Principal Component Analysis:

A very basic ML algorithm that can be used for dimensionality reduction. This algorithm performs a lossy compression on given data that can be used to reduce the amount of storage the data takes up. It can also be used to learn representation of data that is non-linear.

Notations and definitions:

n: Scalar denoting the dimension of original data.

m: Scalar denoting number of examples.

I: Scalar denoting the dimension after PCA is performed on data

X: The feature matrix of dimension m × n. Every row of this matrix is a single vector $\mathbf{x}_i$ denoting the i^{th} example that is n-dimension.

C: The code matrix of dimension m × l. Every row of this matrix is a single vector $\mathbf{c}_i$ denoting the i^{th} example that is of l-dimension.

f(x): The encoder function. This is the first function we need to derive to perform the dimensionality reduction of **X** to get **C.** Basically, $\mathbf{c}_i = f(\mathbf{x}_i)$.

g(c): The decoder function. This is the second function we need to derive to perform the dimensional expansion of **C** to obtain **X** back. Here, we will define a matrix **D** to estimate the functions, listed next. $\mathbf{x}_i = g(\mathbf{c}_i) = g(f(\mathbf{x}_i))$.

D: A matrix of shape n × l that provides the mapping between **C** and **X**, thus defining the decoding function as g(c) = Dc. We restrict the columns of **D** to stay orthogonal to each other. This will simplify our calculation because our assumption makes $\mathbf{D}^T = \mathbf{D}^{-1}$.

Derivation of the encoder function, f(x):

To find the encoder function, we need to find the optimal code point for every $\mathbf{x}_i$ because $\mathbf{c}_i = f(\mathbf{x}_i)$. We can minimize the Euclidean distance between the original data and the data computed after we decode the code point. This distance can be given using the L2 norm. Further, we will get rid of the i subscript:

$||\mathbf{x} - g(\mathbf{c})||_2$

Since this needs to be minimized, we will just be proceeding with the square of the L2 norm because minimizing the square will give us the same optimal point for **c**:

$(||\mathbf{x} - g(\mathbf{c})||_2)^2$

$= (\mathbf{x} - g(\mathbf{c}))^T(\mathbf{x} - g(\mathbf{c}))$ $\quad (||\mathbf{x}||_2)^2 = \mathbf{x}^T\mathbf{x}$

$= \mathbf{x}^T\mathbf{x} - \mathbf{x}^T g(\mathbf{c}) - g(\mathbf{c})^T\mathbf{x} + g(\mathbf{c})^T g(\mathbf{c})$ $\quad$ (distributive property)

$= \mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T g(\mathbf{c}) + g(\mathbf{c})^T g(\mathbf{c})$ $\quad$ ($g(\mathbf{c})^T\mathbf{x} = (\mathbf{x}^T g(\mathbf{c}))^T$; transpose of a scalar)

$= \mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{D}\mathbf{c} + \mathbf{c}^T\mathbf{D}^T\mathbf{D}\mathbf{c}$ $\quad$ (substituting $g(\mathbf{c}) = \mathbf{D}\mathbf{c}$)

$= \mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{D}\mathbf{c} + \mathbf{c}^T\mathbf{c}$ $\quad$ ($\mathbf{D}^T\mathbf{D} = I$; read definition of **D** above)

The next step is to differentiate this function w.r.t. **c** and equating it to 0 to obtain the value of **c** at minima:

$\nabla_c (\mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{D}\mathbf{c} + \mathbf{c}^T\mathbf{c}) = 0$

$\Rightarrow \nabla_c \mathbf{x}^T\mathbf{x} - \nabla_c (2\mathbf{x}^T\mathbf{D}\mathbf{c}) + \nabla_c \mathbf{c}^T\mathbf{c} = 0$

$\Rightarrow 0 - 2\mathbf{D}^T\mathbf{x} + 2\mathbf{c} = 0$

$\Rightarrow \mathbf{c} = \mathbf{D}^T\mathbf{x}$

For the matrices **X** and **C**,

C = XD (rearranging for the feature matrix X)

Or,

f(x) = ***XD***

Now we can easily encode the data and perform the dimensionality reduction. However, for both the encoder and the decoder functions, we need the numerical value of **D.**

Deriving the value of D:

Again, we will be minimizing the distance between the input and their reconstructions. However, since the value of **D** is not trivial anymore, we shall minimize the norm across all values of **X.** The square of the Frobenius norm can be used for this (basically an L2 norm including all elements of the matrix):

$||\mathbf{X} - g(\mathbf{C})||_F \quad$ s.t. $\mathbf{D}^T\mathbf{D} = \mathbf{I}$

$= (||\mathbf{X} - g(f(\mathbf{X}))||_F)^2$ ($\mathbf{C} = f(\mathbf{X})$)

$= (||\mathbf{X} - g(\mathbf{XD})||_F)^2$ ($f(\mathbf{X}) = \mathbf{XD}$)

$= (||\mathbf{X} - \mathbf{XDD}^T||_F)^2$ ($g(\mathbf{D}^T\mathbf{X}) = \mathbf{XDD}^T$)

$= \text{Tr}((\mathbf{X} - \mathbf{XDD}^T)^T(\mathbf{X} - \mathbf{XDD}^T))$ ($||\mathbf{X}||_F = \sqrt{(\text{Tr}(\mathbf{X}^T\mathbf{X}))}$; Tr represents the Trace operator)

$= \text{Tr}(\mathbf{X}^T\mathbf{X} - \mathbf{X}^T\mathbf{XDD}^T - \mathbf{DD}^T\mathbf{X}^T\mathbf{X} + \mathbf{DD}^T\mathbf{X}^T\mathbf{XDD}^T)$ (Distributive property; maintaining shape along terms)

$= \text{Tr}(\mathbf{X}^T\mathbf{X}) - \text{Tr}(\mathbf{X}^T\mathbf{XDD}^T) - \text{Tr}(\mathbf{DD}^T\mathbf{X}^T\mathbf{X}) + \text{Tr}(\mathbf{DD}^T\mathbf{X}^T\mathbf{XDD}^T))$ (Trace is distributive)

$= \text{Tr}(\mathbf{X}^T\mathbf{X}) - 2\text{Tr}(\mathbf{X}^T\mathbf{XDD}^T) + \text{Tr}(\mathbf{X}^T\mathbf{XDD}^T\mathbf{DD}^T))$ (Terms inside the Trace operator can be interchanged)

$= \text{Tr}(\mathbf{X}^T\mathbf{X}) - 2\text{Tr}(\mathbf{X}^T\mathbf{XDD}^T) + \text{Tr}(\mathbf{X}^T\mathbf{X}))$ ($\mathbf{D}^T\mathbf{D} = \mathbf{I}$)

$= 2\text{Tr}(\mathbf{X}^T\mathbf{X}) - 2\text{Tr}(\mathbf{D}^T\mathbf{X}^T\mathbf{XD})$

To minimize this function and find an optimal point for **D**, we can ignore terms not involving **D** and constants:

$\min -\text{Tr}(\mathbf{D}^T\mathbf{X}^T\mathbf{XD})$

$= \max \text{Tr}(\mathbf{D}^T\mathbf{X}^T\mathbf{XD})$ ----------(1)

To maximize this, we can exploit the eigen decomposition for matrices:

$\mathbf{A} = \mathbf{QLQ}^{-1}$

Where **Q** is an orthogonal matrix of with columns as vectors representing the different eigenvalues of **A** and **L** is a diagonal matrix where $\mathbf{L}_{i,i}$ is the eigenvalue corresponding to the i^{th} column eigenvector of **Q**.

Rearranging,

$\mathbf{L} = \mathbf{Q}^T\mathbf{AQ}$ (Q is orthogonal)

This implies that in the equation (1) to be maximized above, when a column $\mathbf{D}_i$ is the eigenvector of $\mathbf{X}^T\mathbf{X}$, the value that the equation takes is the eigenvalue of $\mathbf{X}^T\mathbf{X}$ and to maximize it, we need the l maximum eigenvalues of $\mathbf{X}^T\mathbf{X}$.

Therefore, the required value of **D** consists of l eigenvectors corresponding to the l maximum eigenvalues of $\mathbf{X}^T\mathbf{X}$ as its columns. The eigenvectors of $\mathbf{X}^T\mathbf{X}$ can be calculated easily by finding the right-singular eigenvectors of X by performing Singular Value Decomposition on **X**.

Once D is calculated, the algorithm is as listed below.

The algorithm

SVD(X): Calculates the singular value decomposition of X (returns left-singular eigenvectors, eigenvalues and right-singular eigenvectors)

-U, S, V := SVD(X)

-Sort S and correspondingly, V (descending)

-D = matrix of first l columns of V

To encode: $C = X \times D$; To decode: $X = C \times D^T$ (rearranging for the feature matrix X)

Minimum l:

As l decreases, the compression will be lossier, i.e., when decoded, the error will be higher. An obvious way of finding minimum l is by finding the value of l where the disparity between decoded value and original data exceeds a small error (say 1%). An easier way is finding the number l for which the sum of l maximum eigenvalues accounts for 99% of the sum over all eigenvalues (here, s is the vector of eigenvalues obtained by SVD):

$$\frac{\sum_{i=1}^{l} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

*inspired by Deep Learning by Ian Goodfellow

*scale the data while coding

*example code in Python provided