and without pre-processing

It is worth trying to utilize PCA/LDA to reduce dimension and then apply t-SNE. Procedures are:

1. Apply PCA/LDA to reduce 2048-dim to N-dim (N is to be determined).
2. Apply t-SNE(barnes_hut) to reduce N-dim to 2-dim or 3-dim.
3. Apply SVC(rbf kernel) to classify and visualize.

When it comes to the choice of N, notice that:

1. For PCA, the accuracy increases as dimension increases, and converges at 64-dim, thus $N = 128$ is appropriate.
2. For LDA, the accuracy increases as dimension increases, and because dimensions are in the range [1,50). thus $N = 49$ is appropriate.

Results are shown in the following table and graph:

	Accuracy		
	PCA pre-processed	LDA pre-processed	without pre-processed
n_component = 2	86.43%	95.15%	86.81%
n_component = 3	88.47%	95.27%	88.80%

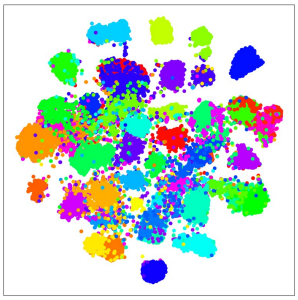


Figure 9: PCA pre-processed t-SNE (2D)

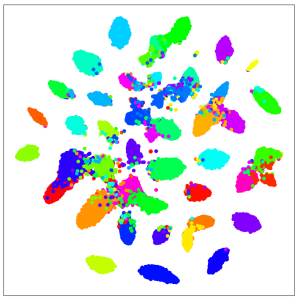


Figure 10: LDA pre-processed t-SNE (2D)

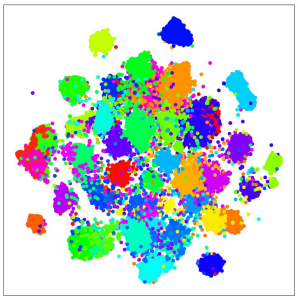


Figure 11: Without pre-processed t-SNE (2D)

Notice that: LDA pre-processing greatly improve the accuracy from 86% and 88% to 95% and 95% for #comp=2 and #comp=3 respectively.

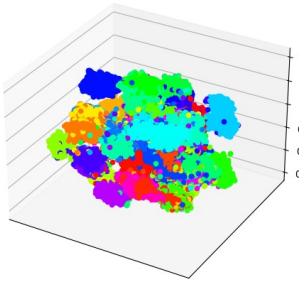


Figure 12: PCA pre-processed t-SNE (3D)

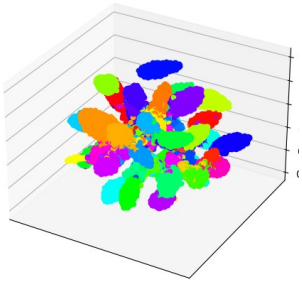


Figure 13: LDA pre-processed t-SNE (3D)

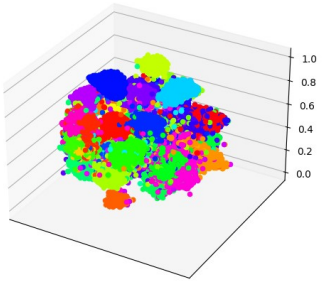


Figure 14: Without pre-processed t-SNE (3D)

5.2 Locally Linear Embedding (LLE)

5.2.1 Background Theory

Locally Linear Embedding (LLE) is another dimensionality reduction algorithm. It is composed of two main stages:

First, Find a set of the nearest neighbors of each point and compute the optimal set of weights that could best describe the certain point as a linear combination. The construction error is represented by the cost function:

$$\min E(W) = \sum_i \left| \mathbf{X}_i - \sum_j \mathbf{W}_{ij} \mathbf{X}_j \right|^2 \tag{5}$$

And the sum of every row of weight matrix W is equal to 1:

$$\sum_j \mathbf{W}_{ij} = 1 \tag{6}$$

Second, use an eigenvector-based technique to find the low-dimensional embedding points. Suppose the algorithm is to reduce D-dim to d-dim, in which $D >> d$; and suppose every X_i in D-dim maps to Y_i in d-dim respectively. The goal is to: with the weight matrix W unchanged, minimize the cost function:

$$\min C(Y) = \sum_i \left| \mathbf{Y}_i - \sum_j \mathbf{W}_{ij} \mathbf{Y}_j \right|^2 \tag{7}$$

5.2.2 Comparison among component and neighbor number choices

Procedures are:

1. Apply LLE to reduce 2048-dim to N-dim.
2. Apply SVC (rbf kernel) to classify (because the RAM space is not enough, we could not visualize the results).

Plot accuracy over #neighbor and log_2(#comp):

Note that:

	LLE Accuracy #neighbor=4	#neighbor=8	#neighbor=16	#neighbor=32	#neighbor=64
#comp=2	33.24%	40.61%	42.58%	47.94%	39.24%
#comp=4	49.76%	49.52%	58.05%	66.54%	59.07%
#comp=8	58.23%	67.28%	73.96%	75.42%	77.12%
#comp=16	65.26%	79.80%	82.61%	82.39%	83.99%
#comp=32	83.08%	84.95%	85.15%	86.56%	88.30%
#comp=64	85.85%	86.87%	88.43%	89.09%	90.27%
#comp=128	88.75%	89.84%	89.68%	90.25%	90.85%
#comp=256	89.39%	90.38%	90.58%	90.77%	91.10%

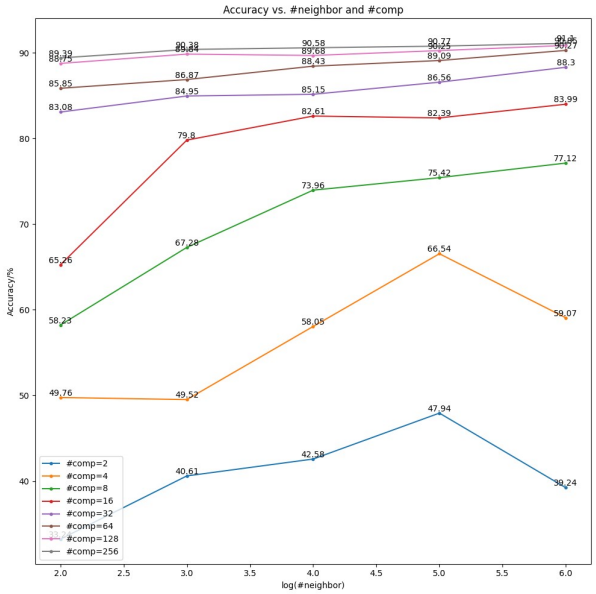


Figure 15: Accuracy vs. #neighbor and log_2(#comp)

- Both #neighbor and #comp follows the law of diminishing marginal utility, which means, when #neighbor and #comp increase to some extent, each additional unit augmentation input results in less and less output augmentation of accuracy.
- Accuracy converges to 91%, and the maximum accuracy happens when #neighbor $= 64$, #comp $= 256$.

5.2.3 Comparison among PCA/LDA pre-preocessing and without pre-processing

It is worth trying to utilize PCA/LDA to reduce dimension and then apply LLE. Procedures are:

1. Apply PCA/LDA to reduce 2048-dim to N-dim (N is to be determined).
2. Apply LLE to reduce N-dim to d-dim (d is to be determined).
3. Apply SVC(rbf kernel) to classify and visualize.

When it comes to the choice of N and d, notice that:

1. For PCA, the accuracy increases as dimension increases, and converges at 64-dim, thus $N = 128$ is appropriate.
2. For LDA, the accuracy increases as dimension increases, and because dimensions are in the range [1,50). thus $N = 49$ is appropriate.
3. Thus $d <= 48$

Results are shown in the following table and figure:

	PCA-LLE Accuracy #neighbor=4	#neighbor=8	#neighbor=16	#neighbor=32	#neighbor=64
#comp=2	31.43%	36.74%	48.29%	40.09%	35.81%
#comp=4	37.02%	57.04%	61.05%	57.10%	53.29%
#comp=8	44.25%	66.09%	74.76%	71.33%	71.54%
#comp=16	67.95%	80.77%	79.94%	78.46%	82.22%
#comp=32	82.51%	84.51%	84.12%	83.39%	86.21%

	LDA-LLE Accuracy #neighbor=4	#neighbor=8	#neighbor=16	#neighbor=32	#neighbor=64
#comp=2	37.8%	20.29%	17.35%	56.23%	56.39%
#comp=4	48.7%	21.73%	17.36%	75.56%	76.39%
#comp=8	70.89%	55.08%	76.16%	86.66%	87.87%
#comp=16	79.67%	88.55%	90.82%	91.31%	92.35%
#comp=32	88.74%	91.97%	94.12%	92.60%	94.20%

Note that:

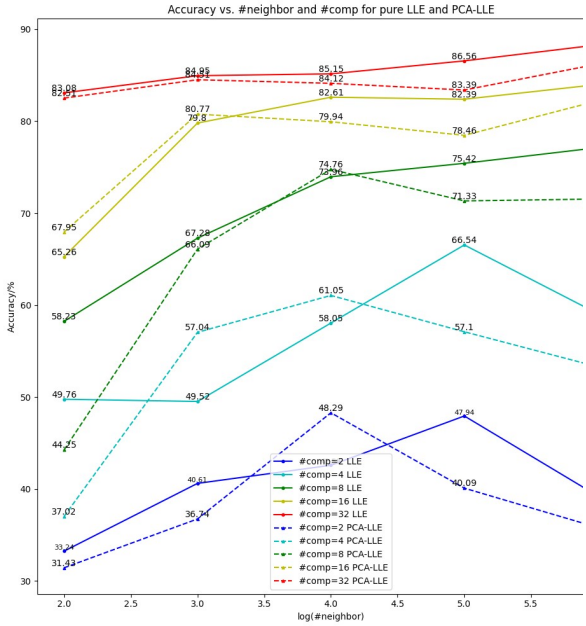


Figure 16: PCA-LLE with pure LLE

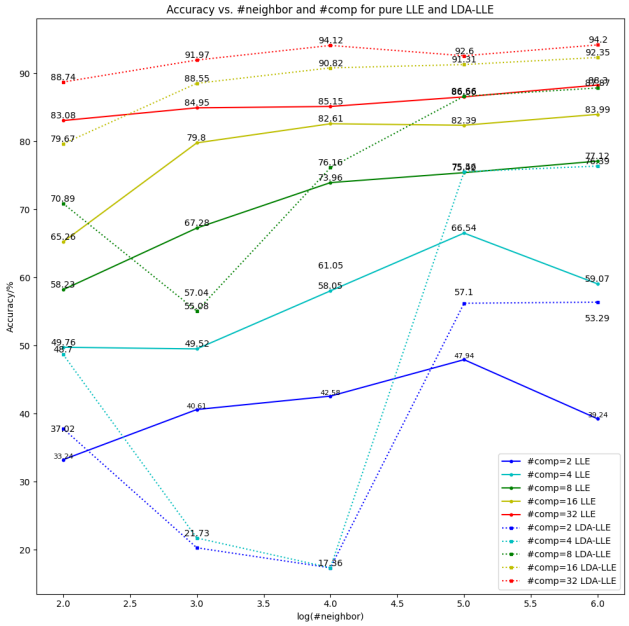


Figure 17: LDA-LLE with pure LLE

- Both PCA and LDA pre-processing behaves better when #neighbor is large. That's because LLE is an algorithm based on local points, and when #neighbor increases, it approaches global algorithm performances.
- LDA-LLE dramatically increase the accuracy performance from 88% (#comp=32, #neighbor=64) to 94% at the same circumstance.
- Like the pure LLE algorithm, both #neighbor and #comp follows the law of diminishing marginal utility.

6 Conclusion

The optimal accuracy and optimal intermediate/ final dimensionality for each method is shown in the following table: In conclusion, considering both accuracy and efficiency, it is recommended to adopt LDA and KPCA, which achieves 93.08% and 92.86% respectively.

If visualization is highlighted, it is recommended to adopt LDA-tSNE, using LDA to reduce dimensionality to a proper dimension and using t-SNE to classify and visualize.

	Optimal Accuracy	Intermediate Dim	Final Dim
Baseline	93.46%	\	\
χ^2	93.30%	\	\
ANOVA-F	93.36%	\	\
Tree-Based Selection	93.04%	\	\
Variance Threshold	93.19%	\	\
KPCA	93.08%	\	512
LDA	92.86%	\	49
tSNE	88.80%	\	3
PCA-tSNE	88.47%	128	3
LDA-tSNE	95.27%	49	3
LLE	91.10%	\	256
PCA-LLE	86.21%	128	32
LDA-LLE	94.20%	49	32