CS 558 Project Notes

Dimensional Reduction and Principal Component Analysis

REFERENCE: <https://hackernoon.com/supervised-machine-learning-dimensional-reduction-and-principal-component-analysis-614dec1f6b4c>

In short, the more dimensions the training set has, the greater the risk of overfitting our model.

There are two primary approaches towards dimensional reduction, Projection and Manifold Learning.

Projection – Reducing an n dimensional set of information to a dimension less than n without loosing too much information. This can be done because in many training sets, there are elements that are highly correlated or the data is constant around a certain feature (such as planer or linear)

Manifold Learning – When you bend and twist a high-dimensional space such that is can be mapped to a lower-dimensional space where the high-dimensional space can locally be resembled in a low-dimensional space. Most popular method employed in ML. It enables the algorithm to find the optimal manifold in reducing the dimensionality of our data set.

The weakness of Manifold Learning is often the lower-dimensional subspace is not simpler.

Principle Component Analysis – PCA is a dimensionality reduction method whose objective is to identify a hyperplane that lies closest to the data points and then project the data onto it. In PCA the “right” hyperplane is, in most cases, determined by the axis through our data set that preserves the maximum amount of variance.

The axis that preserves the maximum amount of variants is referred to as the Principal Components (PCA1). A second Principal Component, orthogonal to the first, is then chosen. This is called PCA2 and it accounts for the largest remaining variance in the dataset.

Finding Principal Components – The most common method to find principle components is known as Singular Value Decomposition (SVD). This is a basic matrix factorization technique where you decompose the training set matrix into the dot product of three matrices.

M = U dot Σ dot V\* (dot is the dot product)

M = m × n matrix whose entries come from a field, where either field consists of real numbers or complex numbers.

U = a m × m unitary matrix. (left singular vector)

Σ = m × n diagonal matrix with non-negative real numbers.

V = n × n unitary matrix . ( right singular vector)

V\* = conjugate transpose of the n × n unitary matrix.

Implementation can be done by importing the function PCA from Sci-Kit Learn

from sklearn.decomposition import PCA

pca = PCA(n\_components = 2)

X2D = pca.fit(X).transform(X)

After fitting the dataset, we can transform our data set into a Pandas DataFrame Object, with the respective labels (PCA1, PCA2)

df = pd.DataFrame(X2D,columns = ['PCA1','PCA2'])

Then we can plot the Principal Components onto a scatterplot.

If we want to better understand the variance, we can invoke a function

pca.explained\_variance\_ratio  
array([0.82,0.15])

The answer indicates that 82% of the variance lies along the first axis and 15% lies along the second axis with the remaining variance captured in other Principal Components.

This works well, when we are looking to perform data visualisation tasks in our high dimensional dataset. However, when we are modelling, we often want to capture a certain proportion of the variance within our training set.

As a general rule: It is often preferred that we capture at least 95% of the variance within our training set.

To achieve this, we simply change our n\_components parameter from an integer to a float between 0.0 to 1.0, indicative of the ratio of variance we would like to capture:

pca = PCA(n\_components = 0.95)

X = pca.fit\_transform(X\_train)