Dimensionality Reduction with PCA
AI/ML Guild Lab #4
March 25, 2021

1 Introduction

Often your data isn't very nice to you. Sometimes, the dimensions are just too damn high! They are hard to vizualize, there are too many features and many seem intertwined. Principal Component Analysis is an unsupervsied technique that can help resolve theses issues and can aid in model building and data visualization.

1.1 The Curse of Dimensionality

"As the number of features or dimensions grows, the amount of data we need to generalize accurately grows exponentially." – Charles Isbell

Having a large feature set sounds good on paper, but it also has the consequence of requiring more data to train a model. A common rule of thumb is that there needs to be 5 training points per dimension (feature).

PCA is one way to reduce the number dimensions (thus also reducing the amount of data needed to train a good model.) Reducing the number of features down to a core set that provides the most information (most variance) is called *Feature Selection*.

2 Principle Component Analysis

There are several different ways to derive PCA, this is going to go over the *Maximum Variance* approach. (Another way is via *Minimum Error*.)

PCA can be defined as the orthogonal projection of data onto a lower dimensional linear space such that the variance of the projected data is maximized.

PCA is computed by evaulating the mean $\bar{x}$ and the covariance matrix S of the data set and then calculating the M eigenvectors of S that correspond to the M largest eigenvalues.

Ok.

Let's go over some of the tools for that.

The mean of our data:

$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N}(x_n)$$

That's pretty simple.

The variance of our data:

$$\sigma^2 = \frac{\sum_{n=1}^{N}(x_i - \bar{x})^2}{n}$$

i.e. The variability from the mean.

The covariance of our data:

$$S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^{\mathbf{T}}$$

Measures how different variables/features/axes vary jointly. i.e. greater values in one feature, corresponds to greater values in another feature.

The resulting matrix looks like this:

$$S = \begin{pmatrix} \sigma(x_1, x_1) & \dots & \sigma(x_1, x_n) \\ \vdots & \ddots & \vdots \\ \sigma(x_n, x_1) & \dots & \sigma(x_n, x_n) \end{pmatrix}$$

The diagonal entries of the covariance matrix are just the variances for that data point.

To get from our covariance matrix to principal components, we have to use Lagrangian multipliers.

Along the way we would calculate eigenvectors and eigenvalues. These aren't too difficult to understand. If T is a linear transformation and v is non-zero vector, then v is an eigenvector of T if

$$T(v) = \lambda v$$

λ is a scalar and is the eigenvalue associated with v.

Which if you read the article on Lagrangians you'll see the main formula defined as:

$$\mathcal{L}(x, \lambda) = f(x) - \lambda g(x)$$

If you know we're looking for a set of eigenvector and eigenvalues, you can imagine that we're likely solving for multiple Lagrangians iteratively to get to the top M principal components. What we're maximizing for with the Lagrangian is the variance of the feature. In the end of this process, we'll have a set of dimensions, corresponding to features, sorted by variance.

Luckily, there are many libraries available that do these computations for us.