Dimensionality Reduction

In machine learning, we often encounter datasets which has huge number of features based on which, analysis is done. But there are many downfalls associated with this high dimensional data:
1) It is very hard to visualize dataset with too many features. Also, computational power and time will also be high in this case.
2) More features mean complex model and therefore more chances of overfitting.
3) More storage is required.
4) In order to avoid overfitting, we will be required huge number of records.

Therefore, dimensionality reduction is used to reduce the number of features in dataset without losing much information. Sometimes, many features are correlated and are redundant which we can remove.

There are two methods of dimensionality reduction:

1) Feature Selection:

In this method, we try to find subset of features from the original feature set. We can perform this method using any of the three ways-

a) Filter methods:

This method apply a threshold on a statistical measure to determine whether a feature will be useful. It includes-
* Variance threshold: It remove any features that do not vary widely.
* Correlation threshold: It remove highly correlated features.

Filter methods require manual setting of threshold which is not preferred as it can remove important information if threshold is very low.

b) Wrapper methods:

This method iterates through different combinations of features and train model on each. The feature combination which resulted in the best model performance metric (accuracy, AUC, RMSE, etc.) is selected. There are two ways for this method-

* Forward selection: Begin with zero features in your model and iteratively add the next most predictive feature until no additional performance is reached with the addition of another feature.
* Backward selection: Begin with all features in your model and iteratively remove the least significant feature until performance starts to drop.

These methods take a long time to compute and can result in overfit model also.

c) Embedded methods:

These methods perform feature selection during the model training, which is why we call them embedded methods. While training the model, these methods derive feature importance i.e. how much is feature important when making a prediction and then remove non-important features. There are various ways to perform this:

* Regularized Linear Regression: These methods add a penalty that corresponds to the value of the coefficients. This way, coefficients with lower predictive power are weighted less heavily, and the overall variance of the dataset is reduced. There are three key types of regularized linear regression:

Lasso Regression (L1 regularization): The penalty is equivalent to the absolute value of the coefficient. This means that coefficients of unimportant features can be reduced to zero, which effectively eliminates them from the model.

Ridge Regression (L2 regularization): The penalty is equivalent to the square of the coefficient value. Because coefficient values are squared, coefficients cannot be eliminated from the model, but can be weighted less heavily. It does not set the coefficient to zero, but only approaching zero.

Elastic nets (L1/L2 regularization): It is a combination of the L1 and L2. It incorporates their penalties, and therefore we can end up with features with zero as a coefficient (similar to L1).

* Regularized Tree: When building tree-based models, we can use metrics like entropy, gain or Gini impurity to measure feature importance and can then elect to only keep the top N most important features. Random forests provide us with feature importance using straightforward methods — **mean decrease impurity** and **mean decrease accuracy**. When training a tree, feature importance is calculated as the decrease in node impurity weighted in a tree.
The **higher** the value, the more **important** the feature.

2) Feature Extraction:

With this technique, we generate a new feature set by extracting and combining information from the original feature set. For example, converting dataset from high dimensional space to 2-D or 3-D. It allows us to keep parts of features, so that we just keep the most helpful bits of each feature.

We have many methods for feature extraction. Some of which are factor based and some are projection based.

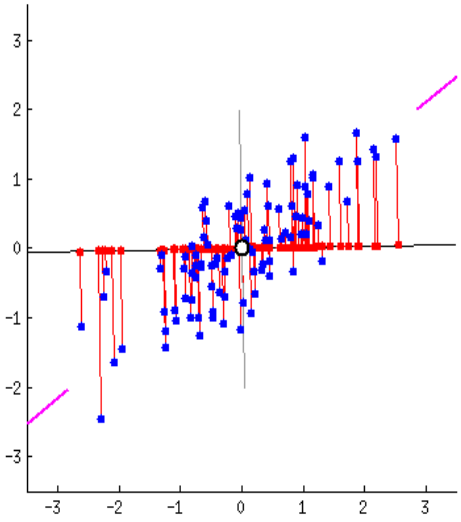
(a) Factor based: Factor Analysis, *PCA*, ICA
(b) Projection based: ISOMAP, UMAP, *TSNE*

PCA

PCA (Principal Component Analysis, unsupervised and linear method): PCA works by trying to maximize the variance in data while reducing the dimensionality of it. It converts the data into linearly uncorrelated variables called principal components. For exp, if we would like to have two-dimensional transformed data, we would use the two principal components with the largest variance to reveal the structure in the data.

The principal components are the eigenvectors of a covariance matrix, and hence they are orthogonal. The eigenvectors of the covariance matrix are actually the direction of axes where there is most variance (most information).

let's assume that the scatter plot of our data set is as shown below, can we guess the first principal component? Yes, it's approximately the line that matches the purple marks because it goes through the origin and it's the line in which the projection of the points (red dots) is the most spread out or we can say it's the line that maximizes the variance (the average of the squared distances from the projected points (red dots) to the origin). The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.



Steps for implementing PCA:

Step 1) Normalize the data:

The dataset on which PCA technique is to be used must be scaled. The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis. If there are large differences between the ranges of initial variables, those variables with larger ranges will dominate over those with small ranges (For example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.

This can be done by subtracting the mean and dividing by the standard deviation for each value of each variable.

$$z = \frac{value - mean}{standard\ deviation}$$

Step 2) Calculate the covariance matrix

The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them.

For 3D, covariance matrix will be:

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

In above matrix, If Cov() value is positive, the two variables are directly correlated.
If Cov() value is negative, the two variables are inversely correlated.

Step 3) Calculate the Eigen Value and Eigen Vector
we need to compute eigen value and eigen vector from the covariance matrix to determine the ***principal components*** of the data. Eigenvalues are simply the coefficients attached to eigenvectors, which give the *amount of variance carried in each Principal Component*. By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.

Step 4) Feature vector
If we have a dataset with *n* variables, then we have the corresponding n eigenvalues and eigenvectors. To reduce the dimensions, we choose the first *p* eigenvalues and ignore the rest. We do lose out some information in the process, but if the eigenvalues are small, we do not lose much. If we choose to keep only ***p*** eigenvectors (components) out of ***n***, the final data set will have only ***p*** dimensions.
Next we form a feature vector which is a matrix of vectors (eigenvectors).

Step 5) Forming Principal Components
To form principal components, we take transpose of the original dataset and multiply it with transpose of the feature vector.

NewData = $FeatureVector^T$ x $ScaledData^T$
Here, scaled data is scaled version of original dataset.

TSNE

TSNE (t-distributed Stochastic Neighbor Embedding, an unsupervised and non-linear method):

This method works in two steps:
1) For high dimensional dataset, it measures the similarity between each pair and create similarity matrix (S1). Then it converts that similarity distance to probability (joint probability) according to the normal distribution. It means similar objects are assigned higher probability while dissimilar points are assigned a lower probability.
2) Now, TSNE arranges all of the data points randomly on the lower dimensional space and does the same calculation (calculate similarity distance) but for lower dimension, it assigns probability according to t-distribution instead of normal and thus similarity matrix (S2) will be formed. Now, t-SNE compares S1 and S2 and tries to minimize difference (Kullback-Leibler divergence) between these two matrix.

Simply the goal of the procedure is to find a mapping onto the 2-dimensional space that minimizes the differences between these two distributions over all points. In high dimensional space, this is modeled as a gaussian distribution but in lower dimensional space, this is modeled as a t-distribution. Therefore, it is known as t-SNE.

Advantages of t-SNE:
1) Handles nonlinear data efficiently.
2) Preserves local and global structure.

Disadvantages of t-SNE:

1) Computationally complex
2) Non-deterministic
3) Requires hyperparameter tuning
4) Noisy pattern

PCA v/s TSNE

PCA is linear technique that seeks to maximize variance and preserve large pairwise distances. While t-SNE is concerned with preserving only small pairwise distances or local similarities.