Open in app

uday sai

4 Followers About

Dimensionality Reduction : SVD, PCA and LLE

uday sai May 1, 2020 · 5 min read

Machine Learning projects often consists of millions of features for each training instance. These features not only make the training the model slow but also thwarts us from obtaining a good solution. This is referred as Curse of Dimensionality

Approaches of Dimensionality Reduction

> *Projection*
>
> *Manifold Learning*

Projection

The rationale behind this approach is the features are not uniformly spread around the high-dimensional space as some of the features are constant and some are highly correlated. So these features are often concentrated in low-dimension subspace. This intuition triggers to a projection in which the highly concentrated features are projected into a low-dimensional space with a minimal information loss. However projection is not

Manifold Learning

A d-dimensional manifold is a part of an n-dimensional space (where d<n) that locally resembles a d-dimensional hyperplane. Many dimensionality reduction algorithms work by modeling the manifold on which the training instances lie; this is called Manifold Learning. It relies on the manifold assumption, also called the manifold hypothesis, which holds that most real-world high-dimensional datasets lie close to a much lower-dimensional manifold. This assumption is very often empirically observed.

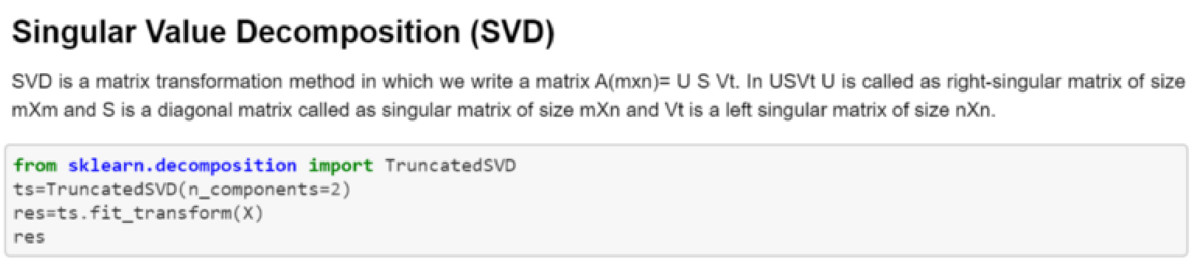
The manifold assumption is often accompanied by another implicit assumption: that the task at hand (e.g., classification or regression) will be simpler if expressed in the lower-dimensional space on the manifold. However, this assumption does not always hold. In a nutshell dimensionality reduction definitely speeds up training but it does not always guarantee a better or simpler solution.

Let us explore some of the algorithms for Dimensionality Reduction

Singular Value Decomposition (SVD)

This method is based on matrix factorization technique which implies that any matrix 'A' can be represented as USVt i.e., A=USVt in which 'U' is a left singular matrix, S(pronounced as sigma) is a singular matrix and Vt is a right singular matrix. Besides 'U' and 'Vt' are ortho-normal matrices and 'S' is a singular matrix and contains all the singular values. To reduce dimensions we apply SVD on original dataset and choose the desired number of singular values for instance 'k'.

Truncated SVD is a special case of SVD in which we choose 'k' values and make the rest of singular values 0. This truncated SVD is available in Sklearn library. The following is a case in which I applied TruncatedSVD to wine dataset (available in GitHub ,kaggle). Notice is that 'X' is not centered in the case of SVD.



Singular Value Decomposition (SVD)

SVD is a matrix transformation method in which we write a matrix A(mxn)= U S Vt. In USVt U is called as right-singular matrix of size mXm and S is a diagonal matrix called as singular matrix of size mXn and Vt is a left singular matrix of size nXn.

```
In [4]: from sklearn.decomposition import TruncatedSVD
        ts=TruncatedSVD(n_components=2)
        res=ts.fit_transform(X)
        res
```

Image : TruncatedSVD

Principal Component Analysis (PCA)

The intuition behind PCA is that the data is linearly transformed on to a lower-dimensional subspace with maximum variance and minimum sum of squares errors from original points to transformed points. The lower-dimensional axis are called as "Principal Components (PC's)" and are orthogonal to each other. PCA can be applied on top of SVD are directly from sklearn. If we are applying PCA through SVD we need to make sure to transform our feature matrix (let us call it as 'X') to be centered. This can be applied in Python as below

Image : PCA using Numpy

Image : PCA using sklearn

In the above code snippet we can see that explained_variance_ratio_ attribute is used to check the variance explained by the transformed features. In this case the first PC exhibits a variance of almost 99% which is awesome. So instead of using the entire dataset we can just use this single column to predict 'Wine Class' (the target variable) which enormously brings down the training time. This will be extremely useful in case of datasets with high dimensions.

To Choose the n_components which means the number of principal component axes to choose we can simply specify n_components=0.95 which means to retrieve principal components which explains 95% of variance in original dataset. In our case this will

Image: Elbow Method

PCA for Compression

We can attempt to reconstruct the original dataset from the reduced matrix (the matrix obtained after applying PCA) which can be accomplished by using inverse_transform method of PCA. However, the obtained matrix is not exactly same as the original dataset and the error between the obtained matrix and original is called reconstruction error.

Incremental PCA

This variant of PCA is used in the case of large datasets which does not fit into memory. The data is fed into algorithm batch-wise and at the end of last batch we train the algorithm completely.

Image : Incremental PCA

Randomized PCA

This is a stochastic algorithm used to find first ‘d’ principal components in a relatively faster way to other variants. The computation complexity is very minimal compared to

Open in app

We have already discussed that PCA is a linear transformation technique. How about the case of non-linear transformation? — Well we have a workaround for this which is mathematically called a kernel technique. In this technique we map instances to a higher-dimensional space (feature space) enabling non-linear transformation. KernelPCA is available in sklearn with kernel "rbf" and "sigmoid". Note that in KernelPCA the fit_inverse_transform method must be marked as "True" to apply it.

Image : Kernel PCA

Locally Linear Embedding (LLE)

LLE is a manifold and non-linear dimensionality reduction technique. In this technique we first choose 'k' (let us say k=10) and reconstruct an instance of a dataset xi (x subscript i) as linear function of all the xk (meaning 10 closest neighbors) of xi. The algorithm then looks for a low-dimension representation while trying to preserve aforementioned relationship. This lower-dimension is the manifold which represents the maximum variance of the dataset. This technique is available in sklearn as sklearn.manifold.LocallyLinearEmbedding

Please refer my GitHub profile in references for the entire code

References

github.com

Hands-On Machine Learning with Scikit-Learn and Tensorflow by Aurelien Geron

Machine Learning Data Science Dimensionality Reduction Principal Component Svd