Principle Component Analysis:

PCA is a powerful linear transformation technique to project the dataset on a new set of orthogonal vector subspace (principle components). Often real world data is described in very large dimensional space and the desire is to reduce dimensionality while still retaining most of the information. PCA achieves this by finding directions of maximum variance from dataset, in the process of finding new set of orthogonal vectors. Increasing the variance is equivalent to minimizing the squared reconstruction error between the principle components and the dataset. It means that the principle components, will be closest to the dataset.

PCA essentially is eigendecompostision of the covariance matrix. For any m x n centered matrix A,

covariance matrix C, is given by $A^T(A)/(n-1)$

and its eigendecomposition is given by

$$C = P\ S\ (P^T)$$

where P, eigenvector of covariance matrix, and S is a diagonal matrix with eigenvalues at diagonals.

C, captures the correlations between different features used to describe data. Features that are highly colinear will have high covariance and can be assumed to be redundant, or linearly dependent on each other. Covariance matrix is not diagonal for real world data as there will be redundant features in dataset. PCA diagonalizes the covariance matrix by creating principle components that are linearly independent to each other. For huge datasets identifying redundant features to reduce dimensionality become important to increase computational efficiency.

There are mathematical proofs to show that eigenvectors are the directions, which will have maximum variance. I will be using wholesale customer dataset from https://archive.ics.uci.edu/ml/datasets/wholesale+customers for demonstrations.

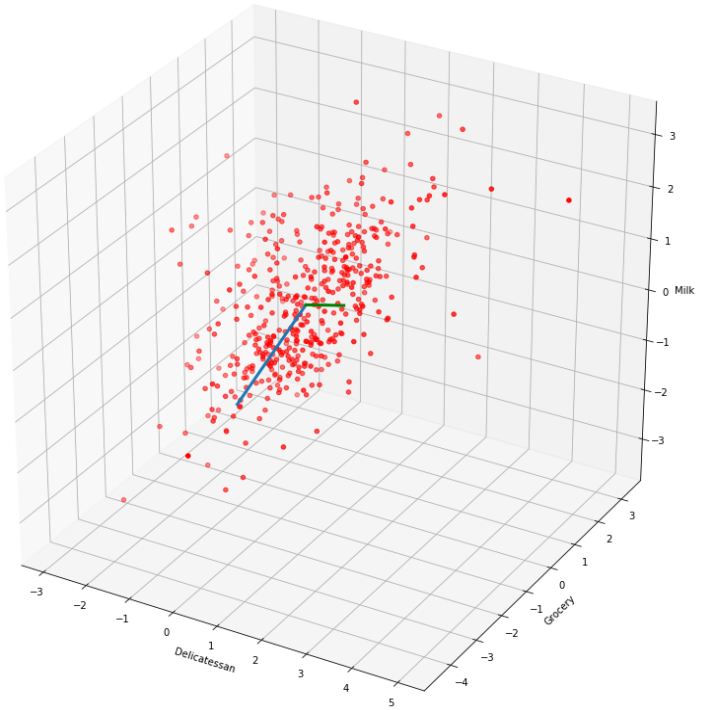

Fig.1: 3D representation of wholesale customer dataset with 'Milk', 'Grocery', and 'Delicatessan' as standard bases. Blue line represents first principle component. Green line represents the second principle component. Variance in direction of PC is captured by eigenvalues.

But, how to find these principle components? (Eigenvectors of the covariance matrix). Here I will use iterative methods to calculate few eigenpairs instead of full eigenspace.

a) *Lanczos algorithm, power method*

This is an iterative process to find the eigenvectors of the symmetric square matrix, covariance matrix.

- Choose a random unit vector of compatible shape to the dataset.
- Multiply covarince matrix with random vector. Covariance matrix acts as a linear transformation; hence it rotates and scales the vector. Note, the matrix will rotate any vector in a similar fashion, as it is a linear transformation.
- Upon iterative transformation of the vector with covariance matrix, vector will reach a direction that will no longer change; this is the first principle component.
- Normalize the vector, so it has unit length. (Without this constraint, you will get many vectors instead of 1)
- Subtract the projection of the first principle components from the matrix and repeat the process to find the next principle component analysis.

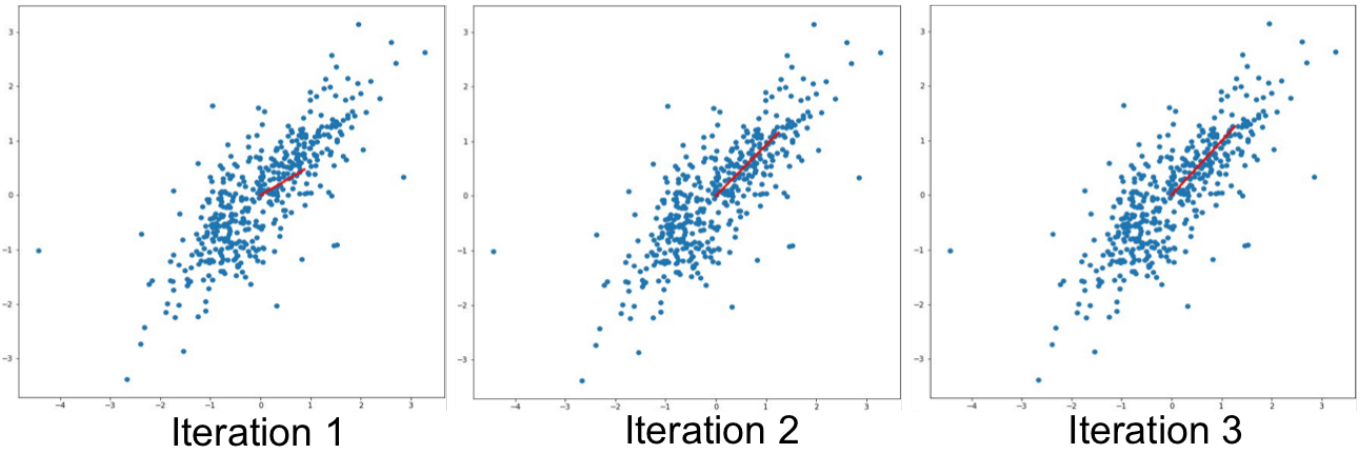

Fig.2: Finding direction of eigenvectors starting from random vector. 'Grocery' and 'Milk', from customer dataset represent x, y standard basis. At each iteration covariance matrix is multiplied with the vector, which rotates and scales initial vector. New vector is normalized before being multiplied for next iteration. After few iterations, vector stops changing direction as it represent the eigenvector of the covariance matrix.

b) *Singular Value Decomposition, finding directions with maximum variance by Gradient Descent*

Let A be centered m x n matrix (mean of each feature column is 0), then A can be written as a product of 3 matrices such that

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

- U is left singular matrix and has orthonormal columns, hence $(U^T)U = I$. Here the columns are eigenvectors of AA^T
- V is right singular matrix and has orthonormal columns, hence $(V^T)V = I$. Here the columns are the eigenvector for A^TA, covariance matrix.
- Σ is a diagonal matrix with singular values at the diagonals.

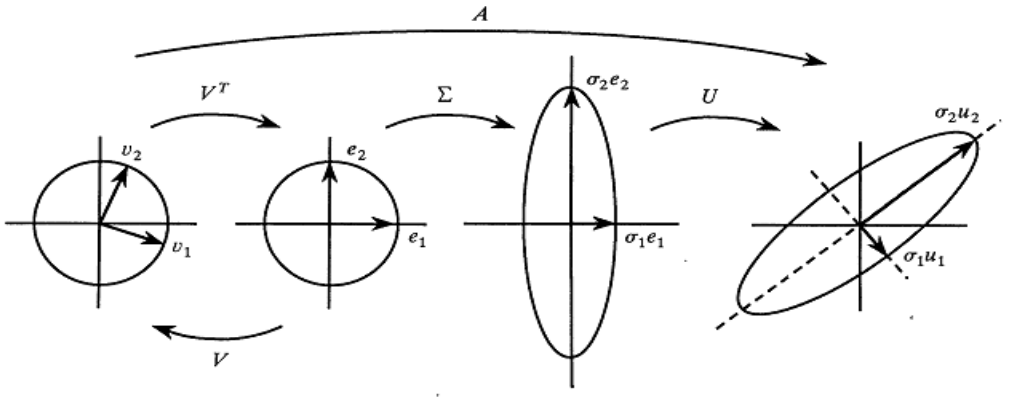

Fig.3: Diagram showing the linear transformation of a matrix A when SVD is applied.

SVD allows rotating the axes of the feature space of dataset so that the new axes points in the direction of certain linear combination of original axes. The new axes have high variance in transformed data and directions corresponding to low variances can be neglected. Hence achieving reduction in dimension.

SVD is a generalized decomposition of any matrix and performs a low rank matrix approximation. It does not require calculating $X^T(X)$ which can cause loss of precision. SVD provides subspace that directly spans the datasample as compared to PCA, which spans the deviation from mean of datasample.

Here I am not decomposing the matrix, rather calculating the eigenvector based on direction of maximum variance. These directions will be columns of matrix V in SVD. I will be using gradient descent technique to converge to eigenvector faster.

- Center the data around each feature mean.
- Choose a random compatible unit vector u, and initialize the update to 1 (I chose this at random to start the update).
- Set epoch to in range 50. At each epoch, calculate projections of datapoints wrt vector and find variance. This is the function we want to maximize.

  Variance = ((X.dot(u)).^{T}).dot(X.dot(u))/n.
- Slope of function = 2*X.dot(u) -2*u and update, u = u+ learning_rate*X.dot(slope)
- At each iteration update is made to vector, and when the difference between 2 consecutive updates is e-05, vector is found.
- Subtract the projection of the first principle components from the matrix and repeat the process to find the next principle component analysis.

I found that both these methods had similar iteration counts to find the eigenvectors and gave same vector compared to sklearn implementation of PCA. But they are much slower than sklearn for bigger dataset like Madelon.

Finally eigenvalues are calculated as,

eigenvalue = eigenvector.dot(C).dot(eigenvector)

Transform data after PCA:

After the eigenvectors are found, select p vector that have the high variance and create truncated eigenvector array V_P, as column vectors. Center the data around mean as X.

Transformed data, $Y = X.dot(V_P^T)$

Data will be transformed to eigenspace

Reconstructing original data:

$$X' = Y.dot(V_P) = X.dot(V_P^T).dot(V_P)$$