CS245 Project 1: Dimensionality Reduction

1st Kaipeng Zeng
Shanghai Jiao Tong University, Shanghai, China
518030910374
zengkaipeng@sjtu.edu.cn

1st Yuxuan Xiong
Shanghai Jiao Tong University, Shanghai, China
518030910370
xiongyx@sjtu.edu.cn

***Abstract*—In this paper, we have tried three types of data dimension reduction methods, including feature selection, feature projection and feature learning, to reduce the dimensionality of pre-extracted deep learning features for AwA2 dataset. In order to measure the performance between each method, we train SVMs based on low-dimensional features produced by each method and compare the accuracy of each SVM. Our experiment shows that t-distributed stochastic neighbor embedding (t-SNE) method with dimensionality $n = 3$ and LDA preprocessing achieves the best accuracy, while Linear Discriminant Analysis (LDA) method with dimensionality $n = 50$ has the best time and space properties.**

***Index Terms*—dimensionality reduction, feature selection, feature projection, feature learning, SVM**

I. Introduction

Dimensionality reduction is a very common and important method in data mining. Reducing feature dimensionalities of the data, we can not only save a lot of memory space and training time, but can also alleviate the problem of overfitting to some extent. Therefore dimensionality reduction has been widely used in (the pre-processing of) various kinds of machine learning algorithms.

There are many kinds of dimensionality reduction methods, which can be divided into three categories in total, including feature selection, feature projection and feature learning. Each type of dimensionality reduction methods have their own strengths and weaknesses, which means that they are suitable for different application scenarios.

In this paper, we focus on the task of image classification. We will implement several types of dimensionality reduction methods on our own and use them to reduce the dimensionality of the pre-extracted deep learning features for AwA2 dataset. After that, we will train SVMs based on those low-dimensional features and compare the performance (classification accuracy) of each SVM in order to rank each method.

The structure of this paper will be organized as follows: Firstly, we will briefly introduce the mathematical principle and implementation of each dimensionality reduction algorithms in section II. What's next, we will conduct experiments to test the performance of each algorithm in section III. Conclusion of the while paper will be presented in section IV.

II. Methods

A. Feature Selection

Feature selection has the methods with the simplest idea among all. The core of the method is how to pick the most useful dimensions out of other dimensions.

Two basic method of feature selection are introduced in the classes, namely forward search and backward search. In forward search, we start from an empty set and add the useful dimensions into the set greedily while in the backward search, we remove dimensions from the full set. However, the two method might have the following two shortcomings:

- The algorithm might not work if we want to reduce our features to specific dimensions.
- The greedy algorithm might leads to a local optimum for feature selection.
- Though forward selection do not have recursion, sometimes the criterion of the greedy algorithm can still be time-consuming, while modifying criterion might lead to a worse result.

Thus, in this paper, we mainly focus on two kinds of feature selection method, namely *Greedy Forward Selection* and *Variance-based Selection*.

1) Faster-Greedy Forward Selection: The algorithm comes from a very simple idea: start from the empty set and pick out a random dimension from the feature and verify whether adding to to the set will improve the valid accuracy by a five-fold validation. However, according to the experiment, the five-fold validation is quite time-consuming and it take days to go through all 2048 dimensions. However, as it's shown in figure 1, when the number of selected dimensions is relatively low, the more dimension you choose, the higher tendency it will be for you to get a high validation accuracy. With this tendency, we can replace the original five-fold cross validation with a simpler criterion, which is using 20% random data to validate and other 80% to train. If adding a dimension will improve the performance, this dimension will be preserved in the set. On the contrary, if adding a dimension will lead to a worse performance, we will preserve this dimension with a specific probability p to erase the influence of randomness of new criterion and preserve the probability that this dimension can improve performance together with some unexplored dimensions.

2) Variance-based Selection: This algorithm is inspired by the principle of **PCA**. The dimension with higher variance

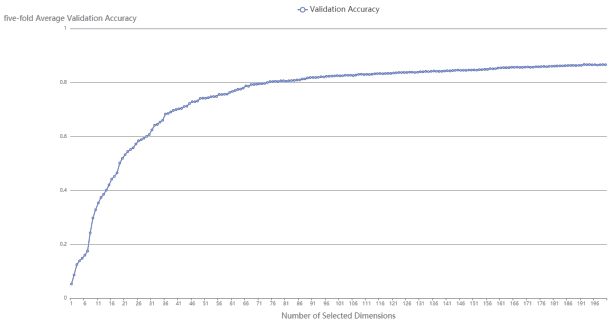


Fig. 1. Validation Accuracy under different numbers of dimensions. The dimensions are randomly selected.

might contribute more to the classification. Thus we choose the top k dimensions with largest variance and use them to compose the new features.

B. Feature Projection

Feature projection is a type of dimension reduction methods which project the original high-dimensional features into a low-dimensional space. The key of this kind of method is to find a set of base vectors of the new feature space. By elaborately selecting the base vectors of the low-dimensional space, feature projection methods can achieve a pretty well performance on refining original features. In this paper, we mainly focus on three kinds of feature projection method, namely *Principal Component Analysis (PCA)*, *Linear Discriminant Analysis (LDA)* and *Auto-Encoder*.

1) Principal Component Analysis: Principal Component Analysis (PCA) is one of the most popular dimesionality reduction methods. The intuition of PCA is to choose base vectors on which direction the data have maximum variance.

Mathematically, given a set of decentralized data, we want to find a unit vector v to maximize the variance of the data, i.e.

$$\max_v \frac{1}{n}\sum_i (v^T x_i)^2$$
$$s.t. \quad ||v|| = 1$$

or equivalently

$$\max_v v^T X X^T v$$
$$s.t. \quad v^T v = 1$$

The Lagrange form of above objective is:

$$\mathcal{L}(v) = v^T X X^T v - \lambda(v^T v - 1)$$

According to KKT conditions, we have

$$\frac{\partial \mathcal{L}}{\partial v} = 2XX^T v - 2\lambda v = 0$$

So

$$XX^T v = \lambda v \tag{1}$$

v is an eigenvector of the matrix XX^T. Bring equation (1) back to the original objective, we have

$$\max_v \lambda v^T v$$
$$s.t. \quad v^T v = 1$$

i.e.

$$\max \lambda$$

So v is the the eigenvector corresponding to the maximum eigenvalue of the matrix XX^T.

In the actual code implementation, we just need to find the maximum d eigenvalues $\{\lambda_1, \lambda_2, \cdots, \lambda_d\}$ and their corresponding (unit) eigenvectors $\{v_1, v_2, \cdots, v_d\}$ of matrix XX^T. Then we can reduce the dimensionality of the original features by projecting them into the sapce formed by base vectors $\{v_1, v_2, \cdots, v_d\}$.

2) Linear Discriminant Analysis: Linear Discriminative Analysis (LDA) is another type of feature projection method to reduce feature's dimensionality. Unlike PCA which finds the component axis that maximize the variance, LDA is a label-aware method which intends to find the component axis that maximize the class separation.

More concretely, given a set of data which can be divided into two categories, LDA want to find a base vector which can maximize the variance between different categories while minimize the variance within the same category. This intuition leads to the definition of the Fisher function

$$\begin{aligned} J(v) &= \frac{(v^T\mu_1 - v^T\mu_2)^2}{\sigma_1^2 + \sigma_2^2} \\ &= \frac{(v^T\mu_1 - v^T\mu_2)^2}{\sum_i^{C_1}(v^T x_{1,i} - v^T\mu_1)^2 + \sum_i^{C_2}(v^T x_{2,i} - v^T\mu_2)^2} \\ &= \frac{v^T(\mu_1 - \mu_2)(\mu_1 - \mu_2)^T v}{v^T(C_1\Sigma_1 + C_2\Sigma_2)v} \end{aligned}$$

Denote

$$\begin{cases} S_B = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^T \\ S_W = C_1\Sigma_1 + C_2\Sigma_2 \end{cases}$$

The objective of LDA is to

$$\max_v J(v) \equiv \max_v \frac{v^T S_B v}{v^T S_W v}$$

or equivalently

$$\max_v v^T S_B v$$
$$s.t. \quad v^T S_W v = 1$$

The Lagrange multiplier of above objective is

$$\mathcal{L}(v) = v^T S_B v - \lambda(v^T S_W v - 1)$$

According to KKT conditions, we have

$$\frac{\partial \mathcal{L}}{\partial v} = 2S_B v - 2\lambda S_W v = 0$$

So

$$S_B v = \lambda S_W v \tag{2}$$

$$S_w^{-1} S_B v = \lambda v \tag{3}$$

According to equation (3), v is an eigenvector of matrix $S_W^{-1} S_B$. Bring equation (2) back to the original objective, we have

$$\begin{aligned} &\max_v \lambda v_T S_W v \\ &s.t. \quad v^T S_W v = 1 \end{aligned}$$

i.e.

$$\max \lambda$$

So same as PCA, v is the the eigenvector corresponding to the maximum eigenvalue of the matrix $S_W^{-1} S_B$.

In the actual code implementation, we just need to find the maximum d eigenvalues $\{\lambda_1, \lambda_2, \cdots, \lambda_d\}$ and their corresponding (unit) eigenvectors $\{v_1, v_2, \cdots, v_d\}$ of matrix $S_W^{-1} S_B$. Then we can reduce the dimensionality of the original features by projecting them into the space formed by base vectors $\{v_1, v_2, \cdots, v_d\}$.

Note that when there are more than two categories in the dataset, we can extend the expression of S_B and S_W as

$$\begin{cases} S_B = \sum_i C_i (\mu_i - \mu)^T (\mu_i - \mu) \\ S_W = \sum_i \sum_{j=1}^{C_i} (x_{i,j} - \mu_i)^T (x_{i,j} - \mu_i) \end{cases}$$

3) Auto-Encoder: Auto-Encoder is a kind of special symmetric neural network, which learns to generate a output as close to its input as possible. The network mainly consists of two parts, namely the encoder and decoder. The encoder transform the input into a feature with lower dimension, which is a hidden layer in the neural network in fact. And the decoder reconstruct the feature from the code. Or in mathematics for a input $X \in \mathcal{X} \subset \mathbb{R}^n$, if we want to map the feature into $c \in \mathcal{C} \subset \mathbb{R}^m$, with Φ as encoder an Ψ as decoder, we will have

$$\begin{aligned} \Phi &: \mathcal{X} \to \mathcal{C} \\ \Psi &: \mathcal{C} \to \mathbb{R}^n \end{aligned} \tag{4}$$

And the objective function of auto-encoder will be

$$\begin{aligned} &\min_{\Psi, \Phi} \left\| X - \hat{X} \right\|^2 \\ &\text{s.t.} \quad \hat{X} = \Psi(\Phi(X)) \end{aligned} \tag{5}$$

For any input X, $c = \Phi(X)$ has lower dimension, which means it can be regarded as a kind of low-dimension representation of X.

C. Feature Learning

Unlike previous two types of dimensionality reduction methods, **feature learning** usually cannot be written as a linear projection from a high-dimensional space into a low-dimensional space. In fact, feature learning is a type of method which learns some "inner structure" of the original features and try to find a new set of features which holds the same "inner structure" in a low-dimensional space. In this paper, we mainly focus on two kinds of feature learning method, namely *t-distributed stochastic neighbor embedding (t-SNE)* and *Locally Linear Embedding (LLE)*.

1) t-Distribution Stochastic Neighborhood Embedding: In Stochastic Neighborhood Embedding (SNE), we want to find a new set of low-dimensional features which maintain same "relative distance" between each features.

More concretely, we define the "conditional probability" of node j given node i as follows

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

As we can see, $p(j|i)$ is a value related to relative distances between each pair of nodes. Given the whole conditional probability distribution $p(j|i)$, we can depict the structure of data based on their relative distances.

In the same way, we can define the "conditional probability" of node j given node i after dimensionality reduction as

$$q(j|i) = \frac{exp(-||\tilde{x}_i - \tilde{x}_j||^2)}{\sum_{k \neq i} exp(-||\tilde{x}_i - \tilde{x}_k||^2)}$$

to depict the relative distance between each pair of $\tilde{x}$, where $\tilde{x}$ is the low-dimensional feature generated from x

The key of SNE algorithm is that we want two distributions $p(j|i)$ and $q(j|i)$ to be as close as possible, since close distribution between x and $\tilde{x}$ means $\tilde{x}$ can represent x well.

Mathematically, we can measure the "distance" between two probability distribution using their KL distance, i.e.

$$KL(p||q) = \mathbf{E}(p \log \frac{p}{q})$$

Define the loss function L as

$$L = \sum_i KL(p_i||q_i) = \sum_i \sum_j p(j|i) \log \frac{p(j|i)}{q(j|i)}$$

By minimize L, we can get the optimal set of low-dimensional features $\{\tilde{x}_i\}$.

If we replace the expression of $p(j|i)$ (same for $q(j|i)$) as

$$p(j|i) = \frac{(1 + ||x_i - x_j||^2)^{-1}}{\sum_{k \neq i} (1 + ||x_i - x_k||^2)^{-1}}$$

then this algorithm is called t-Distribution Stochastic Neighborhood Embedding (t-SNE).

2) Locally Linear Embedding: Locally Linear Embedding (LLE) is similar to SNE algorithm, except that LLE only focuses on every node's closest k neighbors. Specifically, for each node i, we find its top k closest neighbor nodes $\mathcal{N}_i$ and try to represent it by those k nodes:

$$\begin{aligned} &\min_W \sum_i ||x_i - \sum_{j \in \mathcal{N}_i} w_{ij} x_j||^2 \\ &s.t. \quad \sum_j w_{ij} = 1 \end{aligned}$$

We can simplify above loss function as

$$\begin{aligned}&\sum_i ||x_i - \sum_{j\in\mathcal{N}_i} w_{ij}x_j||^2\\ =&\sum_i ||\sum_{j\in\mathcal{N}_i} w_{ij}(x_i - x_j)||^2\\ =&\sum_i ||(X_i - N_i)^T W_i||^2\\ =&\sum_i W_i^T(X_i - N_i)(X_i - N_i)^T W_i\end{aligned}$$

where $X_i^{(n\times k)} = [x_i, x_i, \cdots, x_i]$ and $N_i^{(n\times k)} = concat(\mathcal{N}_i)$. Denote $Z_i = (X_i - N_i)(X_i - N_i)^T$, then the original objective can be written as

$$\begin{aligned}&\min_W \sum_i W_i^T(x_i - x_j)(x_i - x_j)^T W_i\\ &s.t.\ \forall i,\ W_i^T 1_k = 1\end{aligned}$$

Using Lagrange multiplier method and KKT conditions, we can get the optimal W^*:

$$W_i^* = \frac{Z_i^{-1}1_k}{1_k^T Z_i^{-1} 1_k}$$

In LLE algorithm, we want to find a set of low-dimensional features Y which maintain the same property as X, i.e.

$$\begin{aligned}&\min_Y \sum_i ||y_i - \sum_{j\in\mathcal{N}_i} w_{ij}^* y_j||^2\\ &s.t.\quad Y^T Y = 1\end{aligned}$$

Using the similar method used before, we can work out that the optimal value of Y satisfies:

$$(1-W)(1-W)^T Y^T = \lambda Y^T$$

By choosing the eigenvectors corresponding to the d smallest eigenvalues of matrix $(1-W)(1-W)^T$, we can form matrix Y^T and therefore get a new set of d-dimensional features Y.

III. Experiments

In order to compare the performance of each dimensionality reduction methods mentioned above, we train SVMs for each method and rank all dimensionality reduction methods based on the accuracy of their corresponding SVMs.

A. Experimental Settings

1) Dataset: The dataset we used in our experiments is Animals with Attributes (AwA2) dataset [1]. This is a image classification dataset consists of 37322 images of 50 animal classes. We use deep learning features of the images pre-extracted by ResNet101 as the original feature (2048 dimension).

We shuffle the whole dataset and split it into two parts: 60% of the images (original features) are used for training and the other 40% of the images (original features) are used for testing. Moreover, we use k-fold cross-validation within the training set in order to tune the hyper parameters in our algorithms (such as the parameter C in SVM). In the rest of this paper, we take $k = 5$ in default.

2) Support Vector Machine: A support vector machine (SVM) will be trained on the post-processing features (in the training set) generated by each dimensionality reduction method. After that, we will test the performance (accuracy) of the SVM on the test dataset.

SVM is a type of generalized linear classifier based on supervised learning. The intuition of SVM is to find the maximum-margin hyperplane which can classify the data into two categories with the strongest robustness.

Mathematically, the margin of a SVM w.r.t a single sample point (X_i, y_i) is defined as

$$\gamma_i = y_i(w^T X_i + b)$$

The goal of (hard-)SVM is to find a hyperplane

$$w^T X + b = 0$$

which maximize the lowest margin among all the sample points, i.e.

$$\begin{aligned}&\max_{w,b,\tau} \tau\\ s.t.\quad &\forall i,\ y_i(w^T X_i + b) \geq \tau\\ &||w|| = 1\end{aligned}$$

or equivalently

$$\begin{aligned}&\min_{w,b} \frac{1}{2}||w||^2\\ s.t.\quad &\forall i,\ y_i(w^T X_i + b) \geq 1\end{aligned}$$

Note that our dataset may cannot be perfectly separated, which may bring failure to the hard-SVM. So we can modify above objective into soft-SVM:

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

Using Lagrange multiplier method, we can transfer the original objective into its dual form:

$$\begin{aligned}&\max_\alpha \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j y_i y_j \alpha_i \alpha_j X_i^T X_j\\ s.t.\quad &\forall i,\ 0 \leq \alpha_i \leq C\\ &\sum_i \alpha_i y_i = 0\end{aligned}$$

Above objective is a standard **quadratic programming** problem, which can be solved by some specific convex optimization algorithms. Because SVM is not our keypoint in this paper, so we directly use the `svm.SVC` class provided by `sklearn` library in our following experiments.

B. Experimental Results

1) No Dimesionality Reduction: First of all, we directly use the original features from AwA2 dataset to train a linear SVM as the baseline. The performance of SVM are shown in Fig. 2.

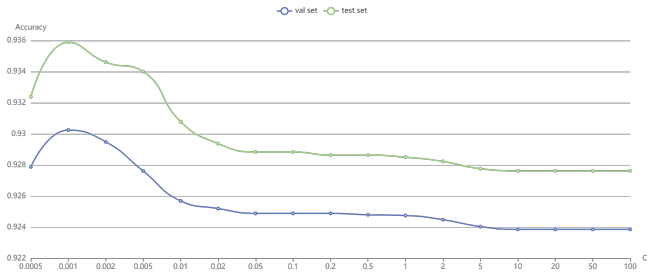


Fig. 2. The accuracy of SVM trained on original features with different C's

As we can see from fig. 2, the performance of our SVM achieves highest accuracy when $C = 0.001$ (Val Acc = 93.03%, Test Acc = 93.59%). Higher or lower value of C will decline the performance of our SVM. However, it is noteworthy to mention that the influence of C is limited in this case, since the accuracy of SVM only changes within 1% under different C's.

2) Faster-Greedy Forward Selection: The initial experiment starts with a simple greedy algorithm: If adding a dimension can improve average validation accuracy of five-fold validation, preserve it in the final answer, or remove it. However, in the experiment, two problems are founded:

- this algorithm gets stuck when the number of selected dimension reaches 78 and can't add anymore dimensions.
- If our target number of dimension is larger than 78, the algorithm will explore all the 2048 dimensions. This is quite time-consuming. Even if the cross validation is replaced with a faster criterion, it still takes us **7 hours 23 minutes and 40 seconds** to generate the compressed feature.

Using the generated 78-dim features to train SVM with different C, the result is shown in table I. The faster-greedy

TABLE I
THE VALIDATION ACCURACY AND TEST ACCURACY OF SIMPLE GREEDY ALGORITHM UNDER DIFFERENT C

C	Validation Acc	Test Acc
0.02	83.92%	84.35%
0.1	83.94%	84.44%
1	82.57%	82.92%
10	81.75%	82.25%
100	81.48%	82.05%

forward selection is born for solving the problems above. Set the probability of accepting a worse result as $p = 0.18$, we have the result of different dimension under different hyper-parameter C, which is shown in table II, where n represents the number of dimensions

TABLE II
THE VALIDATION ACCURACY OF FASTER-GREEDY FORWARD SELECTION UNDER DIFFERENT DIMENSIONS AND VALUES OF C

n \ C	0.02	0.1	1	10	100
1	5.46%	5.89%	6.10%	6.08%	6.08%
2	6.47%	7.04%	7.31%	7.36%	7.35%
5	14.72%	16.35%	17.04%	17.08%	17.06%
10	26.34%	30.57%	32.23%	32.44%	32.54%
20	42.28%	45.98%	47.87%	47.90%	47.91%
50	75.58%	76.63%	75.66%	74.96%	74.70%
100	82.65%	82.94%	81.43%	80.46%	80.21%
200	89.03%	88.22%	87.17%	86.93%	86.89%

Using the optimal Cs of different dimensions, we will have the test accuracy of different dimension of faster-greedy forward selection shown in table III

TABLE III
OPTIMAL C AND CORRESPONDING TEST ACCURACY OF DIFFERENT DIMENSION

num of dimensions	optimal C	Test Accuracy
1	1	6.18%
2	10	7.83%
5	10	17.11%
10	100	33.35%
20	100	48.50%
50	0.1	77.60%
100	0.1	83.54%
200	0.02	**89.58%**

To prove that our algorithm reduce the run-time greatly at the cost of only little test accuracy, we replace the simple criterion in faster-greedy selection algorithm with the cross validation and use the same options set of C, which is $\{0.02, 0.1, 1, 10, 100\}$ to train SVM. The C with the highest validation accuracy is called the optimal C, and we have the test accuracy under optimal Cs and the run-time compared with the faster-greedy forward selection shown in table IV, where FG means the our faster-greedy forward selection, CV means replacing the simple criterion with cross validation and n represents the number of dimensions. It's very easy to find that using simple criterion will not lead to a sharp decrease of test accuracy, and even under some of the dimension the test accuracy is higher.

TABLE IV
THE RUN-TIME AND THE TEST ACCURACY UNDER OPTIMAL C OF SIMPLE CRITERION AND CROSS VALIDATION

n	Run-time		Test Accuracy	
	FG	CV	FG	CV
2	30s	3min 19s	7.83%	8.62%
5	1min 45s	17min 43s	17.11%	19.71%
10	4min 28s	33min 35s	33.35%	23.45%
20	7min 49s	43min 31s	48.50%	53.14%
50	21min 27s	1h 26min 37s	77.60%	70.48%
100	1h 11min 39s	2h 23min 52s	83.54%	84.78%
200	2h 46min 48s	6h 20min 28s	89.58%	89.50%

3) Variance-based Selection: Sorting all the dimensions in the descending order and pick out the top k order, we can have a new feature with k as the number of dimension of the data. That's is how the variance-based selection works. The result of different dimension count under different hyper-parameter C is shown in table V. In table V, n represents the number of dimensions, and the red items in the table represent the best performance under the same number of dimension.

TABLE V
THE VALIDATION ACCURACY OF THE VARIANCE-BASED SELECTION UNDER DIFFERENT DIMENSIONS AND DIFFERENT VALUES OF C.

n \ C	0.01	0.02	0.1	0.5	1	10
1	6.39%	6.65%	6.91%	6.91%	6.92%	6.90%
2	8.68%	8.90%	9.42%	9.53%	9.53%	9.53%
5	23.28%	23.95%	24.66%	24.92%	24.98%	24.99%
10	39.71%	40.86%	42.40%	42.94%	43.07%	43.15%
20	64.21%	65.31%	66.53%	66.85%	66.77%	66.62%
50	81.90%	82.16%	81.94%	81.53%	81.29%	80.72%
100	89.05%	89.05%	88.26%	87.58%	87.27%	86.74%
200	91.11%	90.91%	90.09%	89.50%	89.36%	89.25%
500	92.15%	91.87%	91.38%	91.29%	91.29%	91.23%
1000	92.49%	92.33%	92.21%	92.21%	92.22%	92.14%
1500	92.55%	92.45%	92.43%	92.42%	92.42%	92.33%
2000	92.57%	92.52%	92.50%	92.49%	92.49%	92.40%

Using the optimal Cs under different dimension, we can get the test accuracy of different data representation under different dimensions, which is shown in figure 3 and table VI.

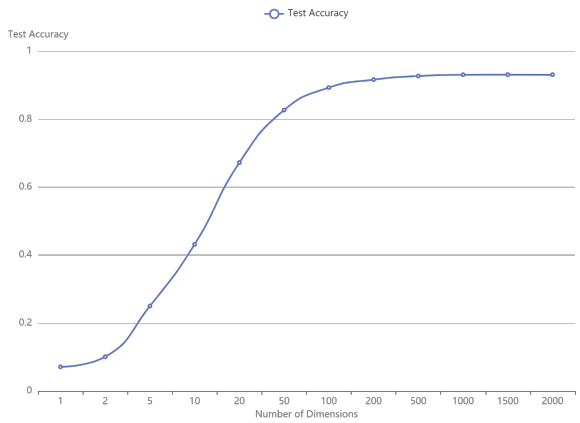


Fig. 3. Test Accuracy Under the optimal C of different number of dimensions of variance-based selection

From table VI and figure 3, We can get as 97% performance as the original feature with only 200 dimension of them. We can also find that the variance-based selection algorithm can have a outstanding performance when the number of dimension is just 100, this means variance-based selection algorithm is quite efficient. Also we can find out that when we increase the number of dimension from 200 to 2000, the test accuracy only have an increase of 1.46%, which means there is much redundancy among the features, providing a strong support for dimensionality reduction in turn.

TABLE VI
OPTIMAL C AND CORRESPONDING TEST ACCURACY OF DIFFERENT DIMENSIONS OF VARIANCE-BASED SELECTION

num of dimension	optimal C	test accuracy
1	1	7.12%
2	10	10.09%
5	10	25.03%
10	10	43.20%
20	0.5	67.23%
50	0.02	82.70%
100	0.01	89.30%
200	0.01	91.63%
500	0.01	92.71%
1000	0.01	93.07%
1500	0.01	**93.13%**
2000	0.01	93.09%

4) Principal Component Analysis: Following the mathematical principle introduced in Section II, we realize the PCA algorithm with `numpy` python library and train a SVM on the low-dimensional features generated by our PCA function. We use k-fold ($k = 5$) to find the best value of the hyper parameter C. The validation results are shown in TABLE VII.

Observing TABLE VII, we can conclude as follows:

- Different dimensionalities (n) of features have different optimum hyper parameter C.
- When $n = 2000$ with $C = 0.001$, SVM achieves its optimum performance ($Accracy = 92.92\%$).
- The performance of SVM increases monotonically with the increase of feature dimensionality n under the same C. This phenomenon is also confirmed in Fig. 4, in which we visualize ACC-n curve under several C's.

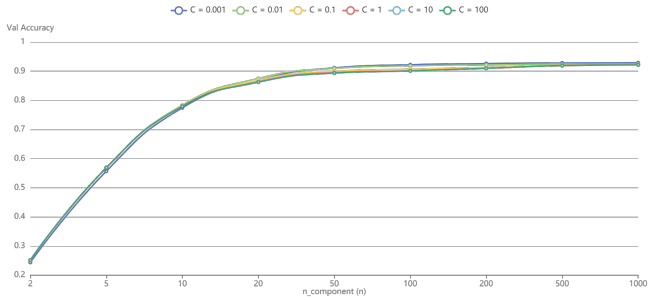


Fig. 4. [PCA] The Val ACC - n curve under several C's.

After training and validation phase, we test our SVMs which are trained on various dimensionalities of features with their own optimum C, the test result in listed in TABLE VIII and visualized in Fig. 5. Farthermore, we also recorded the training time spended to train SVM on each n (with their optimum C), which is also visualized in Fig. 5.

As we can see from Fig. 5, the performance of SVM is monotonously increasing with the increase of n, which is consistent with our conclusion made in validation phase. However, the training time of SVM has its minimum when $n = 20$. Both the decrease of n and increase of n from the value of 20 will lead to a longer training time.

TABLE VII

THE SVM VALIDATION ACCURACY TRAINED ON DIFFERENT DIMENSIONAL (n) FEATURES REDUCED BY PCA ALGORITHM WITH VARIOUS HYPER PARAMETER C. THE RED VALUE IN EACH ROW IS THE OPTIMAL ACC. UNDER CORRESPONDING n. NOTE THAT WHEN $n = 1000$ AND $C = 0.001$, SVM ACHIEVES THE OPTIMUM PERFORMANCE.

n \ C	0.0005	0.001	0.002	0.005	0.01	0.02	0.05	0.1	0.2	0.5	1	2	5	10	20	50	100
2	23.96%	24.40%	24.73%	25.03%	25.14%	25.15%	25.14%	25.11%	25.15%	25.15%	25.19%	25.19%	25.19%	25.18%	25.17%	25.18%	25.15%
5	54.41%	55.69%	56.48%	56.92%	56.92%	56.97%	56.93%	56.87%	56.92%	56.92%	56.93%	56.91%	56.89%	56.89%	56.89%	56.87%	56.87%
10	76.57%	77.46%	78.07%	78.31%	78.35%	78.47%	78.34%	78.24%	78.18%	78.10%	78.05%	78.00%	78.01%	77.98%	78.02%	77.99%	78.00%
20	86.24%	86.88%	87.40%	87.63%	87.52%	87.37%	87.07%	86.91%	86.75%	86.52%	86.43%	86.39%	86.33%	86.35%	86.30%	86.25%	86.25%
50	90.87%	91.18%	91.30%	91.19%	91.02%	90.75%	90.30%	89.98%	89.72%	89.66%	89.56%	89.52%	89.46%	89.41%	89.37%	89.35%	89.36%
100	91.88%	92.24%	92.23%	92.07%	91.89%	91.43%	91.00%	90.70%	90.44%	90.29%	90.21%	90.22%	90.15%	90.13%	90.14%	90.11%	90.11%
200	92.38%	92.64%	92.69%	92.51%	92.21%	91.78%	91.44%	91.29%	91.16%	91.08%	91.08%	91.08%	91.04%	91.02%	91.00%	90.99%	90.99%
500	92.65%	92.86%	92.81%	92.59%	92.35%	92.22%	92.10%	92.05%	92.02%	92.02%	92.02%	92.02%	91.95%	91.94%	91.93%	91.93%	91.93%
1000	92.75%	**92.92%**	92.82%	92.66%	92.49%	92.41%	92.24%	92.25%	92.25%	92.25%	92.26%	92.25%	92.20%	92.17%	92.17%	92.17%	92.17%

TABLE VIII

THE SVM TEST ACCURACY TRAINED ON DIFFERENT DIMENSIONAL (n) PCA-REDUCED FEATURES WITH THEIR OPTIMUM HYPER PARAMETER C.

n	optimum C	Test Accuracy
2	1	25.585%
5	0.02	57.334%
10	0.02	78.580%
20	0.005	87.606%
50	0.002	91.885%
100	0.001	92.522%
200	0.002	93.193%
500	0.001	93.441%
1000	0.001	**93.542%**

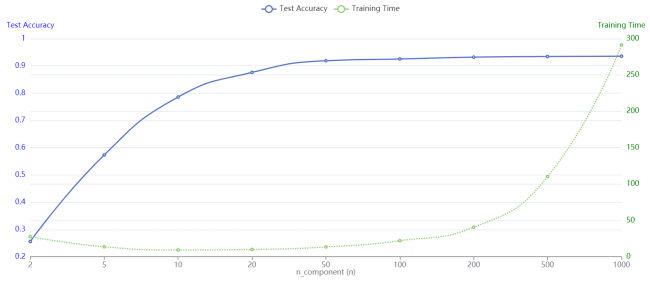


Fig. 5. [PCA] The Test ACC - n curve (blue solid curve) and training time curve (green dotted curve) under each n's optimum C.

The increase of n leading to longer training time is quite easy to explain, since larger dimensionality of features means more calculation to do with the matrices. However, the decrease of n also leading to longer training time is a little bit counterintuitive. After analysis, our group guess that may be small value of n cannot separate each type of data well. This fact may lead to a larger #learning iteration for the SVM to find the optimum parameters and therefore increase the total training time.

In conclusion, the performance of SVM trained on PCA-reduced features increased monotonously with the increase of dimensionality of feature n. However, if we consider both the accuracy of SVM as well as the training speed of SVM, our group think it would be a nice choice to choose $n = 50 \sim 100$ because of their relatively high accuracy and remarkable high speed (comparing with greater n).

5) Linear Discriminant Analysis: Following the mathematical principle introduced in Section II, we realize the LDA algorithm with `numpy` python library and train a SVM on the low-dimensional features generated by our LDA function. We use k-fold ($k = 5$) to find the best value of the hyper parameter C. The validation results are shown in TABLE IX.

Observing TABLE IX, we can conclude as follows:

- Different dimensionalities (n) of features have different optimum hyper parameter C (same as PCA).
- When $n = 50$ with $C = 0.05$, SVM achieves its optimum performance ($Accracy = 96.96\%$).
- Different from the PCA case, the performance of SVM increases with the increase of feature dimensionality n when n is small, and then decreases (slightly) with the increase of n when n is large (under the same C). This phenomenon is also confirmed in Fig. 6, in which we visualize ACC-n curve under several C's.

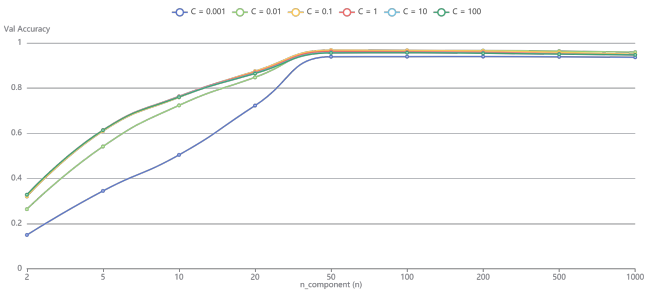


Fig. 6. [LDA] The Val ACC - n curve under several C's.

The test result of SVM trained on LDA-reduced features (n-dim) under n's optimum hyper parameter C is shown in TABLE X and visualized in Fig 7. Farthermore, we also recorded the training time spended to train SVM on each n (with their optimum C), which is also visualized in Fig. 7.

As we can see from Fig. 7, the SVM trained on $n = 50$ and $C = 0.05$ not only achieves the highest accuracy among all sets of parameters, but also spends the lowest period of time for training. This shows that $(n = 50, C = 0.05)$ is absolutely

TABLE IX

THE SVM VALIDATION ACCURACY TRAINED ON DIFFERENT DIMENSIONAL (n) FEATURES REDUCED BY LDA ALGORITHM WITH VARIOUS HYPER PARAMETER C. THE RED VALUE IN EACH ROW IS THE OPTIMAL ACC. UNDER CORRESPONDING n. NOTE THAT WHEN $n = 50$ AND $C = 0.05$, SVM ACHIEVES THE OPTIMUM PERFORMANCE.

n \ C	0.0005	0.001	0.002	0.005	0.01	0.02	0.05	0.1	0.2	0.5	1	2	5	10	20	50	100
2	11.10%	14.98%	17.53%	20.50%	26.40%	29.71%	31.07%	32.02%	32.57%	32.81%	32.74%	32.74%	32.77%	32.78%	32.78%	32.78%	32.79%
5	27.26%	34.48%	43.45%	50.96%	54.15%	57.53%	60.19%	60.98%	61.19%	61.29%	61.41%	61.40%	61.33%	61.33%	61.39%	61.36%	61.38%
10	40.61%	50.43%	59.57%	68.14%	72.36%	74.41%	75.58%	75.91%	76.22%	76.35%	76.34%	76.32%	76.22%	76.15%	76.08%	76.06%	76.02%
20	60.40%	72.25%	77.73%	83.20%	84.77%	86.37%	87.14%	87.47%	87.44%	87.11%	87.02%	86.89%	86.60%	86.65%	86.54%	86.47%	86.44%
50	91.39%	93.89%	95.69%	96.30%	96.63%	96.80%	**96.96%**	96.81%	96.71%	96.48%	96.25%	96.06%	95.90%	95.71%	95.68%	95.58%	95.55%
100	91.44%	93.92%	95.69%	96.36%	96.64%	96.84%	96.82%	96.71%	96.51%	96.22%	96.07%	95.89%	95.76%	95.73%	95.70%	95.67%	95.64%
200	91.37%	93.94%	95.65%	96.35%	96.61%	96.67%	96.70%	96.47%	96.22%	95.99%	95.88%	95.74%	95.60%	95.54%	95.48%	95.43%	95.41%
500	91.24%	93.84%	95.47%	96.20%	96.35%	96.21%	96.06%	95.85%	95.57%	95.37%	95.26%	95.14%	95.09%	95.09%	95.08%	95.04%	95.02%
1000	90.98%	93.63%	95.16%	95.73%	95.91%	95.62%	95.29%	95.03%	94.96%	94.82%	94.78%	94.79%	94.77%	94.76%	94.73%	94.69%	94.69%

TABLE X

THE SVM TEST ACCURACY TRAINED ON DIFFERENT DIMENSIONAL (n) LDA-REDUCED FEATURES WITH THEIR OPTIMUM HYPER PARAMETER C.

n	optimum C	Test Accuracy
2	0.5	31.084%
5	1	57.870%
10	0.5	72.027%
20	0.1	83.260%
50	0.05	**93.139%**
100	0.02	93.032%
200	0.05	92.811%
500	0.01	92.750%
1000	0.01	92.764%

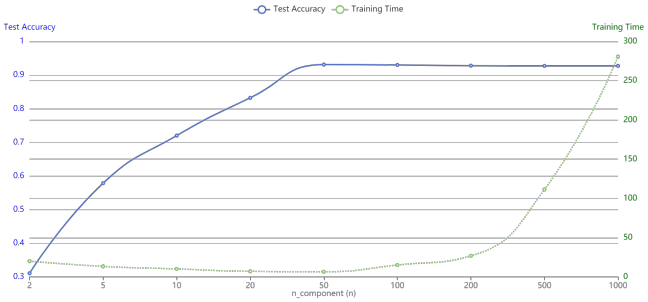


Fig. 7. [LDA] The Test ACC - n curve (blue solid curve) and training time curve (green dotted curve) under each n's optimum C.

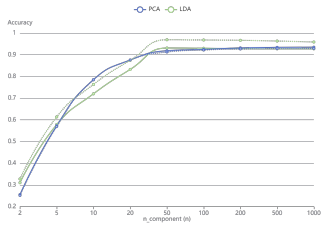


Fig. 8. The test/val Acc. (solid/dotted curves) of SVM trained on PCA/LDA-reduced features with different dimensionality n

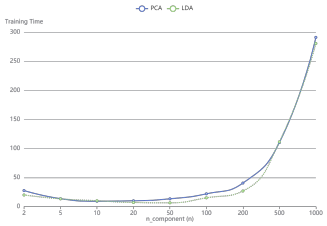


Fig. 9. The training time used to train SVM on PCA/LDA-reduced features with different dimensionality n

the best pair of hyper parameters for LDA algorithm in this task.

Our group also did some comparation between PCA and LDA. Fig. 8 and Fig. 9 shows PCA and LDA's Test/Val Accuracies and Training times respectively.

Fig. 8 shows that these two kinds of feature projection methods have similar test accuracy under different n, while Fig. 8 shows that these two methods also leads similar training time to learn the SVM. These facts shows great similarity between these two methods.

However, if we observe the curves in Fig. 8 more carefully, we can find that there is a wider gap between LDA's val and test Acc. than PCA's. Our group think this phenomenon is reasonable since LDA is a label-aware algorithm which can fit the training set better than PCA. On the other hand, this phenomenon also shows that LDA can more easily leads to overfitting than PCA, which should be carefully avoided in the future.

In conclusion, when we take $n = 50$ and $C = 0.05$, we not only can train a best-performanced SVM on LDA-reduced data, but also can achieve the fastest training speed. So our group strongly recommend $(n = 50, C = 0.05)$ as the parameters used in LDA method.

6) Auto-Encoder: In the experiment, we implement a auto-encoder with six linear layers using `Pytorch`. The structures of both encoder and decoder are very simple. The encoder consists of three linear layers and two $ReLU$ activate function. And the decoder consists of three linear layers and three $ReLU$ function. More narrowly, the dimensions of the input and the hidden layers are $\{2048, 1024, 512, d, 512, 1024, 2048\}$, where d represents the number of dimension of input's representation. The structure of our auto-encoder is shown in Fig. 10.

As for training, we use $Adam$ optimizer to train the auto-encoder with a batch size of 256 and initial learning rate of 0.001. The training is unexpectedly fast and the model converges within only tens of epochs.

Using the the d-dimensional encoded data as new data feature representation, we have the validation accuracy under different C, which is shown in tableXI. In table XI, n represents the number of dimension, and the red items in the table represents the best performance under the same number of dimension. All the accuracy shown in the table is the average

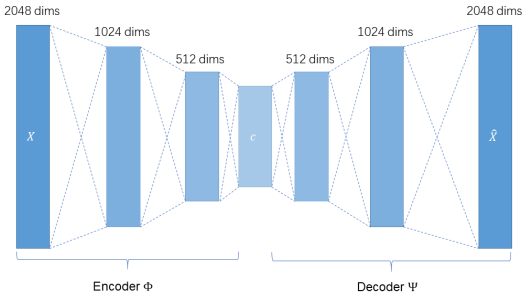


Fig. 10. The structure of our auto-encoder

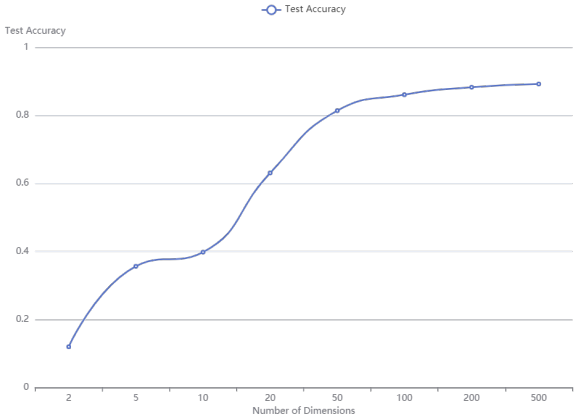


Fig. 11. Test Accuracy of Auto-Encoder Under the Best C of different number of dimensions

accuracy under a five-fold validation. According to this table, we will have when $n = 500$ and $C = 1$, the auto-encoder will have the best performance.

TABLE XI
THE VALIDATION ACCURACY OF THE AUTO-ENCODER UNDER DIFFERENT DIMENSIONS AND DIFFERENT VALUES OF C.

n \ C	0.02	0.1	0.5	1	10	100
2	4.40%	7.05%	7.79%	8.24%	10.85%	11.74%
5	6.98%	10.75%	21.13%	27.73%	34.40%	35.24%
10	4.51%	10.29%	25.00%	32.31%	39.00%	39.62%
20	6.94%	17.53%	49.09%	55.44%	61.72%	62.08%
50	12.57%	52.40%	74.63%	77.70%	81.06%	80.12%
100	20.93%	70.10%	81.90%	83.61%	85.30%	84.07%
200	53.91%	81.02%	86.77%	87.73%	88.18%	86.91%
500	70.43%	84.98%	88.38%	**88.75%**	87.92%	87.05%

Using those C of highest validation accuracy of different number of dimensions, the test accuracy is shown in figure 11. From figure 11, we can find that the increase of test accuracy is not worth mentioning when the number of dimensions is larger than 100, which means the auto-encoder can compress the data feature efficiently (just compress the data into 100 dimensions, we can have a test accuracy of 86.06%, which is 92% of the SVM use uncompressed feature to train).

7) t-Distribution Stochastic Neighborhood Embedding: we use `sklearn.manifold.TSNE` to implement t-Distribution Stochastic Neighborhood Embedding. The parameters of this function are:

- *n_component*, representing the target number of dimension of reduced feature.
- *method*, can be chosen between `exact` and `barnes_hut`. The `exact` method requires a lot of computation resources and a large memory. It has a time complexity of $O(n^2)$. Also, we can use the `barnes_hut` to make a approximation, which has a time complexity of $O(n \log n)$. However, this approximation only works when $n\_component < 4$.
- *init*, which can be chosen between `random` and `pca`. This parameter determines how the output is initialized. In this experiment, we only use `pca`.

Limited by the computation resources, we only conduct the experiments with $n\_component = 2$ and $n\_component = 3$. Also we tried three method for experiment:

- use the raw data as the input, which is denoted as **RAW-tsne** in the following part.
- use LDA to reduce the input to 49 dimensions and use it as the input of t-sne. This is denoted as **LDA-tsne** in the following part.
- use pca to reduce the input to 200 dimensions and use it as the input of t-sne. This is denoted as **PCA-tsne** in the following part.

We also conduct the training under different values of C, the result is shown in table XII and table XIII. limited by the size of the train set, when the the feature are reduced to 2 dimensions and C are greater than 1, the SVM become very difficult to converge, and the cross validation can't be carried out smoothly. Thus we only use $C \in \{0.02, 0.1, 0.5\}$ to carry out the experiment.

TABLE XII
VALIDATION ACCURACY UNDER DIFFERENT C WHEN REDUCED TO 3 DIMENSIONS

C	0.02	0.1	0.5	1	10	100
RAW	87.62%	87.72%	87.73%	87.73%	87.72%	87.69%
LDA-tsne	95.28%	95.25%	95.25%	95.25%	95.24%	95.24%
PCA-tsne	88.21%	88.29%	88.33%	88.32%	88.34%	88.35%

TABLE XIII
VALIDATION ACCURACY UNDER DIFFERENT C WHEN REDUCED TO 2 DIMENSIONS

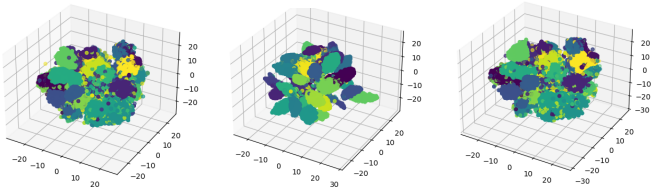
C	0.02	0.1	0.5
RAW	86.92%	86.93%	86.93%
LDA-tsne	95.31%	95.31%	95.32%
PCA-tsne	86.82%	86.85%	86.85%

Using the optimal C, we have the test Validation of different experiment under the optimal C shown in table XIV. We can

easily find that using LDA to make a pre-process before t-sne greatly increases test accuracy. This really make sense because what the LDA algorithm do is to maximize $\frac{\sigma^2_{between}}{\sigma^2_{within}}$. In other word, what LDA do make the features within the same class get closer and the features of different classes more separated. This can also be seen directly by visualization. Comparing figure 12(a), 12(b) and 12(c), we can see that features processed by LDA have the largest gap between classes and that's the reason why it has the best performance.

TABLE XIV
TEST ACCURACY UNDER OPTIMAL C OF DIFFERENT EXPERIMENT OF T-SNE

Experiment	num of dimension	Optimal C	Test Accuracy
RAW	2	0.5	87.25%
RAW	3	1	87.75%
LDA-tsne	2	0.5	95.45%
LDA-tsne	3	0.02	**95.62%**
PCA-tsne	2	0.1	87.12%
PCA-tsne	3	100	88.32%



(a) 3D visualization of compressed feature without processing (b) 3D visualization of compressed feature with LDA pre-processing (c) 3D visualization of compressed feature with PCA pre-processing

Fig. 12. 3D visualization of features compressed by t-SNE

8) Locally Linear Embedding: In this experiment, we directly use `LocallyLinearEmbedding()` function provided in `sklearn.manifold` library to implement LLE dimensionnality reduction. We have tried different paris of parameters $(neighbor, n\_component, C)$, where $neighbor$ is the number of nodes in target nodes's neighbor $\mathcal{N}$, $n\_component$ (i.e. n) is the dimensionality of features after reduced, C is a hyper parameter in SVM.

Note that LLE algorithm need to maintain a $N \times N$ matrix W ($N = 37322$ is the total number of features in the dataset) in RAM, whose size is definitely greater than the total size of our device's virtual memory. In order to avoid this problem, we reduce the size N of the original dataset by sampling 20% of features from each category and formed a new dataset with only $\frac{1}{5}$ of previous size. The following experiments are all conducted on this new dataset.

The results of our experiments with different pairs of parameters $(neighbor, n\_component, C)$ are shown in TABLE XV. Note that different from previous methods, LLE can not learn a projection matrix which suits all kinds of data. Instead, LLE have to relearn low-dimensional features given new data. This fact means that we have to put all data (train set + test set) together to learn low-dimensional features, instead of only use the train dataset.

Observing TABLE XV, we can conclude as follows:

- The performance of SVM is very poor when C is small, but the circumstance changes dramatically when C starts to grow bigger and eventually greater than $2 \sim 5$. The performance of SVMs have a big jump at $C = 2$ or 5. This phenomenon is also depicted in Fig. 13 (we take $neighbor = 20$, $n = 50$ as example).
- When $neighbor = 50$, $n = 200$ with $C = 200$, SVM achieves its optimum performance ($Accracy = 89.30\%$). We speculate that the performance of SVM can grow higher if we continue to increase the values of $neighbor$ and n.
- The performance of SVM increases monotonically with the increase of the values of $neighbor$ and n under the same C.

In order to explore why the performance of SVM jumps dramatically when $C = 2 \sim 5$, we visualize the low-dimensional features when $neighbor = 20$ and $n = 2$. The visualization is shown in Fig. 14.

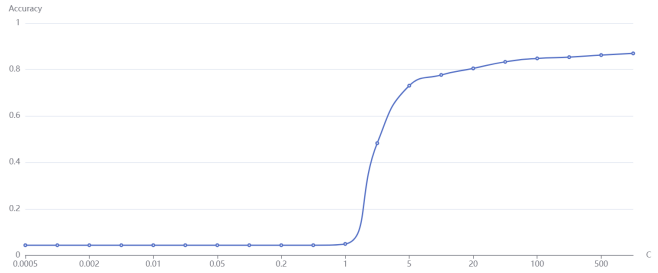


Fig. 13. [LLE] The accuracy-C curve of the SVM when $neighbor = 20$, $n = 50$

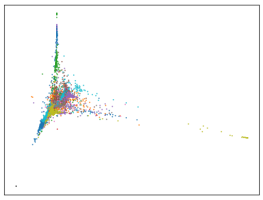


Fig. 14. The visualization of LLE-reduced features when $neighbor = 20$, $n = 2$

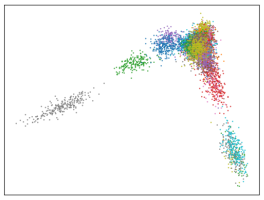


Fig. 15. The visualization of LDA-reduced features when $neighbor = 20$, $n = 2$

As we can see from Fig. 14, the LLE-reduced features of different categories mixed with each other heavily. Since SVM with large C has a greater penalty to false classifications, we think a "harder" SVM may form a "harder margin" between different categories and therefore divided them into groups better than a "softer" SVM in this case.

As a contrast, we also visualize the low-dimensional features reduced by LDA in Fig. 15. We can obviously see that

TABLE XV
THE SVM VALIDATION ACCURACY TRAINED ON DIFFERENT DIMENSIONAL (n) FEATURES REDUCED BY LDA ALGORITHM WITH VARIOUS HYPER PARAMETER C.

$neighbor$	$n\backslash C$	0.01	0.02	0.05	0.1	0.2	0.5	1	2	5	10	20	50	100	200	500	1000
5	2	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	6.73%	8.57%	7.95%	9.61%	10.81%	12.05%	12.30%	12.34%
	5	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.44%	9.19%	13.52%	13.07%	14.81%	15.98%	17.36%	20.84%	21.04%
	10	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	5.90%	5.90%	5.90%	26.79%	27.79%	29.15%	32.03%	37.25%	38.36%
	20	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	9.32%	42.42%	50.52%	56.89%	59.89%	59.89%	65.06%	68.52%	71.76%
	50	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.48%	44.86%	76.40%	78.38%	80.11%	82.97%	83.91%	84.40%	84.55%	85.11%
	100	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.86%	63.42%	81.49%	82.95%	84.15%	85.93%	85.93%	86.99%	86.88%	86.90%
	200	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	66.53%	84.08%	85.42%	86.42%	87.48%	87.79%	87.90%	87.50%	87.33%
10	2	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.75%	6.73%	6.73%	6.73%	6.73%	6.73%	6.73%	6.79%	8.63%
	5	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.75%	12.14%	12.19%	15.96%	18.05%	20.55%	20.84%	22.73%	27.72%
	10	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.99%	18.76%	29.03%	32.21%	37.11%	38.16%	45.37%	51.88%	54.63%
	20	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	10.32%	45.44%	50.19%	58.91%	66.28%	68.06%	71.48%	74.43%	75.94%
	50	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.93%	54.85%	76.54%	79.25%	82.18%	83.88%	84.88%	85.73%	86.13%	86.59%
	100	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.95%	60.78%	80.67%	83.26%	85.04%	86.50%	87.10%	87.28%	87.81%	87.92%
	200	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	5.31%	63.22%	82.77%	85.46%	86.88%	88.70%	88.72%	88.68%	88.01%	87.86%
20	2	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	6.73%	10.32%	10.32%	14.65%	14.65%	15.29%	21.35%	23.44%
	5	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	10.30%	20.04%	20.62%	23.17%	26.28%	31.79%	39.27%	39.27%
	10	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	6.99%	22.29%	31.10%	37.27%	43.42%	48.83%	54.54%	64.35%	66.30%
	20	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.42%	16.85%	16.85%	51.17%	56.87%	62.55%	67.84%	72.79%	77.03%	78.45%
	50	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.93%	48.30%	73.03%	77.65%	80.47%	83.29%	84.77%	85.31%	86.19%	86.93%
	100	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	55.69%	78.87%	81.84%	84.28%	86.30%	87.26%	87.70%	88.28%	88.12%
	200	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	5.44%	59.05%	81.64%	85.24%	87.33%	88.24%	88.55%	88.83%	88.35%	88.01%
50	2	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	7.39%	7.39%	12.14%	15.21%	15.21%	15.21%	15.21%	25.42%
	5	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	14.25%	16.76%	16.71%	16.71%	25.42%	31.76%	39.02%	44.55%
	10	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	6.99%	19.80%	30.68%	39.58%	52.32%	60.89%	65.04%	71.12%	74.05%
	20	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.42%	14.67%	45.11%	55.21%	60.84%	69.90%	75.23%	78.91%	82.02%	82.42%
	50	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	4.73%	38.11%	72.83%	78.93%	78.93%	85.13%	86.19%	87.06%	87.57%	87.59%
	100	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	5.06%	51.83%	79.58%	83.97%	86.26%	87.48%	88.48%	88.90%	89.03%	88.88%
	200	4.40%	4.40%	4.40%	4.40%	4.40%	4.40%	5.33%	55.07%	82.24%	85.99%	87.90%	88.75%	89.26%	**89.30%**	88.46%	88.17%

LDA-reduced features of different categories seperate with each other nicely (compared with LLE case) and therefore lead to a lower optimum C value (0.005). This fact is consistent with our above hypotheses.

In conclusion, the performance of SVM trained on LLE-reduced data strongly depends on the value of C. Specifically, we usually need to choose a very large $C = 200 \sim 1000$ to achieve a relatively good performance of SVM. Moreover, larger value of $neighbor$ and n often leads to higher performance under same value of C. In practice, our group recommend using $neighbor = 50$, $n = 200$ and $C = 200$ as parameters.

C. Results Conclusion

The best performance and corresponding running time of each dimensionality reduction methods are listed in TABLE XVI. Note that the running time listed in TABLE XVI contains both the time used for dimensionality reduction and the time used to train the SVM.

As we can see from TABLE XVI, algorithm PCA ($n = 1000$) achieves best classification accuracy among all kinds of method with only 0.05% difference compared with original high-dimensional features.

From another perspective, algorithm LDA runs fastest among all kinds of method, which only uses $\frac{1}{71}$ of running time compared with original features and $\frac{1}{44}$ compared with PCA. What's more, algorithm LDA has a relatively small optimal n comparing to other methods, which means that

TABLE XVI
THE BEST PERFORMANCE AND CORRESPONDING RUNNING TIME OF EACH DIMENSIONALITY REDUCTION METHODS. NOTE THAT "FGFS" IS THE ABBREVIATION OF "FASTER-GREEDY FORWARD SELECTION", "VBS" IS THE ABBREVIATION OF "VARIANCE-BASED SELECTION". "***" MEANS WE DO NOT RECORD CORRESPONDING DATA.

Method	Best Acc.	Dimension (n)	Running Time
-	93.59%	(2048)	7min 52s
FGFS	89.58%	200	2h 46min 48s
VBS	93.13%	1500	***
PCA	93.54%	1000	4min 52s
LDA	93.14%	50	**6.6s**
AE	89.20%	500	***
t-SNE (+LDA)	**95.62%**	3	6min 27s
LLE	89.30%	200	1min 1s

LDA-reduced features have smaller size and therefore requires less memory space. So in some time-sensitive or space-sensitive circumstances, it will be a better choice to use LDA algorithm rather than PCA algorithm.

Summarizing the performance of three types of dimensionality reduction methods, we have:

- **Feature selection** methods are the simplest type of method which are very easy to understand and implement. Assisted by the intuition of choosing dimensions with maximum variances from PCA, feature selection methods can achieve a relatively high performance in preserving information of original data.
- **Feature projection** methods can be regard as upgrade

version of feature selection. Method PCA performs the highest classification accuracy among all feature projection methods, while LDA has the best time and space property with a well-performed accuracy.

- **Feature Learning** methods are the most special type of method among the three. Method t-SNE (with LDA processing) achieves the best accuracy among all kinds of methods. However, because feature learning methods (e.g. t-SNE, LLE) usually cannot learn a projection matrix W to depict how to project original high-dimensional data into a lower-dimensional space, we have to relearn the projection pattern every time we add some new data into our dataset. This property limits the expansibility of the feature learning methods. What's more, feature learning methods usually consumes a huge scale of memory when the size of dataset are big, which means that feature learning methods have a relatively high requirement on the computing device. Due to these two shortcomings, we only recommend feature learning methods for tasks with relatively small and static dataset.

IV. Conclusion

In this paper, we introduced and implemented totally 7 dimensionality reduction methods, namely *Faster-Greedy Forward Selection*, *Variance-Based Selection*, PCA, LDA, Auto-Encoder, t-SNE and LLE. We evaluated the performance of these methods by training SVM on features reduced by them and comparing the accuracy of each SVM. Our experiments show that t-SNE with LDA preprocessing achieves the highest accuracy, while LDA has the best time and spatial properties.

Acknowledgment

This work is supported by Professor Niu and all teaching assistants in class CS245. We thank Teng Hu, Shengran Cheng, Shuhao Deng for valuable discussion. We greatly thank Professor Wang for the computing equipment provided by his lab.

References

[1] Y. Xian, C. H. Lampert, S. Bernt, and A. Zeynep, "Zero-shot learning - a comprehensive evaluation of the good, the bad and the ugly," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. PP, no. 99, pp. 1–1, 2017.