Project1: Dimensionality Reduction

Jizheng Chen 520021911182

March 20, 2023

1 Introduction

In machine learning and Artificial intelligence, feature extraction is a very important part before model construction and training. A feature is an individual measurable property or characteristic of an object, event, or phenomenon that is used as input to a machine learning algorithm to make predictions or decisions. Features can be thought of as the building blocks of a machine learning model, as they provide the information or signal that the model uses to learn from data and make predictions.

1.1 Curse of Dimensionality

However, when the dimensionality of a sample is too large, a lot of problems will arise. The curse of dimensionality is a phenomenon that occurs when the number of input features or dimensions increases. In high-dimensional data, the space becomes increasingly sparse and the number of observations required to estimate a function accurately grows exponentially. This leads to problems such as overfitting, computational complexity, and reduced predictive accuracy. So it's essential to use feature reduction methods to reduce the dimensionality of feature space.

1.2 Feature Reduction

Feature reduction, also known as dimensionality reduction, is a set of techniques used to address the curse of dimensionality by reducing the number of input features while retaining the most relevant information.

There are three main types of feature reduction techniques: **feature selection**, **feature projection** and **feature learning**.

Feature selection methods select a subset of the original features based on some criteria such as feature importance, correlation with the target variable, or mutual information. Examples of feature selection techniques include forward/backward feature selection, lasso regularization, and mutual information-based feature selection.

Feature projection methods transform the original features into a lower-dimensional space using linear or nonlinear transformations. These methods aim to capture the most important patterns or structures in the data while discarding irrelevant or noisy information. Examples of feature extraction techniques include principal component analysis (PCA), independent component analysis (ICA), and autoencoder neural networks.

Feature learning methods is a set of techniques used to automatically discover and extract relevant features from raw data. Unlike traditional feature engineering, which relies on domain knowledge and human expertise to select or design features, feature learning algorithms use unsupervised or supervised learning methods to learn representations of the input data directly from the raw input.

All of the three kinds of approches can help reduce the curse of dimensionality by reducing the number of input features required to accurately model the data. However, it's important to note that these techniques can also introduce some trade-offs such as loss of interpretability, increased computational complexity, or increased risk of overfitting. Therefore, the choice of a particular feature reduction technique should be carefully evaluated based on the specific problem and the available data. In the following parts, I compared several feature reduction methods to explore the advantages and disadvantages of the three methods.

2 Dimensionality Reduction Methods

In this section, several dimensionality reduction methods are applied to reduce input dimension to svm. I choose **Forward Selection** for feature selection method, **PCA** and **AutoEncoder** for feature projection method, and **Local Linear Embedding** for feature learning method.

2.1 Forward Selection Algorithm

Forward feature selection is a method used in machine learning to select a subset of relevant features from a larger set of input features. The algorithm works by iteratively adding one feature at a time to the model, evaluating the performance of the model using a chosen metric, and selecting the feature that gives the best performance improvement. This process continues until no further improvement can be made by adding additional features, or the performance meets our standard.

The performance of the model can be evaluated using any suitable metric, such as accuracy, precision, recall, or F1 score. In this case it's the final performance of our SVM model on validation set. In my experimental setting I want to kept the final performance superior to 0.85, so features are continuously added until the performance reaches the standard. The stopping criterion can be based on a fixed number of selected features, a certain level of performance improvement, or a threshold on the feature importance.

The algorithm of Forward Selection is illustrated in algorithm 0.

Algorithm 1 Forward Feature Selection

```
1: Start with an empty set of selected features.
2: for each feature $f$ in the input feature set do
3:     Evaluate the performance of the model using feature $f$.
4:     Select feature $f$ as the best performing feature.
5:     Add feature $f$ to the set of selected features.
6: end for
7: while performance improvement is observed do
8:     Evaluate the performance of the model using all possible combinations of the selected
       features and each remaining feature.
9:     Select the feature that gives the best performance improvement and add it to the set of
       selected features.
10: end while
```

2.2 Principal Component Analysis

PCA (Principal Component Analysis) [4] is a popular technique for dimensionality reduction and feature extraction in machine learning and data analysis. It is used to transform a high-dimensional dataset into a lower-dimensional space while retaining most of the original information and minimizing the loss of information due to the dimensionality reduction.

The goal of PCA is to find a set of orthogonal (uncorrelated) vectors, called principal components, that capture the maximum amount of variance in the data. The first principal component captures the largest amount of variance, the second principal component captures the next largest amount of variance, and so on. Each principal component is a linear combination of the original features in the data, and the coefficients of these combinations are called the loadings of the principal component.

PCA works by computing the eigenvectors and eigenvalues of the covariance matrix of the input data. The eigenvectors represent the principal components, and the eigenvalues represent the amount of variance captured by each principal component. The eigenvectors with the largest eigenvalues are retained as the principal components, and the data is projected onto the subspace spanned by these eigenvectors.

specifically, let $X = \{x_1, x_2, ..., x_n\}$ be the matrix of our dataset, and we need to reduce x_i's dimension to k, to get the major component we need to do eigen decomposition on XX^T, i.e:

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

Then sort the eigenvalues and get the max k corresponding eigenvectors, let P be the concatenation of those eigenvectors, the dimensionality projection can be formulated as:

$$Y = PX \tag{2}$$

2.3 AutoEncoder

An autoencoder [1] is a type of neural network that is commonly used for unsupervised learning, dimensionality reduction, and feature extraction. The basic idea behind an autoencoder is to learn a compressed representation of the input data by training a neural network to encode the input data into a lower-dimensional space and then decode it back to the original form.

The autoencoder consists of two main parts: an encoder network that maps the input data to a compressed representation, and a decoder network that maps the compressed representation back to the original form. The encoder network typically consists of one or more hidden layers that transform the input data into a lower-dimensional representation, while the decoder network consists of one or more hidden layers that transform the compressed representation back to the original form. The objective of the autoencoder is to minimize the difference between the original input data and the reconstructed output data, using a loss function such as mean squared error or cross-entropy.

In my case, the structure of both encoder and decoder is a three-layer fully connected network that transform original feature dimension of 2048 bit to a bottleneck of a lower dimension. It's details will be illustrated in following sections.

2.4 Local Linear Embedding

Local Linear Embedding (LLE) [2] is a non-linear dimensionality reduction technique that aims to preserve the local structure of the data in the lower-dimensional space. The basic idea behind LLE is to model the local relationships between the data points using linear approximations and then use these approximations to map the data to a lower-dimensional space.

The algorithm of LLE is at algorithm 0.

Algorithm 2 Local Linear Embedding (LLE)

```
    Input: Data matrix $X \in \mathbb{R}^{n \times d}$, number of neighbors $k$, target dimensionality $d'$
 2: Output: Embedded data matrix $Y \in \mathbb{R}^{n \times d'}$
    Step 1: Construct the neighborhood graph
 4: Compute the pairwise distances between the data points
    For each data point, find its $k$ nearest neighbors based on the distances
 6: Construct the graph with the data points as nodes and the edges connecting each point to
    its $k$ nearest neighbors
    Step 2: Compute the local linear relationships
 8: For each data point $x_i$, find its $k$ nearest neighbors $N_i$
    Compute the weights $w_{ij}$ that best approximate $x_i$ as a linear combination of its neighbors
    $x_j$ using least squares regression:
10:     $W_i = \texttt{argmin} w_i |x_i - \sum j \in N_i w_{ij} x_j|^2$
    Store the weights in a matrix $W \in \mathbb{R}^{n \times k}$
12: Step 3: Embed the data in the lower-dimensional space
    Compute the matrix $M = (I - W)^T (I - W)$
14: Compute the $d' + 1$ smallest non-zero eigenvalues $\lambda_1, \dots, \lambda_{d'+1}$ and corresponding eigenvec-
    tors $v_1, \dots, v_{d'+1}$ of $M$
    Discard the eigenvector corresponding to the smallest eigenvalue, which is 1
16: Normalize the remaining eigenvectors to unit length
    Construct the embedded data matrix $Y = [v_1, \dots, v_{d'}]$
18: return $Y$
```

3 Experiment

I have implemented the methods mentioned above on AwA2 animal dataset based on an SVM using linear kernel. The original dimension is reduced to several lower settings by those methods seperately, on which the SVM gets trained and evaluated. In the following sections I'will first explain the experimental settings and the details of my methods, before giving the final results of the dimension reduction.

3.1 Experimental setting and details

3.1.1 Dataset

The SVM and dimensionality reduction methods are trained and evaluated on AwA2 dataset, which is a widely used benchmark dataset in the field of computer vision and machine learning. The dataset consists of 37322 images of 50 different animal classes, each annotated with 85 binary attribute labels. Here I directly used the features extracted by a Resnet101 [3], which is a vector of length 2048 for each individual picture sample.

The original dataset is divided into a training set consisting 60% of data, and a testing set of 40% data. To decide the value of C in SVM setting, a **3-fold cross validation** is applied further on the training dataset. Finally a value C=2 is choosen with a satisfying performance.

3.1.2 Reduction Details

In this section some implementation details are given based on dimensionality reduction methods mentioned above. Code is available at https://github.com/Otsuts/PDS-projects

- Plain SVM: 2048 features of an individual sample is directly trained and tested on an SVM with C=2 and linear kernel.
- Forward selection: A standard of 75% precision is required when features are choose from scratch. Instead of adding features randomly, I used **chi-square** distance to choose better features first. When a certain number of features are choosen, another SVM is trained using 80% of train dataset and 20% of train dataset to validate. When the precision is superior to the threshold, the process haults.
- AutoEncoder: It consists of a three-layer encoder and a three-layer decoder, hidden layers' neural number are 2 times, 4 times, and 8 times of the final hidden size. It is also trained on 80% of training set and validate on the 20% validation set. Finally the encoder is used to reduce the dimension.
- PCA: Used as a furthor comparison to AutoEncoder so as to see the difference between traditional machine learning method and deep learning method.
- LLE: I choose a neighbor number of 30 and iterated for 100 times.

	Precision	Reduction Time	SVM Train Time	SVM Inference Time
Plain SVM	0.927	0	51.485	132.305
SFG	0.792	0.448	5.432	9.520
AutoEncoder	0.816	59.958	4.346	12.358
PCA	0.888	0.965	10.773	6.833
LLE	0.600	662.506	25.794	24.967

Table 1: Results: Reduction of 32 dims

	Precision	Reduction Time	SVM Train Time	SVM Inference Time
Plain SVM	0.927	0	51.485	132.305
SFG	0.858	0.406	4.890	8.703
AutoEncoder	0.873	53.613	3.569	9.468
PCA	0.903	0.902	7.613	8.107
LLE	0.710	676.690	29.866	29.060

Table 2: Results: Reduction of 64 dims

3.2 Experiment Results

In this section I'll display the results gotten using the above dimensionality reduction method. All methods are tried on 32, 64, 128 and 256 dimension. Note that I also tried SNE, but it only accept a dimension inferior to 4 in sklearn package, so I did't take it into account when doing experiments.

The following table records the presicion each methods get on different dimensions along with the time consumption on dimension reduction, SVM training and SVM prediction. Every single experiment has been done for 3 times and calculated the average result.

3.3 Analysis

The following graphicx show the precision 1 and time consumption 2 in training SVM of each dimensionality reduction method with different reduced dimension. As the reduction times varies a lot concerning on different methods, the reduction time is not demonstrated in the graph.

First we can draw the conclusion that features from Resnet101 is well-extracted. The plain SVM without any dimensionality reduction method becomes the state-of-the-art method, getting over 92% precision. And other reduced dimensionality damaged the performance to some extent.

	Precision	Reduction Time	SVM Train Time	SVM Inference Time
Plain SVM	0.927	0	51.485	132.305
SFG	0.883	0.443	5.393	12.330
AutoEncoder	0.887	93.332	4.768	10.760
PCA	0.906	1.331	7.352	13.813
LLE	0.795	683.980	38.913	38.486

Table 3: Results: Reduction of 128 dims

	Precision	Reduction Time	SVM Train Time	SVM Inference Time
Plain SVM	0.927	0	51.485	132.305
SFG	0.900	0.464	4.932	19.979
AutoEncoder	0.899	188.504	5.726	15.462
PCA	0.913	1.441	5.563	21.957
LLE	0.833	679.298	54.099	50.080

Table 4: Results: Reduction of 256 dims

Second, PCA get optimal performance in both low dimension of 32 and higher dimension of 256, and compared with deep learning methods like autoencoder, it doesn't take much time to train the network. The reason why it gets good results may be that the extracted feature is well fitted in the 2048-dim latent space, and after PCA process it will be better recognized and categorized by a regression model like SVM.

The forward selection method is less well performed, that's because the feature selection process does not have a good heuristic feature decision method, so the reduced dimension may not able to represent the sample well. But we can see that as the dimension gets bigger, performance catches up.

Finally, LLE method does not perform well, a possible reason is that this method uses neighborhood information a lot, but the sample feature doesn't have a clustering distribution. So the feature dimension may not be well extracted and represented.

As for the optimal dimensionality, we can see that with the dimension changes from 32 to 64, the SVM training time decreases, and from 64 to 256, the training time increases for a bit. That's because of the tradeoff between the input dimension and the converge speed. More time will be comsumed as the dimension becomes larger, but more information also helps the SVM model to converge faster. 64-dimension may be the balance point, and the reduction time is satisfying. So I'll argue that 64-dim is the optimal dimensionality.

4 Conclusion

From this project, I implemented different methods of feature dimension reduction, learned their advantages, disadvantages and time performance, and thus enhanced my coding and hands-on skills. In addition, I also learned that different application scenarios require different adaptation methods, and a good feature extractor can help improve the overall training effect

References

[1] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *arXiv preprint arXiv:2003.05991*, 2020.

[2] Jing Chen and Yang Liu. Locally linear embedding: a survey. *Artificial Intelligence Review*, 36:29–48, 2011.

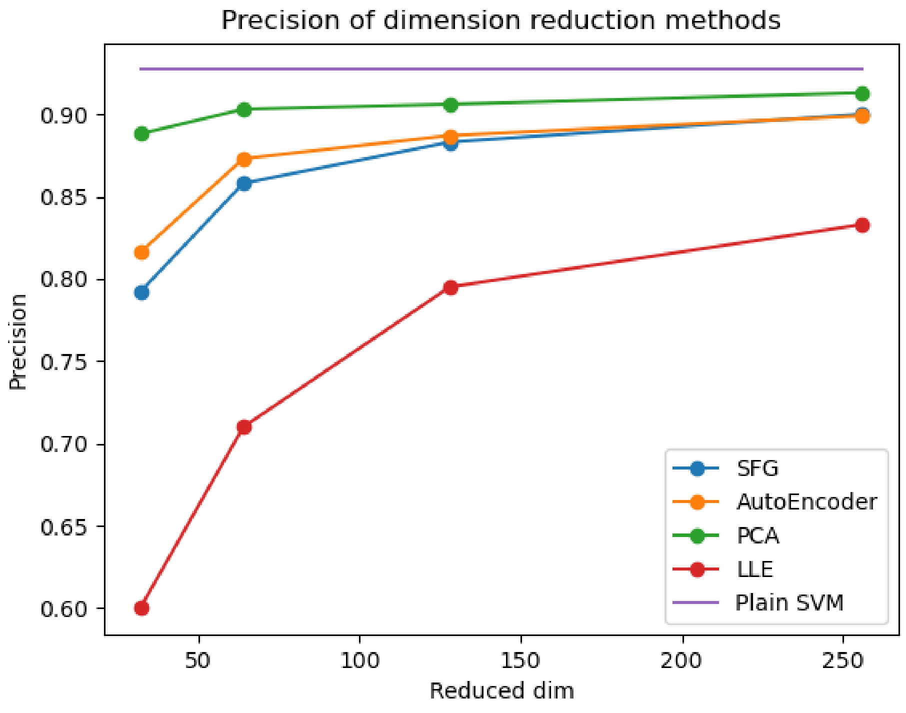

Figure 1: Precision with dimension

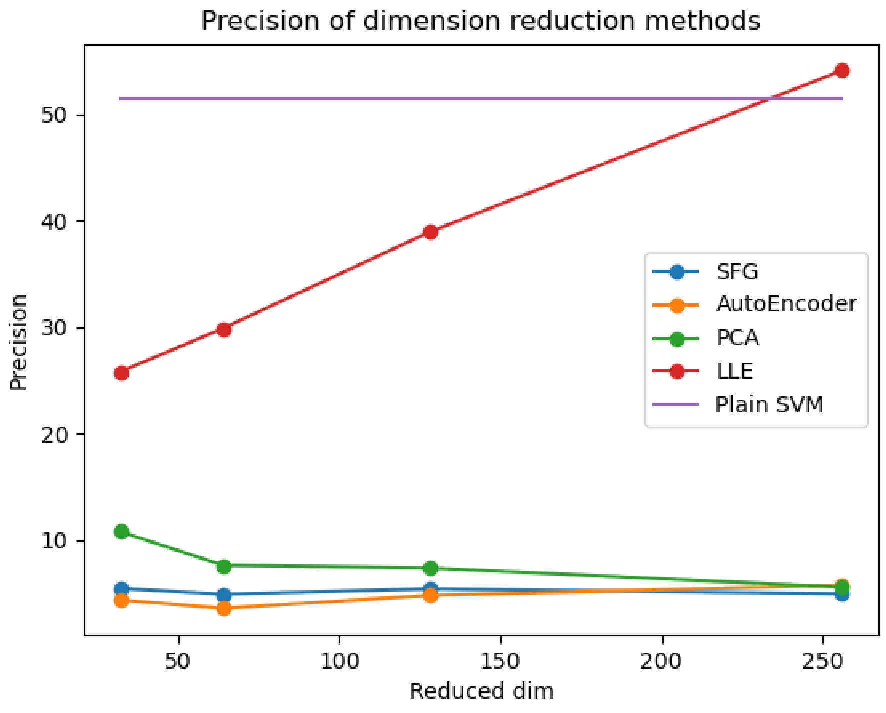

Figure 2: Training time with dimension

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[4] Andrzej Maćkiewicz and Waldemar Ratajczak. Principal components analysis (pca). *Computers & Geosciences*, 19(3):303–342, 1993.