Project1 Report: Dimensionality Reduction

Hou Shengyuan 518021910604
Yan Binghao 518021910753
Wu Shiqu 518021910665

Abstract. Nowadays unstructured high-dimensional data like video, audio, text and images has become hot topics in mining research. However, high-dim data is often accompanied with the problem of substantial computation cost and low training efficiency. Otherwise high dimension brings about sparseness of data space representations, making it more likely to be overfitted. As a consequence, dimensionality reduction has to be applied to the preprocess of data. In this report, we try nine different dimensionality reduction methods, including selection by variance, Random Forest, PCA, kernel PCA, LDA, AE, VAE, t-SNE, Umap. Then we made overall comparisons between performance of various approaches and hyperparameters. The experiment on AwA2 dataset shows that LDA gets attains the most efficient performance with 0.93 accuracy and only 49 dimensions, while PCA with sigmoid kernel function reaches the best accuracy 0.935 but reduces dimension barely to 1024.

Keywords: Dimentionality Reduction, selection by variance, Random Forest, PCA, kernel PCA, LDA, AE, VAE, t-SNE, Umap

1 Introduction

With the rapid development of information technology, human beings are coming into "Big Data" era with quantities of data springing up, including structured data like relational table in the database, semi-structured data like HTML or XML file and unstructured data such as images, videos, audios and texts. Huge data capacity requests for economical and high-efficient mining techniques. Unfortunately, however, remaining algorithms and models often fail in the learning performance due to the high dimensionality of data, which makes the training process computationally-overweighted and low-efficient. Additionally, when feature vector representation becomes high, limited data sampling will give rise to sparse distribution in the feature space. These are often called "*curse of dimensionality*", which means the performance of classifier starts to descend after the number of dimensions excel a threshold, as showed in Fig. 1.

In this project, we take advantage of three kinds of dimensionality reduction methods to tackle this problem, including feature selection(**Selection By Variances**, **Random Forest** [1]), feature projection(**PCA** [2], **Kernel PCA** [3] , **LDA** [6], **AE**, **VAE**) and feature learning(**t-SNE** [4], **UMAP** [5]). The report is organized as follows: in Section 2, we introduce all methods mentioned above respectively. And next in Section 3, we do experiments on AwA2 dataset [8]. Then in Section 4 ,we compare and evaluate perfoemances on different methods and analyse the reasons behind phenomenons. Finally in Section 5, we make our conclusions and list our contributions.

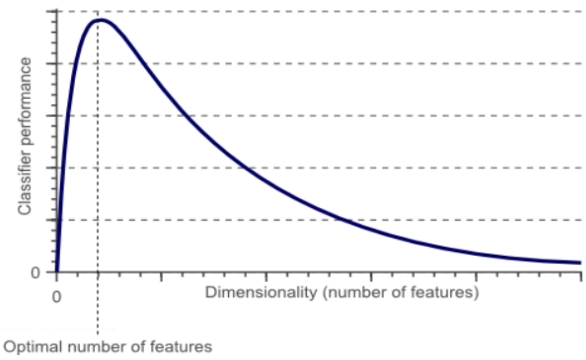

Fig. 1: Curse of Dimensionality

2 Related Works

2.1 Support Vector Machine (SVM)

Support Vector Machine is a binary classifier which partitions the data points by a hyperplane in the input vector space. The principle for partition is to maximize the distance between two data clusters, which is a convex quadratic optimization problem if data points are linearly separable.

$$\max_{\boldsymbol{w},b} \frac{2}{||\boldsymbol{w}||} \quad s.t. y_i(\boldsymbol{w^T x_i} + b) \geq 1 \quad (i = 1, 2, ..., m)$$

where w is the n-1 dim normal vector of the hyperplane, y_i is the label of i-th point and b is the bias of hyperplane. This is equivalent to solving

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2 \quad s.t.\ y_i(\boldsymbol{w^T x_i} + b) \geq 1 \quad (i = 1, 2, ..., m)$$

If $y_i(w^T x + b) \geq 1$ is no longer satisfied, we could add a penalty item ϵ_i for every data point to the right side of the inequality

$$y_i(\boldsymbol{w^T x} + b) \geq 1 - \xi_i (i = 1, 2, ..., m)$$

For compensation we add penalty item to objective function.

$$\min_{\boldsymbol{w},b} \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{m} \xi_i^2 \quad s.t.\ y_i(\boldsymbol{w^T x_i} + b) \geq 1 - \xi_i \quad (i = 1, 2, ..., m)$$

where C is the penalty parameter. C is larger means the penalty for wrong classification is larger. The object is to minimize the $\frac{1}{2}||w||^2$ which aims to enlarge the interval and minimize the penalty item which aims to reduce the wrong classified points. The objective function could be solved by using Lagrange Multiplier Method. Limited to the article length we do not spread it in detail.

2.2 Feature Selection by Variance

Feature selection is the most basic method to reduce dimension, it is intuitive and easy to understand. Generally speaking, according to the form of feature selection, there are three types of feature selection methods: 1. **Filter**, which scores each feature according to divergence or relevance, sets the threshold or the number of thresholds to be selected, and selects the feature . 2. **Wrapper**, according to the objective function (usually the prediction effect score), select several features at a time, or exclude several features. 3. **Embedded**, first use some machine learning algorithms and models for training, get the weight coefficient of each feature, and select the feature from large to small according to the coefficient. Similar to the Filter method, but through training to determine the pros and cons of features.

We choose the "Removing features with low variance" method for experimentation. Consider a variable in our dataset where all the observations have the same value, say 1. If we use this variable, it's obvious that the variable won't improve the model we will build because this variable has zero variance. So, we need to calculate the variance of each variable we are given then drop the variables having low variance as compared to other variables in our dataset. In short, variables with a low variance will not affect the model performance.

2.3 Random Forest

Random forest is an algorithm that uses multiple decision trees to train, classify and predict samples. Its essence belongs to a major branch of machine learning-ensemble learning (Ensemble Learning) method, which is mainly used in regression and classification scenarios. While classifying the data, the random forest algorithm can also give the importance score of each variable, and evaluate the role of each variable in the classification.

The random forest training algorithm applies the general techniques of bagging to tree learning. Given the training set $X = x_1, \cdots, x_n$ and target $Y = y_1, \cdots, y_n$, the bagging method is repeated (B times) from the training set Replace the sampling, and then train the tree model on these samples. After the training, the prediction of the unknown sample x can be realized by averaging the predictions of all single regression trees on x. The mathematical expression is as follows:

$$\hat{f} = \frac{1}{B} \sum_{b=1}^{B} f_b(x')$$

This bagging method reduces the variance without increasing the bias, resulting in better performance. This means that even if the prediction of a single tree model is very sensitive to the noise of the training base, for multiple tree models, as long as the trees are not related, this situation will not occur. Simply training multiple tree models on the same data set will produce a strongly correlated tree model (or even the exact same tree model).

In addition, the standard deviation of the predictions of all single regression trees on x' can be used as an estimate of the uncertainty of the prediction:

$$\sigma = \sqrt{\frac{\sum_{b=1}^{B}(f_b(x') - \hat{f})^2}{B-1}}$$

2.4 Principle Component Analysis(PCA)

The principle component analysis aims to find new k directions and do linear transformation for all data points in order to maintain the variance structure as much as possible. These k directions are called *principal component*. In other words within original space points along some directions are centralized while others are not and therefore we could directly drop centralized ones. Given the data matrix $\boldsymbol{X_{d\times n}}$, where d is original dimension and n is the number of data points we get and assume that $\boldsymbol{x} \in R^d$ is random vector, the goal is to empirically find an orthogonal projection vector $\boldsymbol{u} \in R^d(\boldsymbol{u^T u} = 1)$ which maximizes the variance of data points after projection. Column vectors of $\boldsymbol{u}$ are new orthonormal basis vectors that represent the subspace.

$$\max_{\boldsymbol{u}} \ Var(\boldsymbol{u^T x})$$

So we get

$$\begin{aligned} Var(\boldsymbol{u^T x}) &= E[\boldsymbol{u^T x}(\boldsymbol{u^T x})^T] - E[\boldsymbol{u^T x}]E[(\boldsymbol{u^T x})^T] \\ &= \boldsymbol{u^T}(E[\boldsymbol{x x^T}] - E[\boldsymbol{x}]E[\boldsymbol{x^T}])\boldsymbol{u} \\ &= \boldsymbol{u^T}\Sigma\boldsymbol{u} \end{aligned}$$

Empirically, we could replace the expectation by the mean of sample data point vectors, and for formula simplification we could centralize all dimensions to 0 at first so that $\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x_i} = 0$ So we get

$$\begin{aligned} \sigma^2(\boldsymbol{u^T x}) &= \boldsymbol{u^T}(\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x_i x_i^T} - \frac{1}{n^2}\sum_{i=1}^{n}\boldsymbol{x_i}\sum_{i=1}^{n}\boldsymbol{x_i^T})\boldsymbol{u} \\ &= \frac{1}{n}\boldsymbol{u^T X X^T u} \end{aligned}$$

Therefore the object is actually.

$$\max_{\boldsymbol{u}} \ \boldsymbol{u^T X X^T u} \ \ s.t.\ \boldsymbol{u^T u} = 1$$

Next, use Lagrange multiplier to solve this optimization problem, define

$$L = \boldsymbol{u^T X X^T u} + \lambda(\boldsymbol{I} - \boldsymbol{u^T u})$$

$$\boldsymbol{X X^T u} = \lambda\boldsymbol{u}$$

$$\boldsymbol{u^T X X^T u} = \boldsymbol{u^T}\lambda\boldsymbol{u} = \lambda$$

Therefore, the optimal condition informs that $\boldsymbol{u}$ should be eigenvectors of $\boldsymbol{X X^T}$ and λ should be the corresponding eigenvalue. And since we want to maximize $\boldsymbol{u^T X X^T u}$ which is actually λ, we should select k largest eigenvalue of $\boldsymbol{X X^T}$ and their corresponding orthonormal eigenvectors as the columns of projection matrix $\boldsymbol{U}$.

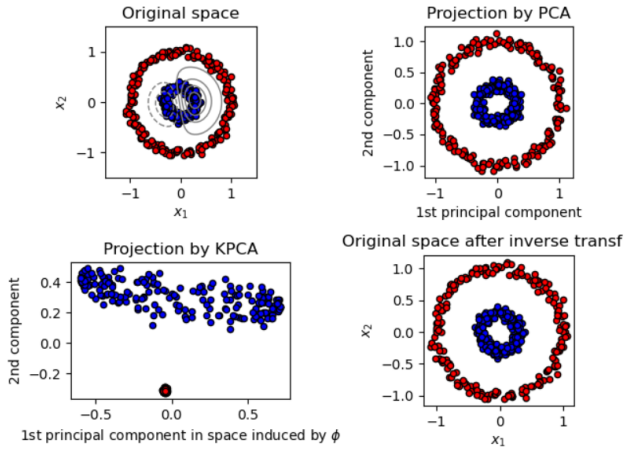

Fig. 2: Effects of Kernel PCA.

2.5 kernel PCA

Ordinary PCA aims to project high dimensional linear discriminative data into lower dimensional subspace while maintaining variance structure. However, for linear non-discriminative data, it could not project them into a discriminative one, but kernel PCA tackles this problem by implicitly mapping data points to high-dim space and reducing their dimensions with kernel technique(Fig. 2).

Essentially, kernel function could be expressed as the inner product of two high-dim vector .

$$K(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^T \phi(\boldsymbol{y})$$

ϕ is a function mapping data to high dimensional space which is often incomputable. Then we could use different kinds of kernel functions to express the inner product or similarity in high-dim space.

Given data matrix $\boldsymbol{X_{d\times n}} = [\boldsymbol{x_1}, \boldsymbol{x_2}, ..., \boldsymbol{x_n}]$, kernel PCA first map them to high dimensional space by ϕ to make them linear discriminative and then exert PCA on $\phi(\boldsymbol{x})$ to avoid curse of dimensionality. Thus the object is similarly.

$$\max_{\boldsymbol{u}} \boldsymbol{u}^T \phi(\boldsymbol{X}) \phi(\boldsymbol{X})^T \boldsymbol{u} \quad s.t.\ \boldsymbol{u}^T \boldsymbol{u} = 1$$

The optimal $\boldsymbol{u}$'s are thus k eigenvectors of $\phi(\boldsymbol{X})\phi(\boldsymbol{X})^T$ with topest eigenvalues. Do SVD on $\phi(\boldsymbol{X})$, which is $\boldsymbol{S\Sigma V^T}$, $\boldsymbol{S}$ and $\boldsymbol{V}$ are orthogonal. Since $\phi(\boldsymbol{X})\phi(\boldsymbol{X})^T = \boldsymbol{S\Sigma^2 S^T}$, where eigenvectors of $\phi(\boldsymbol{X})\phi(\boldsymbol{X})^T$ are just column vectors of $\boldsymbol{S}$, we could deduce that $\boldsymbol{U} = \boldsymbol{S} = \boldsymbol{\Sigma}^{-1}\boldsymbol{V}\phi(\boldsymbol{X})$. So we could gain projection results in a skillful way that

$$\boldsymbol{U}^T \phi(\boldsymbol{X}) = \boldsymbol{S}^T \phi(\boldsymbol{X}) = \boldsymbol{\Sigma V}^T$$

$K(\boldsymbol{X}, \boldsymbol{X}) = \phi(\boldsymbol{X})^T \phi(\boldsymbol{X}) = \boldsymbol{V\Sigma^2 V^T}$, thus we could diagonalize $K(\boldsymbol{X}, \boldsymbol{X})$ to get its eigenvalue matrix $\boldsymbol{\Sigma}$ and columns of eigenvectors $\boldsymbol{V}$ and finally calculate $\boldsymbol{U}^T\phi(\boldsymbol{X}) = \boldsymbol{\Sigma V}^T$ to project data onto lower dimensional space. In conlusion , steps are: (1) Given $\boldsymbol{X_{d\times n}}$ calculate $K(\boldsymbol{X}, \boldsymbol{X})$. (2) diagonalize $K(\boldsymbol{X}, \boldsymbol{X}) = \boldsymbol{V_{n\times n}\Sigma^2_{n\times n}V^T_{n\times n}}$. (3) select k eigenvector columns of $\boldsymbol{V}$ corresponding to k largest eigenvalues of $\boldsymbol{\Sigma^2}$. (4) calculate projection result $\boldsymbol{U}^T\phi(\boldsymbol{X}) = \boldsymbol{\Sigma V}^T$.

2.6 Linear Discriminant Analysis(LDA)

Compared to Principle Component Analysis, which aims to find the component axis with maximum covariance, linear discriminant analysis aims to find component axis which maximizes the class seperation. It maximizes variance S_b between diffent classes and minimizes variance S_w of data points within one class. So we calculate the ratio of two indexes. Assume that we have only two classes, which is called Fisher's Discriminant Analysis(FDA), the object is

$$\max_{\boldsymbol{v}} \frac{||\widetilde{\boldsymbol{\mu}}_{\boldsymbol{1}} - \widetilde{\boldsymbol{\mu}}_{\boldsymbol{2}}||^2}{\widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{1}} + \widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{2}}}$$

Since it's obvious that

$$\widetilde{\boldsymbol{\mu}} = \boldsymbol{v}^T\boldsymbol{\mu} \quad \widetilde{\boldsymbol{\Sigma}} = \boldsymbol{v}^T\boldsymbol{\Sigma}\boldsymbol{v}$$

Next we could simplify our objective function

$$\frac{||\widetilde{\boldsymbol{\mu}}_{\boldsymbol{1}} - \widetilde{\boldsymbol{\mu}}_{\boldsymbol{2}}||^2}{\widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{1}} + \widetilde{\boldsymbol{\Sigma}}_{\boldsymbol{2}}} = \frac{\boldsymbol{v}^T(\boldsymbol{\mu}_{\boldsymbol{1}} - \boldsymbol{\mu}_{\boldsymbol{2}})(\boldsymbol{\mu}_{\boldsymbol{1}} - \boldsymbol{\mu}_{\boldsymbol{2}})^T\boldsymbol{v}}{\boldsymbol{v}^T(\boldsymbol{\Sigma}_{\boldsymbol{1}} + \boldsymbol{\Sigma}_{\boldsymbol{2}})\boldsymbol{v}} = \frac{\boldsymbol{v}^T\boldsymbol{S}_{\boldsymbol{b}}\boldsymbol{v}}{\boldsymbol{v}^T\boldsymbol{S}_{\boldsymbol{w}}\boldsymbol{v}}$$

where $\boldsymbol{S_b} = (\boldsymbol{\mu_1} - \boldsymbol{\mu_2})(\boldsymbol{\mu_1} - \boldsymbol{\mu_2})^T$ which measures variances between classes and $\boldsymbol{S_w} = \boldsymbol{\Sigma_1} + \boldsymbol{\Sigma_2}$ which measures variances within classes. Next component vector $\boldsymbol{v}$ should be scaled to 1 so that $\boldsymbol{v}^T\boldsymbol{S_w}\boldsymbol{v} = 1$. Therefore, we could finally get objective.

$$\max_{\boldsymbol{v}} \boldsymbol{v}^T\boldsymbol{S_b}\boldsymbol{v} \quad s.t.\ \boldsymbol{v}^T\boldsymbol{S_w}\boldsymbol{v} = 1$$

Use Lagrange Multiplier

$$L(\boldsymbol{v}, \lambda) = \boldsymbol{v}^T\boldsymbol{S_b}\boldsymbol{v} - \lambda(\boldsymbol{v}^T\boldsymbol{S_w}\boldsymbol{v} - 1), \quad \boldsymbol{S_w^{-1}S_b}\boldsymbol{v} = \lambda\boldsymbol{v}$$

Since $r(\boldsymbol{S_w}) = d$ and $r(\boldsymbol{S_b}) = 1$ we could get $r(\boldsymbol{S_w^{-1}S_b}) = 1$, which means $\boldsymbol{S_w^{-1}S_b}$ has only one eigenvector and non-zero eigenvalue and this v is exactly the optimal solution. For multi-class situation,

$$\boldsymbol{S_w} = \sum_{i=1}^{k} \boldsymbol{\Sigma_i} \quad \boldsymbol{S_b} = \frac{1}{k}\sum_{i=1}^{k}(\boldsymbol{\mu_i} - \boldsymbol{\mu})(\boldsymbol{\mu_i} - \boldsymbol{\mu})^T$$

where k is the number of classes, $\boldsymbol{\mu}$ is the mean of all data points and $\boldsymbol{\mu_i}$ is the mean of i-th class data points. Since $r(\boldsymbol{S_b}) = k - 1$, $r(\boldsymbol{S_w^{-1}S_b}) = \min(k-1, d)$. Therefore , reduced dimensionality should not be more than $\min(k-1, d)$. Steps are: (1) For k-classes dataset, given $\boldsymbol{X_{d\times n}}$ calculate $\boldsymbol{S_w}$, $\boldsymbol{S_b}$ and $\boldsymbol{S_w^{-1}S_b}$. (2) diagonalize $\boldsymbol{S_w^{-1}S_b}$. (3) select t($t \leq \min(k-1, d)$) eigenvector columns to comprise projection matrix $\boldsymbol{V}$ corresponding to k-1 largest eigenvalues. (4) calculate projection result $\boldsymbol{V}^T\boldsymbol{X}$.

2.7 Automatic Encoder(AE)

AE is a type of Feedforward Neural Network mainly used for data dimensionality reduction or feature extraction. The *AE* framework is comprised of encoding process and decoding process, which could be implemented by multiplying linear transformation matrix or even adding non-linear activation layer.

As shown in Fig. 3, we set the input sample x to be mapped to the feature space z through encoder g, which is the encoding process; and then the abstract feature z is mapped back to the original space through decoder f to obtain the reconstructed sample $\hat{x}$, which is the decoding process. The optimization goal is to optimize both the encoder and the decoder by minimizing the reconstruction error.

$$g, f = argmin_{g,f} L\{x, (f \circ g)x\}$$

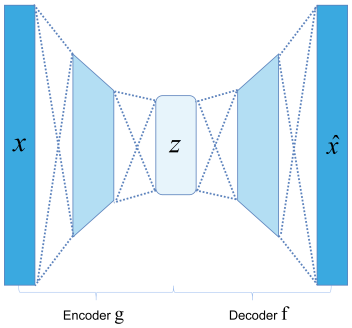

Fig. 3: Illustration of a typical Autoencoder.

2.8 Variational AutoEncoder(VAE)

The major difference between VAE and AE is: VAE no longer maps the input x to a fixed abstract feature z, but assumes that the abstract feature z of the sample x obeys the normal distribution of (μ, σ^2), and then, the abstract feature z is generated through the distribution. Finally, the output is obtained through the decoder based on z. Since the abstract feature z is generated from a normal distribution sampling, the encoder part of the VAE is a generative model, and then combined with the decoder to achieve reconstruction to ensure that the information is not lost. The model framework is shown in the Fig. 4

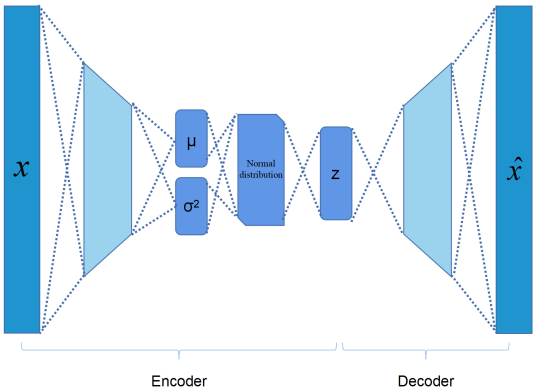

Fig. 4: Illustration of a Variational AutoEncoder.

2.9 t-SNE

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a dimensionality reduction technology used to represent high-dimensional data sets in a low-dimensional space to visualize them. Compared with other dimensionality reduction algorithms, t-SNE creates a reduced feature space, similar samples are modeled by nearby points.

The t-SNE algorithm is improved from SNE. SNE uses conditional probability to describe the similarity between two data. Assuming that x_i and x_j are two points in a high-dimensional space, then a Gaussian distribution with variance σ_i is constructed with the point x_i as the center, and $p_{j|i}$ is used to indicate that x_j is The probability of the neighborhood of x_i, if x_j is very close to x_i, then $p_{j|i}$ is very large, and $p_{j|i}$ is defined as follows:

$$p_{j|i} = \frac{exp(-||\boldsymbol{x_i} - \boldsymbol{x_j}||^2/(2\sigma_i^2))}{\sum_{k \neq i} exp(-||\boldsymbol{x_i} - \boldsymbol{x_k}||^2/(2\sigma_i^2))}$$

Considering that we only care about the similarity between different point pairs, so set $p_{i|i} = 0$, then such conditional probability can also be used to define the distance in the low-dimensional space, assuming that $\boldsymbol{x_i}, \boldsymbol{x_j}$ are mapped to the low-dimensional space corresponding to $\boldsymbol{y_i}$, $\boldsymbol{y_j}$. the conditional probability that $\boldsymbol{y_j}$ is neighborhood of $\boldsymbol{y_i}$ is $q_{j|i}$ (the variance in the low-dimensional space is directly set to $\sigma_i = \frac{1}{\sqrt{2}}$, which is convenient for calculation):

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

To balance the difference between conditional probability distributions, SNE uses K-L divergence (also called relative entropy), so the objective function is:

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

In t-SNE, after introducing the t distribution, in the low-dimensional space, use the t distribution with 1 degree of freedom to redefine $q_{j|i}$:

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

Thus.the cost function of t-SNE is:

$$L = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

Then we could use stochastic gradient descent to minimize the cost function.

2.10 UMAP

Unified Manifold Approximation and Projection (UMAP) is a dimension reduction technology that can retain as many local and global data structures as possible compared with t-SNE. This

method uses the concept of k-nearest neighbor and optimizes the results using stochastic gradient descent. It first calculates the distance between the points in high dimensional space, projects them onto the low dimensional space, and calculates the distance between points in this low dimensional space. It then uses Stochastic Gradient Descent to minimize the difference between these distances.

UMAP can ultimately be described in terms of, construction of, and operations on weighted graphs. For the distribution in the high-dimensional space, use the algorithm of finding the nearest neighbors to obtain the k nearest neighbor set $\boldsymbol{x_{i_1}}, ..., \boldsymbol{x_{i_k}}$ of each $\boldsymbol{x_i}$, and for each $\boldsymbol{x_i}$, we can determine ρ_i and σ_i:

$$\boldsymbol{\rho_i} = \min\{d(\boldsymbol{x_i}, \boldsymbol{x_{i_j}}) | 1 \leq j \leq k, d(\boldsymbol{x_i}, \boldsymbol{x_{i_j}}) > 0\}$$
$$\sum_{j=1}^{k} \exp(\frac{-\max(0, d(\boldsymbol{x_i}, \boldsymbol{x_{i_j}}) - \boldsymbol{\rho_i})}{\sigma_i}) = \log_2(k)$$

Then we get the calculation of the distribution:

$$p_{i|j} = \exp(\frac{-\max(0, d(\boldsymbol{x_i}, \boldsymbol{x_{i_j}}) - \boldsymbol{\rho_i})}{\sigma_i})$$
$$p_{ij} = p_{i|j} + p_{j|i} - p_{i|j}p_{j|i}$$
$$q_{ij} = (1 + a(\boldsymbol{y_i} - \boldsymbol{y_j})^{2b})^{-1}$$

Where $a \approx 1.93$ and $b \approx 0.79$ for default UMAP hyperparameters.
Finally get the cost function (cross entropy):

$$CE(X, Y) = \sum_i \sum_j [p_{ij}(X) \log(\frac{p_{ij}(X)}{q_{ij}(Y)} + (1 - p_{ij}(X)) \log(\frac{1 - p_{ij}(X)}{1 - q_{ij}(Y)})]$$

3 Experiment

3.1 Experimental Setup

In the experiment, We select the *Animals with attributes 2* dataset (*AwA2*) from [8]. The data is composed of 37322 images of 50 animals and 2048-dimensional learned representation features are pre-extracted using *Resnet101* from every image. For the dataset, we first split it into 60% training set and 40% testing set using stratified sampling with a solid random seed=1234. Then, we take the high-dim feature vectors as input and use basic linear support vector machine to measure the baseline effect of the classifier. Next, we use dimensionality reduction method to preprocess the input vector and explore its performance.

3.2 Linear SVM classifier

We take default linear kernel function and use grid search with 5-fold cross validation in the training set to find the best hyperparameter C, which functions as a penalty parameter. Since the testing accuracy should not be enclosed during the determination of any parameter, we do grid search and cross-validation only within training set. And finally we select the best C and measure final accuracy on testing set. The result is as follows in Table 1.

C	***Acc1***	***Acc2***	***Acc3***	***Acc4***	***Acc5***	***Mean Acc***
0.01	**0.924**	**0.926**	**0.924**	**0.928**	**0.925**	**0.925**
0.1	0.923	0.925	0.922	0.926	0.922	0.924
1	0.923	0.925	0.920	0.926	0.921	0.923
10	0.923	0.925	0.919	0.923	0.920	0.922
100	0.923	0.925	0.919	0.923	0.920	0.922

Table 1: Grid Search(5-fold) on C

According to the above table, we could get the best hyper-parameter **C=0.01** and perform it on the final test set. The experimental result shows that accuracy on test set reaches up to **0.931**. In the following experiment and latter project, we will fix **C** as 0.01.

For Linear SVM, there are two types of classification, one-vs-rest(ovr) and one-vs-one(ovo). Ovr means training the SVMs for every specific label as positive samples while regarding the rest of samples as negative labels. Ovo means for every pair of label class and their corresponding samples, we train a SVM classifier. However, the latter one requires $n(n-1)/2$ classifiers whose cost is unimaginable for $n = 50$, therefore we directly use 'ovr'.

3.3 Feature Selection By Variance

In this part, we use $VarianceThreshold()$ function of the *sklearn* library, where *threshold* is the threshold, and dimension is reduced by deleting parameters whose training variance is less than the threshold. Results are shown in the Table 2, from which we can see that the number of feature columns that satisfies $variance > threshold$ drops quickly as threshold increases. And there is a turning point where $threshold = 1.5, reduceddimension = 110, accuracy = 0.875$ that has a low dimension while still keeps the accuracy relatively high. So we choose dimension 110 as the optimal dimension.

threshold	**dim**	***Acc***
0.5	618	**0.896**
0.7	399	0.882
1.0	230	0.881
1.5	110	0.874
2.0	58	0.827

Table 2: ACC On Different Threshold

3.4 Random Forest

Random forest tells us the importance of each feature present in the data set. The experimental part is mainly to reduce the dimension by judging the importance of each function and retaining the most important functions. In Fig. 5, we listed the 50 most important features and quantified their importance. We found that when too many dimensions are selected, in other words, when some less important features are introduced, the accuracy shows a trend of rising fluctuations. These fluctuations also prove that the subsequent introduction of dimensions can no longer be used to bring significant optimization, as shown in the Fig. 6.

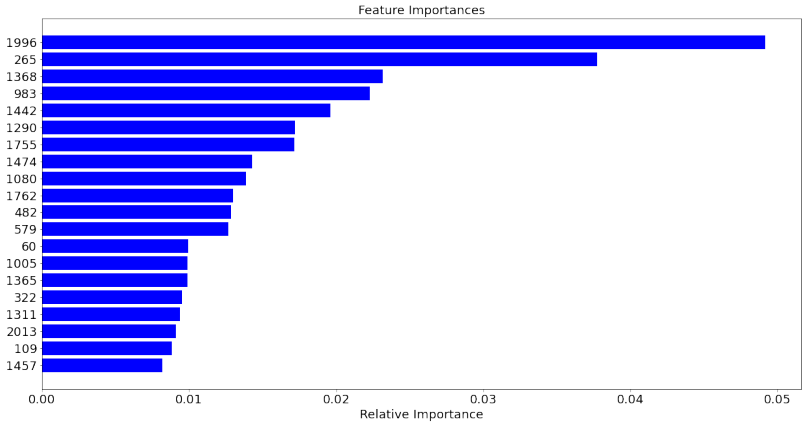

Fig. 5: Top 50 features.

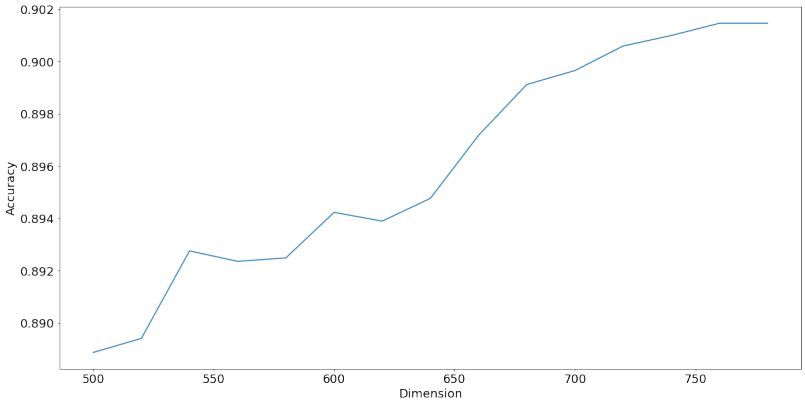

Fig. 6: Relationship between dimension and accuracy after 500 dimensions.

3.5 PCA and Kernel PCA

Different from selection by variance, PCA allows to maximize the variance by translating and rotating old basis to new directions. Since for PCA the only parameter is the goal dimensionality, we do experiments on the dimensionality set $\{2048, 1024, 512, 256, 128, 64, 32, 16, 8, 4, 2\}$, the results are shown in Table 3.

From the above experimental result, we could find that the accuracy starts to drop down dramatically once the dimensionality after reduction becomes lower than 256. By contrast, the accuracy appears to fluctuate around 0.93 and reaches to the top around 1000.

Unlike PCA which only allows linear transformation of old basis, it could not alter linear discriminability of data points. For linear non-discriminative data, such as thimble ring distribution, it's wiser to first map it to high dimensional space to become linearly-discriminative and then use PCA to drop dimensionality down as maintaining separability. This could be done implicitly by

dim	*Acc*
2	0.247
4	0.490
8	0.743
16	0.853
32	0.904
64	0.917
128	0.924
256	0.928
512	0.929
1024	**0.932**
2048	0.931

Table 3: ACC On Different Dimensionality

kernel function, such as

$$
\begin{aligned}
RbfKernel: &\ K(\boldsymbol{x},\boldsymbol{y}) = \exp\left(-\gamma||\boldsymbol{x}-\boldsymbol{y}||^2\right) \\
LinearKernel: &\ K(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{x}^T\boldsymbol{y} \\
PolynomialKernel: &\ K(\boldsymbol{x},\boldsymbol{y}) = (\gamma\boldsymbol{x}^T\boldsymbol{y}+c_0)^d \\
CosineKernel: &\ K(\boldsymbol{x},\boldsymbol{y}) = \frac{\boldsymbol{x}^T}{||\boldsymbol{x}||}\frac{\boldsymbol{y}}{||\boldsymbol{y}||} \\
SigmoidKernel: &\ K(\boldsymbol{x},\boldsymbol{y}) = \tanh(\gamma\boldsymbol{x}^T\boldsymbol{y}+c_0)
\end{aligned}
$$

Linear kernel is equivalent to ordinary PCA since it remains old basis in the original space. Rbf Kernel could be regarded as exponential-distributive distance and if $\gamma = 1/\sigma^2$, it's actually Gaussian kernel with variance σ^2. For polynomial kernel, one distinction is that it allows to consider the interaction between different features rather than just two vectors' similarity. Its parameter include the degree and bias term. Cosine kernel is actually the cosine of the angle between point vectors. Sigmoid kernel is often used as activate function.

In the experiment we take the parameters of linear SVM and set the kernel efficient γ, degree and c_0 as default value $1/n\_features$, 3 and 1. To speedup the training, we set $n\_jobs = -1$ to use all cores. The result is shown in Table 4 and Fig. 7.

It could be inspected that except for ordinary PCA, other kernel function appears to awfully perform. Empirically when num of features is lower than the num of samples, non-linear kernel function usually appears to be more superior than linear one. Therefore, we try to analyze the effect of different parameters for kernel function in further discussion.

3.6 LDA

Linear Discriminant Analysis takes a distinct strategy compared with maximum-variance feature selection or projection. It aims to find component axes that minimize the distance within one class and maximize the distance between classes.

Since num of component axes should not be larger than $\min(k-1, d)$. In the experiment, we select dimension set $\{4, 9, 14, 19, 24, 29, 34, 39, 44, 49\}$ and record the accuracy in Table 5.

dim	*Linear*	*Rbf*	*Polynomial*	*Cosine*	*Sigmoid*
2	0.247	0.057	0.118	0.077	0.044
4	0.490	0.065	0.255	0.126	0.044
8	0.743	0.150	0.463	0.305	0.044
16	0.853	0.296	0.656	0.449	0.044
32	0.904	0.391	0.784	0.560	0.044
64	0.917	0.459	0.835	0.622	0.044
128	0.924	0.480	0.852	0.640	0.044
256	0.928	0.482	0.857	0.646	0.044
512	0.929	**0.484**	0.861	**0.648**	0.044
1024	**0.932**	0.484	0.862	0.648	0.044
2048	0.931	0.484	**0.863**	0.648	0.044

Table 4: ACC On Different Dimensionality and Kernel Function(C=0.01)

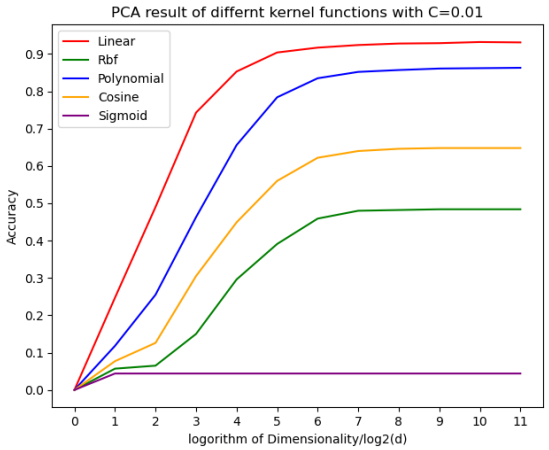

Fig. 7: Performance on Different Kernels when C=0.01 and γ =1/n_features

dim	*Acc*
4	0.521
9	0.689
14	0.790
19	0.832
24	0.876
29	0.898
34	0.906
39	0.915
44	0.924
49	**0.930**

Table 5: ACC On Different Dimensionality

Although when dimensionality reaches up to border(49) LDA gets its best result **0.930** which is slightly lower than ordinary PCA, it enjoys high training efficiency due to its significantly low dimension, while ordinary PCA has to pull its dimension higher than 500. Therefore, LDA generates a better result.

It's interesting to compare the reduction effect with respect to PCA(Fig. 8) and LDA(Fig. 9) on dimension 2, although they do not perform well on 2-dim plane. It's obvious that LDA tries to seperate some classes from the samples to form different clusters, which is identical to maximizing distances between classes and minimizing distances within classes. On the contrary, PCA regards every data point as identical instance without considering class distributions, and therefore considers no internal regular patterns, appearing to be messy and less accurate on 2-dim.

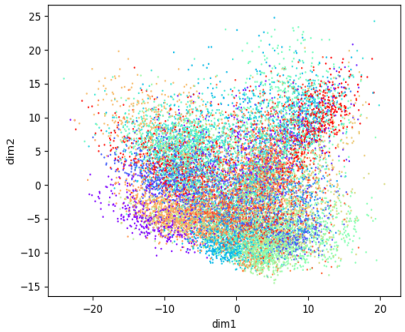

Fig. 8: PCA effect on test data

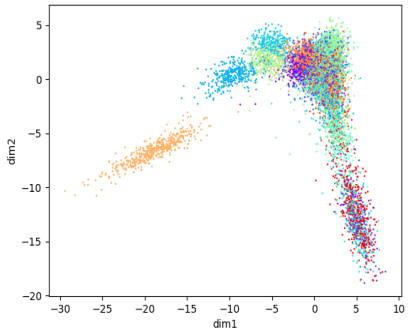

Fig. 9: LDA effect on test data

3.7 UMAP and t-SNE

First, we conducted an experiment on t-SNE, using the $TSNE()$ function of the *sklearn* library. Limited to the dimension selection after dimensionality reduction of this function, only $2, 3$ can be selected, and we found that adjusting the remaining parameters can not achieve better results, as shown in Table 6. On the other hand, the Umap method itself is an optimization of t-SNE, and we will pay more attention to the former in our experiments.

Perplexity	dimension=2	dimension=3
20	0.416	0.636
25	0.431	0.638
30	0.435	0.**639**
40	**0.439**	0.631
50	0.438	0.634

Table 6: ACC On Different Parameters

Then, we are experimenting with the official UMAP library. Therefore, the experiment will mainly focus on the parameters of this function, which may have a significant impact on the embeddings generated. Three main aspects will be introduced:

1) *n_neighbors*: This parameter controls how UMAP balances the local structure and global structure in the data, as shown in Fig. 10 and Fig. 11. It does this by limiting the size of the local neighbor UMAP.

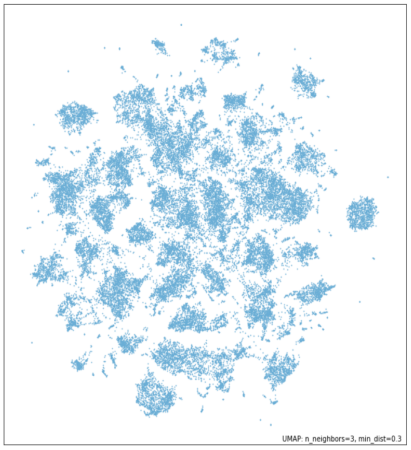

Fig. 10: (a)

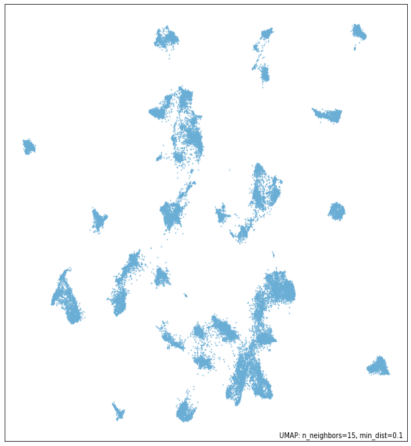

Fig. 11: (b)

2) *min_dist*: The *min_dist* parameter controls how tightly UMAP is allowed to pack points together. It, quite literally, provides the minimum distance apart that points are allowed to be in the low dimensional representation.
3) *n_components*: As is standard for many scikit-learn dimension reduction algorithms UMAP provides a *n_components* parameter option that allows the user to determine the dimension of the reduced dimension space we will be embedding the data into.

For the sake of logical order, we first list results in Table 7.

Comparing the experimental group $(1, 2)$, we find that the accuracy of *n_neighbors* is higher when the value of *n_neighbors* is smaller in the high-dimensional ($dimension = 200$). However, comparing the experimental group $(4-8)$, in the case of low dimension, we found that as *n_neighbors* increases, the accuracy rate increases. According to the definition, we understand that the smaller the *n_neighbors*, the more UMAP focuses on local features, and the larger it is, the more it focuses on global features. This means that a lower value of *n_neighbors* will force UMAP to focus on a very local structure (which may damage the overall structure), while a larger value will force UMAP to look at each point when estimating the manifold structure of the data. Larger neighborhoods, and thus lose fine detail structure, in order to obtain a wider range of data, so a balance needs to be achieved. It can be seen from the experimental results that the number of *n_neighbors* increases, and the global information can be better considered when the amount of data is large, so the accuracy rate gradually increases, but the effect is better since $n\_neighbors = 30$ The rise is not obvious enough, and the time and resources spent on calculation are relatively large at this time. In summary, *n_neighbors* is about 30.

The *min_dist* parameter determiness how closely the points packed by UMAP. Literally, it provides the minimum distance allowed for a point in low-dimensional space. This means that a lower value of *min_dist* will result in a clump (clump) embedding, which is beneficial for UMAP to focus on clustering or more refined topology. A larger value of *min_dist* will prevent UMAP from packing points together, and will focus on preserving the broad topology. It can be seen that the increase of *min_dist* leads to a significant increase in performance, indicating that for a data set with a large number of features and a large amount of samples, retaining the general and global topology will retain more features, which is accurate for SVM The rate increase is more effective. Through the comparison of the experimental group $(9 - 15)$, we get the better value of *min_dist*.

Times	n_neighbors	min_dist	accuracy	dimension
1	5	0.3	0.79	200
2	15	0.3	0.39	200
3	15	0.3	0.63	100
4	3	0.3	0.54	50
5	15	0.3	0.56	50
6	30	0.3	0.67	50
7	50	0.3	0.673	50
8	70	0.3	0.676	50
9	100	0.3	0.665	50
10	100	0.7	0.751	50
11	100	0.9	0.654	50
12	100	0.8	0.813	50
13	100	0.85	0.813	50
14	100	0.82	**0.857**	50
15	100	0.83	0.82	50
16	30	0.82	0.753	70
17	30	0.82	0.816	60
18	30	0.82	0.749	40
19	30	0.82	0.807	50
20	30	0.82	0.8	55
21	30	0.82	0.836	50

Table 7: ACC On Different Parameters

Finally, it is about the adjustment of dimensions and *randomseed*, and the best result is 0.857.

3.8 AE and VAE

In our experiments, we found that AE performs poorly when dimensionality is reduced to a lower dimension (as dimensionality is below 50, accuracy is below 80%); but when we relax the requirements for dimensionality reduction, AE's performance in high dimensions is better than VAE (as dimensionality is reduced to more than 512 dimensions, accuracy rate is above 90%, and performance of AE continues to rise with the increase of dimensionality, this is actually the expense of dimensionality reduction). Considering that our core purpose is dimensionality reduction, only the experimental results of VAE are shown here. Reasons for obtaining such experimental results will be discussed in the conclusion section. The VAE method mainly adjusts the dimension of Encoder after dimension reduction. The experimental results are shown in Table 8.

dim	***Acc***
2	0.290
10	0.820
20	0.825
30	**0.839**
50	0.805

Table 8: ACC On Different Dimensionality

It can be seen from the above experimental results that once the reduced dimension is less than 10, the accuracy loss will be greater. In contrast, when the target dimension is $[30, 50]$, the accuracy is still within an acceptable range.

But as the reduced dimension increases, the training time and memory space increase dramatically, and we can see that when dimension is larger than 50 (here we just display the most representative ones), the performance begin to decrease. The reason is that when the dimension is relatively high, it's hard for us to design a suitable cost function which has a balanced reconstruction loss and KL loss. Another reason is that due to the curse of dimensionality, it's hard to get an appropriate sampling function which can sample from the distribution uniformly. So this is why we think the best dimension for VAE method is around 20.

4 Further Discussion

In this section, we analyze effects for different hyperparameters in kernel PCA with respect tokernel function and then propose some improvements. Then we mention some drawbacks of PCA.

4.1 Why kernels attain different performance

For cosine kernel, after inner product and normalization points with low angles but high Euclidean distance will collapse dramatically, which makes points among different classes intersect with each other. The solution to this is to try to enlarge parameter C to prevent from underfitting.

For Sigmoid and rbf kernel, gradient vanishing gives rise to the disappearance of some high distance features of some space points. The solution is to alter rbf to t-distribution and decrease γ for sigmoid kernel. Collapse of high-dimensional data point is shown in Fig. 12 and Fig. 13

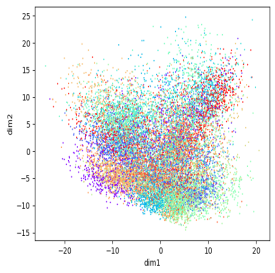

Fig. 12: 2-dim PCA

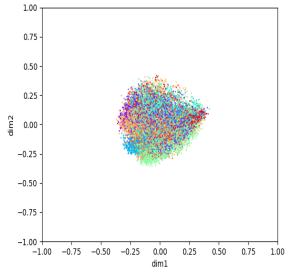

Fig. 13: 2-dim rbf

When C is smaller, svm gets more allergic to the degree of balance . If the data samples bear a serious classing imbalance, F1 score will dop down dramatically, especially when distances are close.

Therefore, it's wise to increase SVM's C in kernel space. When γ is larger, points with large distance originally is more likely to gather in high dimensional space. Therefore, it's wise to drop down γ to alleviate this symptom.

In short, it's better for us to tune C to be larger and γ to be smaller and we set $C = 10$ and $\gamma = 0.01$ for 4 non-linear kernel functions. Updated results are shown in Table 9 and Fig. 14.

Obviously after parameter fune-tuning, performance of some non-linear kernel PCAs finally excels ordinary one, and performance of most kernel functions are close to ordinary one. And it's also the best performance up to now in this report.

dim	*Linear*	*Rbf*	*Polynomial*	*Cosine*	*Sigmoid*
2	0.247	0.196	0.228	0.283	0.262
4	0.490	0.517	0.453	0.551	0.512
8	0.743	0.740	0.738	0.758	0.740
16	0.853	0.842	0.850	0.856	0.850
32	0.904	0.895	0.892	0.904	0.903
64	0.917	0.916	0.911	0.919	0.922
128	0.924	0.922	0.914	0.926	0.928
256	0.928	0.925	0.920	0.930	0.932
512	0.929	0.929	**0.934**	**0.934**	0.934
1024	**0.932**	0.930	0.923	0.934	**0.935**
2048	0.931	**0.932**	0.927	0.933	0.935

Table 9: ACC On Different Dimensionality and Kernel Function(C=10, $\gamma = 0.001$, C=0.01 for linear)

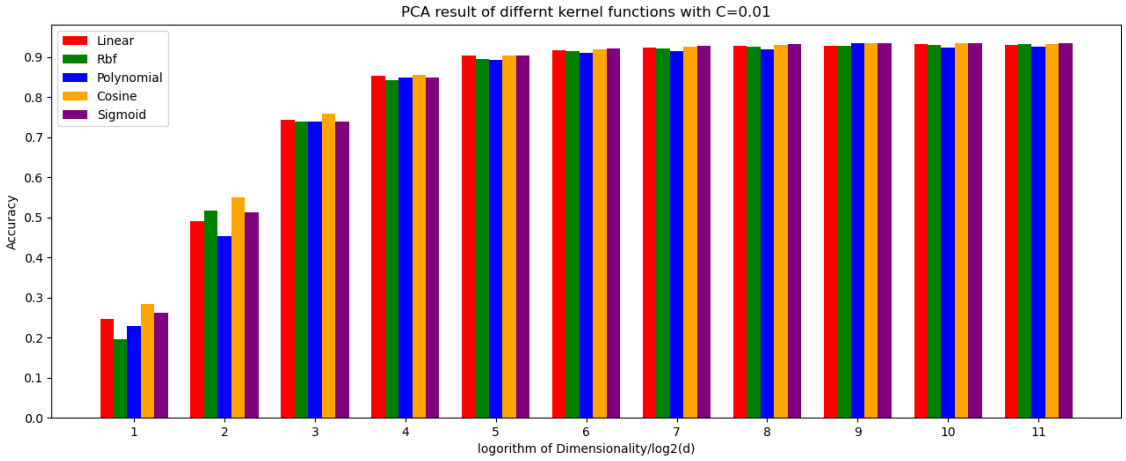

Fig. 14: Performance on Different Kernels when C=10 and $\gamma = 0.001$

4.2 Drawbacks of PCA

In the context of predictive modeling, dimensionality reduction is quite dangerous.

In finance, stock returns have about 15-25% annual standard deviation. Changes in bond yields are historically much lower standard deviation. If we perform PCA on the covariance matrix of stock returns and changes in bond yields, then the top PCs will all reflect variance of the stocks and the smallest ones will reflect the variances of the bonds. If we throw away the PCs that explain the bonds, then there might be some trouble. For instance, bonds might have different characteristics than stocks (thinner tails, different time-varying variance properties etc). These might be very important to model.

If we perform PCA on the correlation matrix, then we might see more of the PCs explaining bonds near the top. Buts ometimes we actually prefer the low variability features for anomaly detection, since a significant shift in a low variability dimension is a strong indicator of anomalous behavior.

5 Conclusion

In this project we explore nine different dimensionality reduction methods in order to reduce the time cost of pure linear SVM classifier and find some interesting phenomenons and regularities. Our experimental result could be summarized as follows

(1) Pure linear SVM classifier enjoys a good performance **0.931** which means pre-extracted features is linear discriminative, but this method is accompanied with high time cost and depends severely on the scale of data samples.
(2) We could use some feature projection methods such as PCA, kernel PCA and LDA to reduce dimensionalities. Although PCA's accuracy **0.932** does not overwhelm pure SVM, it could reduce the dimension gaintly to **1024** or even lower.
(3) For kernel PCA, since kernel function features collapse effect by distance metrics, it's necessary to fine-tune hyperparameters such as γ and C. Due to its special property, kernel PCA with sigmoid kernel reaches highest performance **0.935** with similar dimensions **1024**.
(4) For LDA, it enjoys more outstanding performance than PCA. Although its accuracy **0.930** is slightly lower, it reduce its dimension to only **49** which is one tenth of prior methods. Visualization shows its force to seperate different class clusters.
(5) For AE and VAE, the best result we got is to use VAE to reduce the dimensionality to **30**, and its accuracy is **83.9%**.In terms of characteristics, VAE can force the implicit vector generated by the input sample to roughly follow a standard normal distribution
(6) For Feature Selection By Variance and Random Forest, the performance is average, but they help us better understand the dangers of dimensionality reduction. Despite following some principles, we can observe significant fluctuations in these two methods.
(7) For t-SNE and Umap, the best result is to use Umap to reduce the dimension to **50** dimensions, and its accuracy rate is **85.7%**.But these two are the **fastest** methods, and after further understanding, we found that they are more dependent on the characteristics of the data set.

6 Contribution

6.1 Hou Shengyuan(518021910604)

(1) Related work part :SVM, PCA, kernel PCA and LDA.
(2) Experimental part: program code, Visualize and do experiment on four methods mentioned above and write corresponding report.
(3) Further discussion part: Write and visualise all further discussion part.
(4) Other: Typesetting of the whole report except for citation label and write introduction and abstract .

6.2 Yan Binghao(518021910753)

(1) Related work part: Random forest, Selection by variance, AE, VAE, UMAP and t-SNE.
(2) Experimental part: write six methods detailed in the report.
(3) Other:typesetting of all citation labels.

6.3 Wu Shiqu(518021910665)

(1) Provide material for Yan Binghao in related work part with Random forest, Selection by variance, AE, VAE, UMAP and t-SNE.
(2) Experimental part: programming visualize and do huge amounts of experiments on six methods mentioned above.
(3) Other: literature survey.

References

1. Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.
2. Pearson, Karl. "LIII. On lines and planes of closest fit to systems of points in space." The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science 2.11 (1901): 559-572.
3. Schölkopf, Bernhard, Alexander Smola, and Klaus-Robert Müller. "Nonlinear component analysis as a kernel eigenvalue problem." Neural computation 10.5 (1998): 1299-1319.
4. Van der Maaten, Laurens, and Geoffrey Hinton. "Visualizing data using t-SNE." Journal of machine learning research 9.11 (2008).
5. Mcinnes, Leland , and J. Healy . "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction." The Journal of Open Source Software 3.29(2018):861.
6. Fisher, Ronald A. "The use of multiple measurements in taxonomic problems." Annals of eugenics 7.2 (1936): 179-188.
7. LeCun, et al. "Backpropagation Applied to Handwritten Zip Code Recognition." Neural Computation (1989).
8. Y. Xian, C. H. Lampert, B. Schiele, Z. Akata. "Zero-Shot Learning - A Comprehensive Evaluation of the Good, the Bad and the Ugly", IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI) 40(8), 2018. (arXiv:1707.00600 [cs.CV])