Voice Command Classification with PCA

Introduction[1]

In order for S1XT33N to be able to differentiate we need to implement a classifier that is able to distinguish between 4 sufficiently different commands. Recorded audio signal necessarily has some amount of approximately white noise, and therefore it is highly unlikely that you can generate two exactly similar looking wave-forms. Therefore we need to measure the similarity of two waveform. A waveform with 100 time steps can be represented as a 100 dimensional vector. Ideally we can record a lot of wave forms of the same keywords, and compute their means. That is a 100 dimensional vector representing each word. Then during test time, when a new word is recorded we can measure the euclidean distance to different means and classify the test word to its closest neighbour.

However if we want to do this computation on launchpad we will run out of space. That is why we study Principle Component Analysis (PCA) to compress our vector sizes. Any Compression is a projection from original space R^d to a lower dimensional space R^k (i.e. $k < d$)and comes with information loss. PCA is a **linear** projection which minimizes the amount information loss in the process. Because PCA is an application singular value decomposition (SVD) we will study that as well.

Linear Algebra Jargon Refresher

1. **Covariance matrix**: Covariance quantifies the joint linear variability between two random variables X and Y and is calculated as:

$$cov(X,Y) = \mathbb{E}[(X-\bar{X})(Y-\bar{Y})]$$

A covariance matrix is a square matrix of pairwise covariances of features from a data matrix X (n samples x m dimensions, think about our sound waveforms as an example) and it is computed as $\frac{1}{n-1}(X-\bar{X})^T(X-\bar{X})$ where $\bar{X}$ is vector corresponding to the average of data across all samples (i.e $\bar{X}_j = \frac{1}{n}\sum_{i=1}^{n} X_{ij}$)

2. **Unitary Matrix:** It is a square matrix whose transpose is also its inverse. i.e. $U^TU = UU^T = I$

PCA

PCA aims to find linearly uncorrelated orthogonal axes, which are also known as principal components in the m dimensional space to project the data points onto those vectors. The first PC captures the largest variance in the data. The second PC captures the remaining variation in an orthogonal direction to the first PC. The third PC captures what is left of variation in another orthogonal direction to the first two PCs and so one. This way a data matrix with d dimension has at most d PCs. If there exists a lower dimensional ($k < d$) representation of the data, it is likely that the first few PCs capture most of the variation and the projection of data points on them is sufficient for dimensional reduction with minimal information loss. This link provides a nice illustration on the intuition behind PCA in 2D.

The PCs can be determined via eigen decomposition of the covariance matrix C (Remember that the geometric meaning of eigen decompostion is to find a new coordinate system for C through rotations and scaling.)

$$C = Q\Lambda Q^{-1}$$

In this equation, the covariance matrix $C(d \times d)$ is decomposed to a matrix of eigen vectors $Q(d \times d)$ and a diagonal matrix of eigen values $\Lambda(d \times d)$. The eigenvectors are in fact PCs and their corresponding eigenvalue is the amount of variance captured. Moreover, $\alpha_i = \frac{\sigma_i}{\sum_{i=1}^{d}\sigma_i}$, where σ_i is the i^{th} eigenvalue, shows what percentage of the total variation in data is captured by the i^{th} PC. Once the eigenvectors with highest eigenvalues are found, we can use matrix multiplication to project the data $X(n \times d)$ on to the top k PCs, and hence get a lower dimensional representation of it $X_k(n \times k)$.

$$X_d = XQ_k$$

Where Q_k is the first k columns of matrix Q.

[1]Reference: https://towardsdatascience.com/pca-and-svd-explained-with-numpy-5d13b0d2a4d8

Singular Value Decomposition

The SVD decomposes any $m \times n$, rank r matrix A into a sum of r rank-one matrices:

$$A = \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + ... + \sigma_r \vec{u}_r \vec{v}_r^T$$

such that:

1. The set of vectors $\{\vec{u}_1, \vec{u}_2, ..., \vec{u}_r\}$ is orthonormal.
2. The set of vectors $\{\vec{v}_1, \vec{v}_2, ..., \vec{v}_r\}$ is orthonormal.
3. $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_r \geq 0$

The last restriction is imposed to ensure the uniqueness of the set $\{\sigma_1, \sigma_2, ..., \sigma_r\}$ as we will discuss this further later. We can express this in matrix form as follows:

$$A = U\Sigma V^T$$

Let's develop this expression from the initial sum $A = \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + ... + \sigma_r \vec{u}_r \vec{v}_r^T$. Let U_1 be the matrix formed by joining the column vectors $\{\vec{u}_1, \vec{u}_2, ..., \vec{u}_r\}$, and let V_1 be the matrix formed by joining the column vectors $\{\vec{v}_1, \vec{v}_2, ..., \vec{v}_r\}$. We can now define a diagonal matrix S, where the values $\{\sigma_1, \sigma_2, ..., \sigma_r\}$ are on the diagonal. S, therefore, must be an r x r matrix (since it is diagonal and we have r values σ). Now, we can rephrase the sum

$$A = \sigma_1 \vec{u}_1 \vec{v}_1^T + \sigma_2 \vec{u}_2 \vec{v}_2^T + ... + \sigma_r \vec{u}_r \vec{v}_r^T$$

as the matrix multiplication

$$A = U_1 S V_1^T = \begin{bmatrix} \vec{u}_1 & \vec{u}_2 & ... & \vec{u}_r \end{bmatrix} \begin{bmatrix} \sigma_1 & & 0 \\ & \ddots & \\ 0 & & \sigma_r \end{bmatrix} \begin{bmatrix} \vec{v}_1 & \vec{v}_2 & ... & \vec{v}_r \end{bmatrix}^T, \ U_1 \text{ is } m \times r,\ V_1^T \text{ is } r \times n$$

Since $\{\vec{v}_1, \vec{v}_2, ..., \vec{v}_r\}$ and $\{\vec{u}_1, \vec{u}_2, ..., \vec{u}_r\}$ are orthonormal, $U_1^T U_1 = V_1^T V_1 = I_{r \times r}$.

This, however, is still not the full matrix form of the SVD. An advantage of the SVD is that it provides compression when matrix A is not full rank (this is one of its primary use cases — this includes any time the A is not a square matrix). Since $r \leq m$, U_1 will not necessarily span the full m-dimensional space (nor will V_1 necessarily span the full n-dimensional space). The full matrix form of SVD is as follows:

$$A = U\Sigma V^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \left[\begin{array}{c|c} S & 0 \\ \hline 0 & 0 \end{array} \right] \begin{bmatrix} V_1 & V_2 \end{bmatrix}^T$$

where U_1 and V_1 are the matrices we defined earlier. While $A(n \times d)$ can be any shape, $AA^T(n \times n)$ and $A^T A(d \times d)$ are both square, symmetric, and positive semidefinite. Therefore, we can compress the data by looking at either its variation in each dimension ($C = A^T A$) or its variation across each sample point ($C = AA^T$). If we choose to compress across dimensions matrix V, and if we choose to compress across samples matrix U will emerge.

Before we go into the connection between PCA and SVD let's review the computation of SVD on a toy example. Please consult this great article which provides a nice walkthrough.

From SVD to PCA

PCA and SVD are closely related appraoches and can be both applied to decompose any rectangular matrices. We can look into their relationship by performing SVD on the covariance matrix C:

$$C = \frac{X^T X}{n-1} = \frac{V\Sigma U^T U\Sigma V^T}{n-1} = V \frac{\Sigma^2}{n-1} V^T$$

From this derivation, we notice that the result is similar to the eigenvalue decomposition of C. We can easily see the relationship between singular values Σ and eigenvalues (Λ).

$$\Lambda = \frac{\Sigma^2}{n-1}$$

Wrap-up

So far we have reviewed how we can use PCA/SVD to find a projection of data to a lower dimensional space, so that is easier to store in the limited space of our micro-controller (MSP400). We can also project our data on to two or three dimensions to visualize how separable our data is in the lower dimensions and whether using a simple classification algorithms like 1-means nearest neighbour is feasible. To summarize the PCA-based classification procedure:

1. Construct the initial data matrix A such that the rows of A represent different recordings, and the columns of A represent different timesteps.
2. Subtract the column mean from matrix A (i.e. mean across along samples)
3. Calculate the SVD or the eigenvectors of the covariance matrix.
4. Project your data to the new basis to get the lower dimensional samples
5. Use your projected data as the training data.

Notes written by Kourosh Hakhamaneshi (2020)