PCA_on_MNIST_dataset

August 9, 2017

1 PCA: Principle Component Analysis

1.1 Understanding PCA with an example

1.1.1 Caveat: The example is a long one but I think you will understand it better if you read it.

Let's say you work as a Data Scientist for an Ad Marketing Agency where your task is to analyze the data about customer purchases and find patterns as to **what made a user buy something and how can I make him/her buy more?**

Now, this question we posed above can have so many follow up questions, like: 1. Where did the user prefer buying from ? (Amazon, Flipkart, etc) 2. What kind of add did the user click most time on? (display ads, search ads, product listing ads, email, etc) 3. Which ad networks/channel showed most conversions? (Facebook, Google, Twitter, LinkedIn) 4. Some complex questions can include, did a user first see an ad on Facebook and then googled up to see the best prices and buy it? These are just some question out of the thousands that can be answered to improve user experience, customer acquisition, etc

Let's look what kind of data you may get to analyze. Let's accept the fact that, storing the information in **RDBMS** is not a good option because there will be lot of user tracking data that would come in various formats and very difficult to store in RDBMS. So, this kind of tracking data is usually dumped into a **Data Warehouse** and then various pipelines are written to extract useful ones into RDBMS to make them faster. I will not go into the details of this, if you are interested, you can read about the same with links in the references section.

So without diverting from a main point, if we have huge number of data, we can just shard them and use them sequentially to keep it very simple, however, if we have more number of fields/columns/features to extract information from, then it would become really time consuming and difficult as well. **Imagine you have data about a user, which shows entire tracking of that particular user for a entire day/month/year, from various places, on various sites, on various products, at various times, from various devices, using various social accounts, etc are stored in a warehouse.**

Simply put, not all of what is stored in the warehouse is useful information and hence we can exclude some of them from our analysis. But, the billion dollar question is, 'HOW'??? How do you actually find out which one is useful and which one is not? Here, we have **PCA** smirking at us and lending a helping hand. It says, I can choose the important features for you!

So, you through maybe a **100** features at it and it will tell you to choose **20** of them which actually are useful in finding out information about the user.

In the later sections we will see how it actually does this, with a real world example from **MNIST data set** and various steps included in it.

After that, we will look at one more technique like PCA which does this better, called t-SNE

1.1.2 The Process

- Normalize/Standardize the data, after this step, we get a covariance Vector
- Now find the eigenvectors and eigenvalues of this vector
- The eigen values are scalars that have a certain magnitude
- If eigen values for certain vectors are relatively very low, it is better to discard them off **(these are maybe the features that we were talking about which are not that useful in getting answers about the customer)**. Another way to look at this is, if some of the information is not making much sense, then its better to discard it as to reduce the dimensions and send less features to a machine learning algorithm to predict from.

1.1.3 Is normalization necessary??

For the customer example, if the price of the commodity ranges from 1 to 1,00,00, and the quantity of items purchased will be let's say 1 to 10. Does it not make sense to scale them so that they fall in between some rigorous value and doesn't seem like price is almost always overpowering the quantity.

Note that, we do not want to miss out on any useful information.

1.1.4 Anyways, what is a eigen vector and what does it have to do with PCA??

It's the direction in which the data varies a lot. Let's say we just defined the first eigen vector having the max eigen value(magnitude). The second eigen vector is the one that has maximum variance again but in the orthogonal direction of the first one; similarly, the third eigen vector is the ones that has max variance but is orthogonal to both first and second and so on if there are more dimensions..

So, this eigen vector is called a Priciple Component.

1.1.5 Principle Component ??

Remember, we said we want to be able to discard some features, this priciple component will be used to discard these features but without losing max information. This principle compenent covers the max number of dimensions as it can by reorienting the axis.

1.1.6 How can we reorient the axis, seems so unreal?

Since the principal components or the eigen vectors of the covariance matrix are orthogonal to each other, it is possible that we change the axes of the data i.e reorient the axes.

1.1.7 Facts about PCA

- Linear Transformation Technique
- PCA depends on closeness of the points, and the closeness in measure as the average squared euclidean distances
- PCA holds the directions that have the maximum spread, or variance
- PCA ignores the class labels
- Higher to lower dimensional without loosing much info

- If there is corelation between data, then it makes sense to reduce the dimensionality

2 Code Implementation

Let's play with some code to understand how difficult it is to implement PCA

```
In [1]: import numpy as np
        import scipy.stats as st
        import pandas as pd
        from mpl_toolkits.mplot3d import Axes3D
        import matplotlib.pyplot as plt
        import operator
        import seaborn as sns
        from sklearn.preprocessing import StandardScaler

        from sklearn.manifold import TSNE
        from sklearn.decomposition import PCA

        from IPython.display import display
        from IPython.display import Latex
        from IPython.display import Math

        %matplotlib inline
```

2.0.1 Exploring Data

```
In [2]: # read the data
        df = pd.read_csv('./datasets/train.csv')
```

```
In [3]: df.head()
```

```
Out[3]:    label  pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  \
        0      1       0       0       0       0       0       0       0       0
        1      0       0       0       0       0       0       0       0       0
        2      1       0       0       0       0       0       0       0       0
        3      4       0       0       0       0       0       0       0       0
        4      0       0       0       0       0       0       0       0       0

           pixel8    ...     pixel774  pixel775  pixel776  pixel777  pixel778  \
        0       0    ...            0         0         0         0         0
        1       0    ...            0         0         0         0         0
        2       0    ...            0         0         0         0         0
        3       0    ...            0         0         0         0         0
        4       0    ...            0         0         0         0         0

           pixel779  pixel780  pixel781  pixel782  pixel783
        0         0         0         0         0         0
        1         0         0         0         0         0
```

```
2          0          0          0          0          0
3          0          0          0          0          0
4          0          0          0          0          0

[5 rows x 785 columns]
```

```
In [4]: # remove the label from the training set and add it to another variable
        label = df['label']
        df.drop('label', axis=1, inplace=True)
```

```
In [5]: df.head()
```

```
Out[5]:    pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
        0       0       0       0       0       0       0       0       0       0
        1       0       0       0       0       0       0       0       0       0
        2       0       0       0       0       0       0       0       0       0
        3       0       0       0       0       0       0       0       0       0
        4       0       0       0       0       0       0       0       0       0

           pixel9    ...     pixel774  pixel775  pixel776  pixel777  pixel778  \
        0       0    ...            0         0         0         0         0
        1       0    ...            0         0         0         0         0
        2       0    ...            0         0         0         0         0
        3       0    ...            0         0         0         0         0
        4       0    ...            0         0         0         0         0

           pixel779  pixel780  pixel781  pixel782  pixel783
        0         0         0         0         0         0
        1         0         0         0         0         0
        2         0         0         0         0         0
        3         0         0         0         0         0
        4         0         0         0         0         0

        [5 rows x 784 columns]
```

2.0.2 PCA implementation using sklearn

```
In [6]: df.shape
```

```
Out[6]: (42000, 784)
```

The other technique (t-SNE) takes considerable amount of time on the entire dataset and my laptop almost died of one such attack, hence I am reducing(slicing) the data, so that PCA anad t-SNE comparison is easier and on the same data.

```
In [7]: # Uncomment the below lines if slicing of the data is desired
        #df = df[:2000]
        #label = label[:2000]
```

Normalizing/Standardizing data

```
In [8]: df_std = StandardScaler().fit_transform(df)
```

Create a PCA instance with 5 components, i.e. dimensions, taken randomly after some experiments with other values

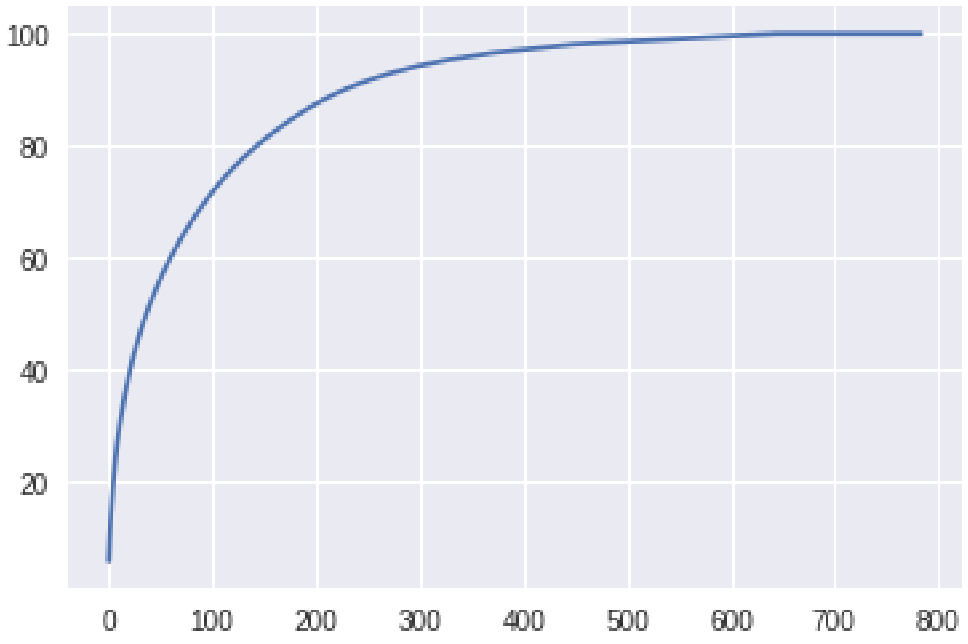
```
In [9]: pca = PCA()
```

```
In [10]: pca = pca.fit(df_std)
```

```
In [11]: variance = np.cumsum(np.round(pca.explained_variance_ratio_, decimals=4)*100)
```

```
In [12]: plt.plot(variance)
```

```
Out[12]: [<matplotlib.lines.Line2D at 0x7f10a4b90f10>]
```



Looking at the data, we see not much steepness after value 75 on the x-axis, thus I will go with 75 components, to fit and transform the data

```
In [13]: pca = PCA(n_components=200)
```

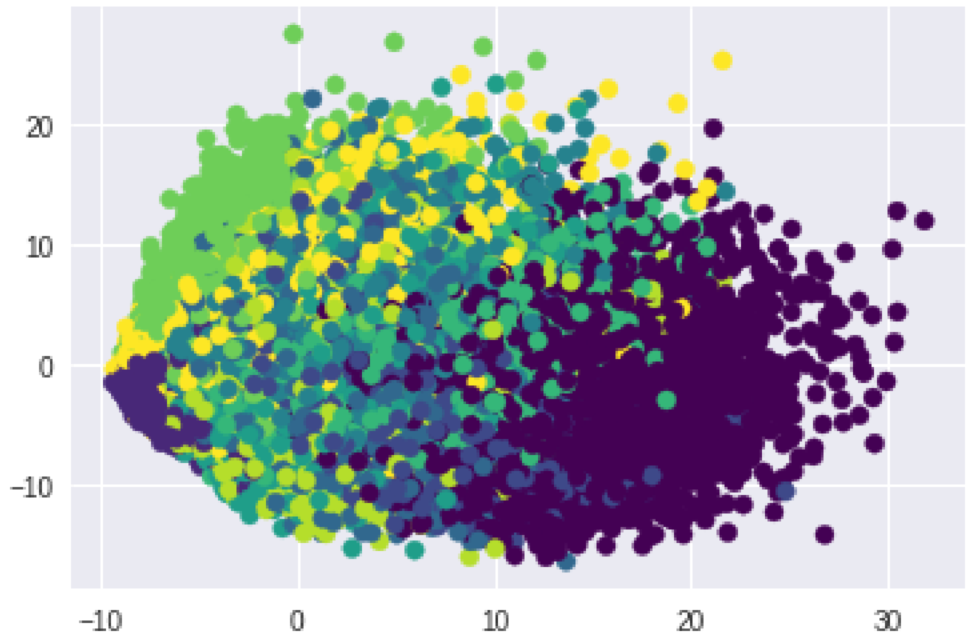
```
In [14]: df_pca = pca.fit_transform(df_std)
```

Transform the data depending on the principle components that were obtained from above step. This is very very easy, all the part of : **covariance matrix, eigen vector calculation, eigen value calculation, choosing the top 5 ones, is done by the above line**

The above line will transform the data according to the new principle components, but we really want to see how it has done it, don't we??

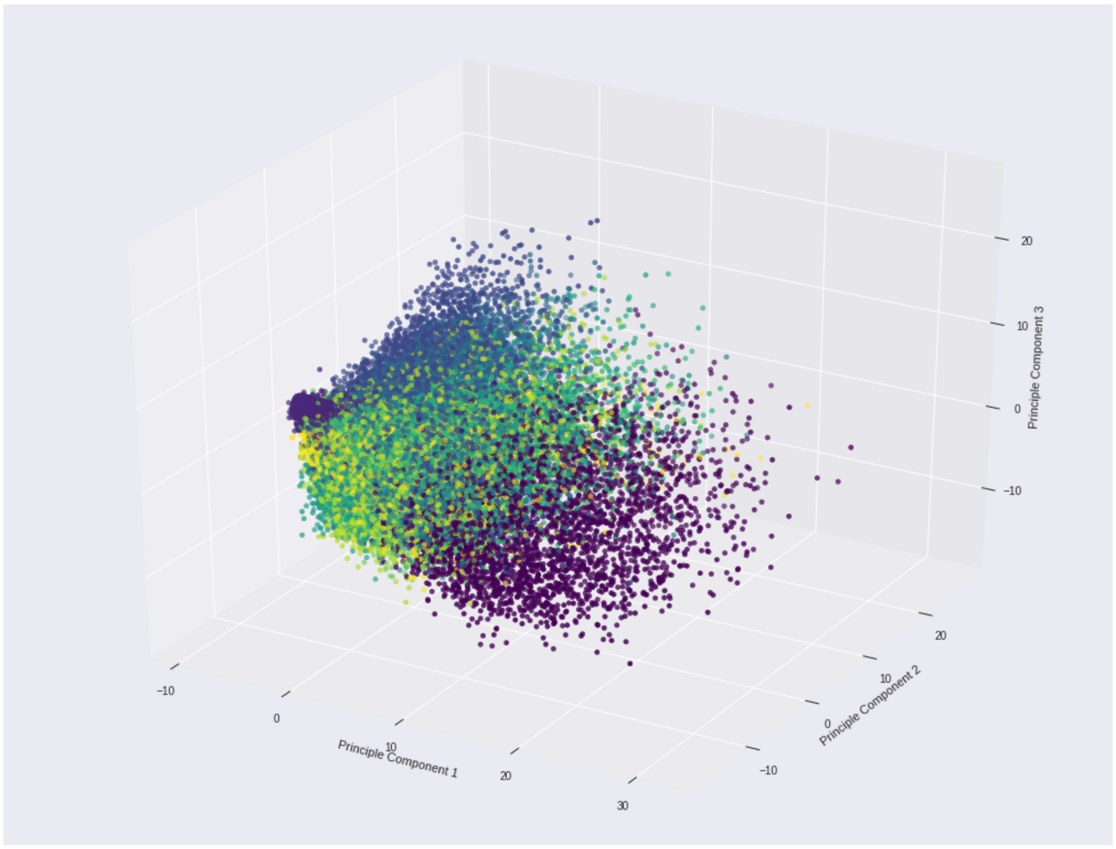
Below I have used a interactive plotting library 'plotly', the code is literally copy pasted from one of the kaggler's code and is very simple to comprehend.

```
In [15]: plt.scatter(data=df_pca, x = df_pca[:,0],y = df_pca[:,1], c=label, cmap='viridis')

Out[15]: <matplotlib.collections.PathCollection at 0x7f1066db7ed0>
```



Let’s try to see the same scatter plot in 3D

```
In [34]: fig = plt.figure(figsize=(18, 14))
         ax = fig.add_subplot(111, projection='3d')
         ax.set_xlabel('Principle Component 1')
         ax.set_ylabel('Principle Component 2')
         ax.set_zlabel('Principle Component 3')
         ax.scatter(df_pca[:,0], df_pca[:,1], df_pca[:,2], c=label, cmap='viridis')
         plt.show()
```



Though there is some fair bit of separation between that we can observe by looking at the colors, it is still not that evident. At this point it may feel that, all the damn efforts have gone down the drain, but we actually learned a fair lot of things here.

In the next post, I will use t-SNE to plot the same figure and we will see how it helps us visualize what we tried to do above in a superb way!

3 Bonus (Beware => Math Inside)

3.1 Unboxing the black box called PCA

We are not done yet, we do need to learn the math behind the scenes.

We just now used a bunch of text book techniques, which will help us reduce the dimensions; but I really want to understand what I did, read on if you are on the same page. I will guide you step by step on what this black-box is doing internally.

3.1.1 Standardizing

Though standardizing in this example won't do us any favour, check the pixel values below, they are anyways binary but I will keep the steps same so as to not confuse you.

```
In [35]: df.head()
```

```
Out[35]:    pixel0  pixel1  pixel2  pixel3  pixel4  pixel5  pixel6  pixel7  pixel8  \
         0       0       0       0       0       0       0       0       0       0
         1       0       0       0       0       0       0       0       0       0
         2       0       0       0       0       0       0       0       0       0
         3       0       0       0       0       0       0       0       0       0
         4       0       0       0       0       0       0       0       0       0

            pixel9    ...     pixel774  pixel775  pixel776  pixel777  pixel778  \
         0       0    ...            0         0         0         0         0
         1       0    ...            0         0         0         0         0
         2       0    ...            0         0         0         0         0
         3       0    ...            0         0         0         0         0
         4       0    ...            0         0         0         0         0

            pixel779  pixel780  pixel781  pixel782  pixel783
         0         0         0         0         0         0
         1         0         0         0         0         0
         2         0         0         0         0         0
         3         0         0         0         0         0
         4         0         0         0         0         0

         [5 rows x 784 columns]
```

```
In [22]: df_std = StandardScaler().fit_transform(df)
```

3.1.2 Covariance matrix

Why is the covariance of a matrix important?

Suppose we have a 2D plot, we saw how variance helps us find the spread of data. But, let's say our spread of data is diagonal and not paraller to any of the 2D axis? In this case, covariance of (x,y) explains the diagonal spread, like the regression line for ex.

Hence, we first find the covariance matrix and then go about finding the eigen values which will give the max spread i.e. spread; hence we can also infer that, sum(eigen values) = sum(covariances of all dimensions)

Thus, if we find the covariance matrix, the diagonal elements are variances and off diagonal are covariances, and hence its symmetrix. Let's see an example below:

$$\sum = [\sigma(x,x)\sigma(x,y)$$
$$\sigma(y,x)\sigma(y,y)]$$

With the above lines we can infer that, a 2D data is explained by its 22 *covariance matrix, and its mean. Simillarly for a nD data, we will require n*n cov matrix.

Reference http://www.visiondummy.com/2014/04/geometric-interpretation-covariance-matrix/

$$CoVarianceMatrix = 1/n - 1 * \sum_{i=1}^{n}(X_i - \mu_x)^T(X_i - \mu_x)$$

```
In [23]: mean_vector = np.mean(df_std, axis=0)
         cov_matrix = (df_std - mean_vector).T.dot((df_std - mean_vector)) / (df_std.shape[0]-1)
```

Another way to get the covariance matrix without much of the above noise is using the numpy way. **np.cov(df_std.T)**

Both the above methods, show the same results, however, its just the verbosity of the code, makes things understand better

3.1.3 Eigen Decomposition

** Why is the eigen decomposition so needed and how do eigen values and eigen vectors help us here anyway?? ** From the above steps, we have the covariance matrix that gives the variance of the data and also the relation between the other features. If we are somehow able to represent the spread(variance) in terms of a vector, we will be able to find the scalar value with which it gets transformed and also the direction, which is our main goal, it would be very helpful in knowing which vectors i.e. dimensions help us explain the max spread in the data. Hence, we will be able to reduce the data to only include those dimensions/vectors that give the max spread.

Now, it can be proved that, if a matrix is diagonal, i.e. the covariances are 0, then the sum(variance) = sum(eigen values), but if the matrix is not diagonal, then we can say that, the eigen values represent the variance of the along the direction of eigen vectors and variance components of the cov matrix explain the, spread of the data across axes.

So, now, on performing the eigen decomposition on the above matrix, we get the eigen values as well as the eigen vectors (principle components) using some numpy functions, calculating of eigen vectors and values manually isn't required as we have a library to support this **eig_vals, eig_vecs = np.linalg.eig(CoVariance_matrix)**

```
In [24]: eig_vals, eig_vecs = np.linalg.eig(cov_matrix)
         print "<<==Eigen Values==>>\n{}".format(eig_vals)
         print "-----------------------------------------------------------------------------
         print "<<==Eigen Vectors==>>\n{}".format(eig_vecs)
```

```
<<==Eigen Values==>>
[  4.06964787e+01   2.91114657e+01   2.67833371e+01   2.08147194e+01
   1.81000206e+01   1.57876737e+01   1.38244007e+01   1.25432643e+01
   1.10638975e+01   1.00889267e+01   9.63617203e+00   8.65579470e+00
   8.04120472e+00   7.88086691e+00   7.43637560e+00   7.16743699e+00
   6.73538375e+00   6.61651973e+00   6.42354578e+00   6.26826675e+00
   5.93960379e+00   5.74928832e+00   5.48826880e+00   5.32649477e+00
   5.15217038e+00   4.94730998e+00   4.88853571e+00   4.70777145e+00
   4.46528559e+00   4.36351702e+00   4.32543150e+00   4.22712324e+00
   4.08726514e+00   4.06176768e+00   3.99903435e+00   3.86804997e+00
   3.81925839e+00   3.71256507e+00   3.57437538e+00   3.45887625e+00
   3.41436841e+00   3.36945857e+00   3.25693182e+00   3.24008824e+00
   3.18312949e+00   3.16286640e+00   3.14244041e+00   3.09287815e+00
   3.06368054e+00   3.02342271e+00   2.96849737e+00   2.91830693e+00
   2.84948896e+00   2.82806029e+00   2.79589667e+00   2.76696531e+00
   2.68974637e+00   2.63703541e+00   2.60842615e+00   2.58938021e+00
   2.50126884e+00   2.48576071e+00   2.20802976e+00   2.24282552e+00
   2.26576633e+00   2.44558621e+00   2.41712754e+00   2.33451020e+00
```

```
2.34856157e+00   2.38231165e+00   2.39156842e+00   2.21393592e+00
2.18219367e+00   2.14724525e+00   2.13473541e+00   2.10380281e+00
2.08757888e+00   2.07894080e+00   4.75373246e-03   1.88077030e-03
5.97321913e-04   2.03801029e+00   2.03133938e+00   2.00490133e+00
2.00287449e+00   1.99449324e+00   1.97689115e+00   1.96008723e+00
1.95636136e+00   1.93942293e+00   1.92446256e+00   1.87937806e+00
1.89848257e+00   1.84200398e+00   1.85551338e+00   1.82806936e+00
1.80457363e+00   1.79485190e+00   1.78704055e+00   1.77056345e+00
1.74032854e+00   1.73010661e+00   1.70906271e+00   1.68261945e+00
1.64494828e+00   1.67268356e+00   1.62487959e+00   1.63015678e+00
1.60339823e+00   1.59274839e+00   1.58361788e+00   1.52008847e+00
1.53504626e+00   1.55901925e+00   1.56267777e+00   1.49520235e+00
1.48603778e+00   1.45055614e+00   1.44459114e+00   1.43033417e+00
1.41795560e+00   1.39484275e+00   1.38579986e+00   1.37650895e+00
1.13878223e+00   1.14433133e+00   1.17802678e+00   1.35771292e+00
1.34771049e+00   1.34213381e+00   1.31356375e+00   1.29299011e+00
1.15545658e+00   1.19329954e+00   1.27261905e+00   1.24739641e+00
1.30493639e+00   1.16093225e+00   1.22793075e+00   1.23115097e+00
1.25879085e+00   1.34570698e+00   1.18916126e+00   1.26981714e+00
1.21762120e+00   1.13064811e+00   1.12236199e+00   8.25674163e-01
8.62347600e-01   1.10780517e+00   1.10288070e+00   1.09195242e+00
1.08405932e+00   1.08133300e+00   1.06294123e+00   1.05738260e+00
8.73931589e-01   8.79240514e-01   8.80187779e-01   1.04663138e+00
1.04507095e+00   1.04311840e+00   8.92863122e-01   8.97572899e-01
9.02871955e-01   1.02998178e+00   1.02558001e+00   9.07482172e-01
9.33037111e-01   9.29057570e-01   9.05966151e-01   9.43466664e-01
9.20584735e-01   9.65831160e-01   1.01540703e+00   9.51684336e-01
9.72246164e-01   1.00880528e+00   9.80237354e-01   9.85156455e-01
9.19459865e-01   9.53894115e-01   9.91308409e-01   1.00401273e+00
9.94638916e-01   1.00085979e+00   9.97489540e-01   1.00059511e+00
9.98732579e-01   6.57072661e-01   6.67469656e-01   6.74514972e-01
6.81612982e-01   6.86033976e-01   6.88015185e-01   6.98586791e-01
7.02034106e-01   7.13229040e-01   7.15415942e-01   7.84585502e-01
8.03255754e-01   8.11592940e-01   8.12861064e-01   7.76178902e-01
8.19252418e-01   7.22733581e-01   7.71126050e-01   7.63294320e-01
8.40676692e-01   7.52893560e-01   8.49807232e-01   7.42636328e-01
7.47750289e-01   7.39046734e-01   8.31539874e-01   7.58028622e-01
5.45699968e-01   5.55960471e-01   5.60115218e-01   5.65957087e-01
5.68610497e-01   5.71834852e-01   5.80874805e-01   5.84207606e-01
6.47560094e-01   6.44656444e-01   5.94951302e-01   5.90502358e-01
6.24610009e-01   6.26853394e-01   6.07370327e-01   6.13360650e-01
6.10604440e-01   6.35679718e-01   6.38688842e-01   6.03973922e-01
6.63476028e-01   4.53273625e-01   4.59936905e-01   4.67378543e-01
4.70889600e-01   4.74146317e-01   5.35246706e-01   4.83352924e-01
5.31868973e-01   5.06213771e-01   4.99094655e-01   4.87552419e-01
5.24854304e-01   5.13704450e-01   4.92323016e-01   4.90344290e-01
5.19563501e-01   5.21171138e-01   5.14673411e-01   4.92977886e-01
3.71892133e-01   3.79533086e-01   4.44865137e-01   4.42581515e-01
```

```
 3.84145290e-01   3.86223501e-01   4.39246566e-01   3.90854436e-01
 3.87473943e-01   3.98159900e-01   4.00795537e-01   4.21478409e-01
 4.25772687e-01   4.29492020e-01   4.12930344e-01   4.07094341e-01
 4.05414453e-01   4.16710554e-01   4.03143926e-01   4.34074136e-01
 4.28364801e-01   4.14210541e-01   3.08783622e-01   3.12036206e-01
 3.13972075e-01   3.18827959e-01   3.19916448e-01   3.69391173e-01
 3.66358393e-01   3.73172176e-01   3.35520774e-01   3.40342867e-01
 3.29214797e-01   3.47669173e-01   3.57916892e-01   3.62248115e-01
 3.51439399e-01   3.55276687e-01   3.21341831e-01   3.38702285e-01
 3.53444103e-01   3.28742206e-01   3.22654681e-01   3.44804820e-01
 2.49108924e-01   2.50312901e-01   3.06327099e-01   3.03285903e-01
 2.98252654e-01   2.95891353e-01   2.56065840e-01   2.59328865e-01
 2.52189835e-01   2.52674271e-01   2.93156105e-01   2.63532046e-01
 2.92655253e-01   2.89165572e-01   2.74542527e-01   2.69170325e-01
 2.79089520e-01   2.62356076e-01   2.81779345e-01   2.71518295e-01
 2.70766198e-01   2.88693138e-01   2.84077254e-01   2.85060320e-01
 2.80576951e-01   2.68093590e-01   2.46769035e-01   2.43578595e-01
 1.97737649e-01   1.98096510e-01   2.39788724e-01   1.99786715e-01
 2.00812560e-01   2.37041838e-01   2.39172871e-01   2.34325533e-01
 2.39559061e-01   2.05305509e-01   2.03856109e-01   2.32192697e-01
 2.31521107e-01   2.11003393e-01   2.25467669e-01   2.23686573e-01
 2.07362911e-01   2.27315459e-01   2.11752378e-01   2.02235560e-01
 2.29877264e-01   2.19310570e-01   2.28369757e-01   2.13657138e-01
 2.16085477e-01   2.07782126e-01   2.22209574e-01   2.16709922e-01
 2.18287058e-01   1.96511555e-01   1.92617457e-01   1.91272983e-01
 1.62976620e-01   1.63320836e-01   1.89166923e-01   1.64574788e-01
 1.87869082e-01   1.85706416e-01   1.83902655e-01   1.83141606e-01
 1.67019615e-01   1.69040063e-01   1.71154472e-01   1.86456483e-01
 1.79920687e-01   1.73803618e-01   1.67363572e-01   1.69525154e-01
 1.72545572e-01   1.77381158e-01   1.81311986e-01   1.75975867e-01
 1.74807379e-01   1.79161251e-01   1.75273366e-01   1.38183242e-01
 1.39131917e-01   1.40338961e-01   1.59755690e-01   1.60732414e-01
 1.60511066e-01   1.59060560e-01   5.38750452e-15   1.41872593e-01
-1.59146613e-16   1.55975646e-01   1.43655513e-01   1.43918353e-01
 1.45311681e-01  -5.65478731e-16  -2.96610127e-16   1.45952701e-01
 1.49425967e-01   1.48261878e-01   1.52775932e-01   1.54403086e-01
 1.51098405e-01   1.47044774e-01   1.42058242e-01   1.57171797e-01
 1.51708097e-01   1.54029034e-01   1.57064154e-01   1.47841906e-01
 1.17564293e-02   1.37172724e-01   1.18705982e-01   1.35870185e-01
 1.33200749e-01   1.30123277e-01   1.29465267e-01   1.19954193e-01
 1.34029004e-01   1.28088097e-01   1.35134046e-01   1.37531329e-01
 1.20624176e-01   1.22589703e-01   1.26247302e-01   1.21671027e-01
 1.24472334e-01   1.25809011e-01   1.23947335e-01   1.34644158e-01
 1.32334859e-01   1.27080982e-01   1.20462986e-01   1.23574365e-01
 8.45760078e-03   1.80344876e-02   1.85495902e-02   1.18317674e-01
 1.17914265e-01   1.15593935e-01   1.17368422e-01   1.14687014e-01
 1.16749450e-01   1.16476892e-01   1.13672068e-01   1.12424754e-01
 1.11504809e-01   1.01967671e-01   1.08218391e-01   1.10530553e-01
```

```
1.09895182e-01   1.10797078e-01   1.06820525e-01   1.09462846e-01
1.05205740e-01   1.04111592e-01   1.07358827e-01   1.02793173e-01
1.05756101e-01   1.04581665e-01   1.03202582e-01   1.03327473e-01
2.01407766e-02   2.07778286e-02   2.11612944e-02   2.14067702e-02
2.20246845e-02   2.28075169e-02   2.31601784e-02   2.33458932e-02
2.34615511e-02   2.37827671e-02   2.40557182e-02   2.42552173e-02
2.44872838e-02   2.48145341e-02   2.56695209e-02   2.53681902e-02
2.52662650e-02   1.00911475e-01   1.01537406e-01   1.01422654e-01
1.00339824e-01   9.96836722e-02   9.91889457e-02   9.88925565e-02
9.80570318e-02   9.76705929e-02   9.72733168e-02   9.61679403e-02
9.67945700e-02   9.34945414e-02   9.41238871e-02   9.51440908e-02
9.48239916e-02   9.53971134e-02   9.25975172e-02   9.19479645e-02
9.31454716e-02   8.97824557e-02   9.11655073e-02   9.08395396e-02
9.06257749e-02   8.87926320e-02   8.99728398e-02   2.60089847e-02
2.63888105e-02   2.65511151e-02   2.68050276e-02   2.68907193e-02
2.70833655e-02   2.73991011e-02   2.76521352e-02   2.77839637e-02
2.81566677e-02   2.84487225e-02   2.87958485e-02   2.86279422e-02
3.02743156e-02   3.05483353e-02   2.98324305e-02   2.91424449e-02
2.95424662e-02   3.07167296e-02   2.93020097e-02   8.82818785e-02
8.80383536e-02   8.74535872e-02   8.59182479e-02   8.70566608e-02
8.69135075e-02   8.66121447e-02   8.52344712e-02   8.47282781e-02
8.40164489e-02   8.41810241e-02   8.37013296e-02   8.28397736e-02
8.31900338e-02   8.18428547e-02   8.12411740e-02   8.06071235e-02
8.02831545e-02   7.95154844e-02   8.08441925e-02   7.13435887e-02
7.91153952e-02   7.11035790e-02   7.87729700e-02   7.17394420e-02
7.22591767e-02   7.82804321e-02   7.28729326e-02   7.75948162e-02
7.32348315e-02   7.45319671e-02   7.56144841e-02   7.76918093e-02
7.49342097e-02   7.35825512e-02   7.36734238e-02   7.40647684e-02
7.60792821e-02   7.53340016e-02   7.67704119e-02   7.62856642e-02
7.66152753e-02   3.09266153e-02   3.13298617e-02   7.01509595e-02
7.03068563e-02   7.04635819e-02   3.15351062e-02   3.20274603e-02
3.18488389e-02   3.17456369e-02   3.28641196e-02   3.26051210e-02
3.23704254e-02   6.64086317e-02   6.70356913e-02   6.82317854e-02
6.74109497e-02   6.87226875e-02   6.93312448e-02   6.94180618e-02
6.90763606e-02   3.31892758e-02   3.30084575e-02   3.37610519e-02
3.41420046e-02   3.42649549e-02   3.57964723e-02   3.55677031e-02
3.48193963e-02   3.50902616e-02   3.46720315e-02   3.53633339e-02
3.44248768e-02   5.98907310e-02   6.03362537e-02   6.86469490e-02
6.12576356e-02   6.59528013e-02   6.19876230e-02   6.25433039e-02
6.48906241e-02   6.56177831e-02   6.07061827e-02   6.35405592e-02
6.75615009e-02   6.45023507e-02   6.37009709e-02   6.39677292e-02
6.41733599e-02   6.55875051e-02   6.23047465e-02   6.31116500e-02
5.88720204e-02   5.83445230e-02   6.10461667e-02   6.04994044e-02
5.92340139e-02   5.79520253e-02   5.94003405e-02   5.76276837e-02
5.70511452e-02   5.64142859e-02   5.73163743e-02   3.62346546e-02
5.60500297e-02   5.42257715e-02   3.64469641e-02   5.29209100e-02
5.26432337e-02   3.69102779e-02   3.67440441e-02   3.71633500e-02
3.77840909e-02   3.87454955e-02   4.20131431e-02   4.16774536e-02
```

```
   4.31202124e-02   3.73884583e-02   3.92530371e-02   4.13158854e-02
   4.24977918e-02   4.11508053e-02   4.04293184e-02   3.95594658e-02
   4.07624811e-02   3.74020397e-02   3.83428564e-02   3.97656617e-02
   3.78940543e-02   3.94446462e-02   4.01847487e-02   5.58520725e-02
   3.83374260e-02   5.57270658e-02   5.54407637e-02   5.48946150e-02
   5.51380741e-02   5.51210642e-02   4.02441394e-02   4.00184899e-02
   5.35555860e-02   5.35984521e-02   4.27271678e-02   5.32855497e-02
   4.26360416e-02   4.36005152e-02   5.07656494e-02   5.21571591e-02
   5.19264791e-02   4.37349341e-02   5.14291040e-02   4.40941743e-02
   4.45039024e-02   5.02345307e-02   5.01174560e-02   4.64812861e-02
   4.96789909e-02   4.60821493e-02   4.67182388e-02   4.69657305e-02
   4.72185987e-02   4.50029003e-02   4.84464610e-02   4.91526310e-02
   4.87500752e-02   4.80670006e-02   4.52431605e-02   4.51191639e-02
   4.56376799e-02   4.78897940e-02   4.34104777e-02   4.92629809e-02
   5.12922009e-02   5.21737306e-02   4.42545198e-02   5.11155832e-02
   4.93287526e-02   4.54685927e-02   4.75959289e-02   4.75548839e-02
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
--------------------------------------------------------------------------------------------------

<<==Eigen Vectors==>>
[[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  0.]
 ...,
 [ 0.  0.  0. ...,  1.  0.  0.]
 [ 0.  0.  0. ...,  0.  1.  0.]
 [ 0.  0.  0. ...,  0.  0.  1.]]
```

3.1.4 Picking the principle components

Now, we sort the eigen vectors in the descending order of eigen values (eig_vals above), which way we get the top principle components.

We can add the eig_vals and eig_vecs in a dictionary, then do a python sort method, which will sort it in descending order like below:

eigen_val_vec_pair.sort(lambda x: x[0], reverse=True)

```
In [25]: # create pairs of the eigen values and eigen vectors so that they can be descendingly s
         #of eigen values
         eigen_val_vec_pair = [(np.abs(eig_vals[i]), eig_vecs[:,i]) for i in range(len(eig_vals)

         # sort them in descending order of eigen values
         eigen_val_vec_pair.sort(key=lambda x: x[0], reverse=True)

         # check if sorted correctly
         print "++++++++++++++++++++++++++++++++++++Top 10 eigen values+++++++++++++++++++++++++
         print [eigen_val_vec_pair[i][0] for i in range(1,10)]
```

```
++++++++++++++++++++++++++++++++++++Top 10 eigen values+++++++++++++++++++++++++++++++++++++
[29.111465657226773, 26.783337098645184, 20.814719425177525, 18.100020588385302, 15.787673733771
```

3.1.5 Explained variance for picking how many components we want to choose

We still have not decided on the number of components that are required, depending on the variance. The explained variance (nothing but a percentage measure of variance) helps us decide the total percent of variance covered by each of the principle component by checking the eigen values.

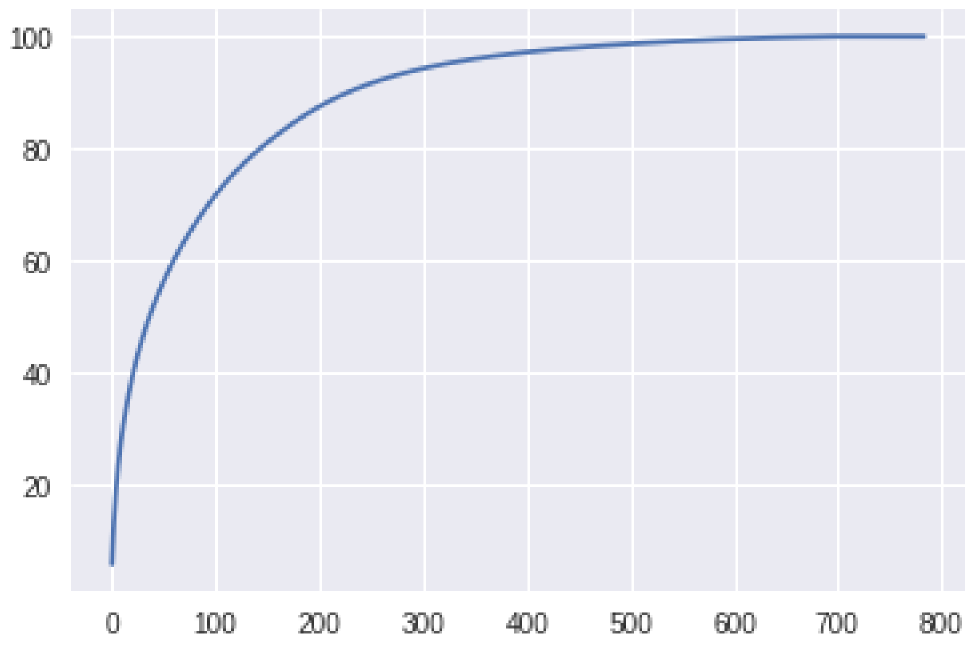
```
In [26]: tot = sum(eig_vals)
         var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
         variance_ = np.cumsum(var_exp)
```

3.1.6 How many principle components should we have/take from the above dict?

For, this, we plot a CDF that will show us a curve and we can make a point where there is not much rise/fall in the graph, and we can choose that as the number of principle components, recheck the similar plot we drew above.

```
In [27]: plt.plot(variance_)
```

```
Out[27]: [<matplotlib.lines.Line2D at 0x7f1066d4e050>]
```



Now, to explain this programatically, what we do is, take the dimension till which 85% of the variance is covered. Now, you may ask why 85?? That is not a golden standard, but it's actually the percentage of marks I wanted to get since 6th Grade in School and finally got it at 10th Grade when it mattered the most. You may want to take 60%, if that's what you scored :P

So, 85% is just a good measure of variance here because, I can see by looking at the values that, there is not much significant gain that I get by including the other 5 or 10 or 15%, which means, if you look at the plot above, the exponentian curve is getting converted into a quadratic curve with less information gain. Hence, fixated at 85%

```
In [28]: sum = 0
         for k,v in enumerate(var_exp):
             sum += v
             if sum >= 85:
                 print k
                 break
```

```
179
```

```
In [29]: sum = 0
         for k,v in enumerate(var_exp):
             sum += v
             if sum >= 90:
                 print k
                 break
```

```
228
```

As you can see above, for just 5% gain in the variance I have increased the dimensions from ~180 to ~230, which means 50 DIMENSIONS. So, I let go of 90% and stick to 85%.

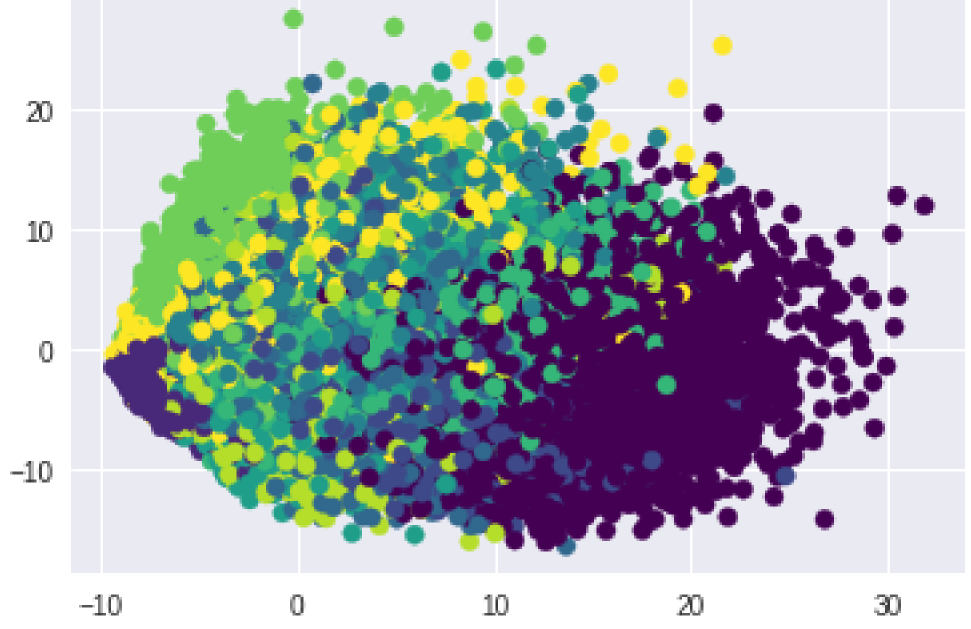
You may want to experiment with the data a lot and get to a close to appropriatae value

3.1.7 Transformation in another space/dimension

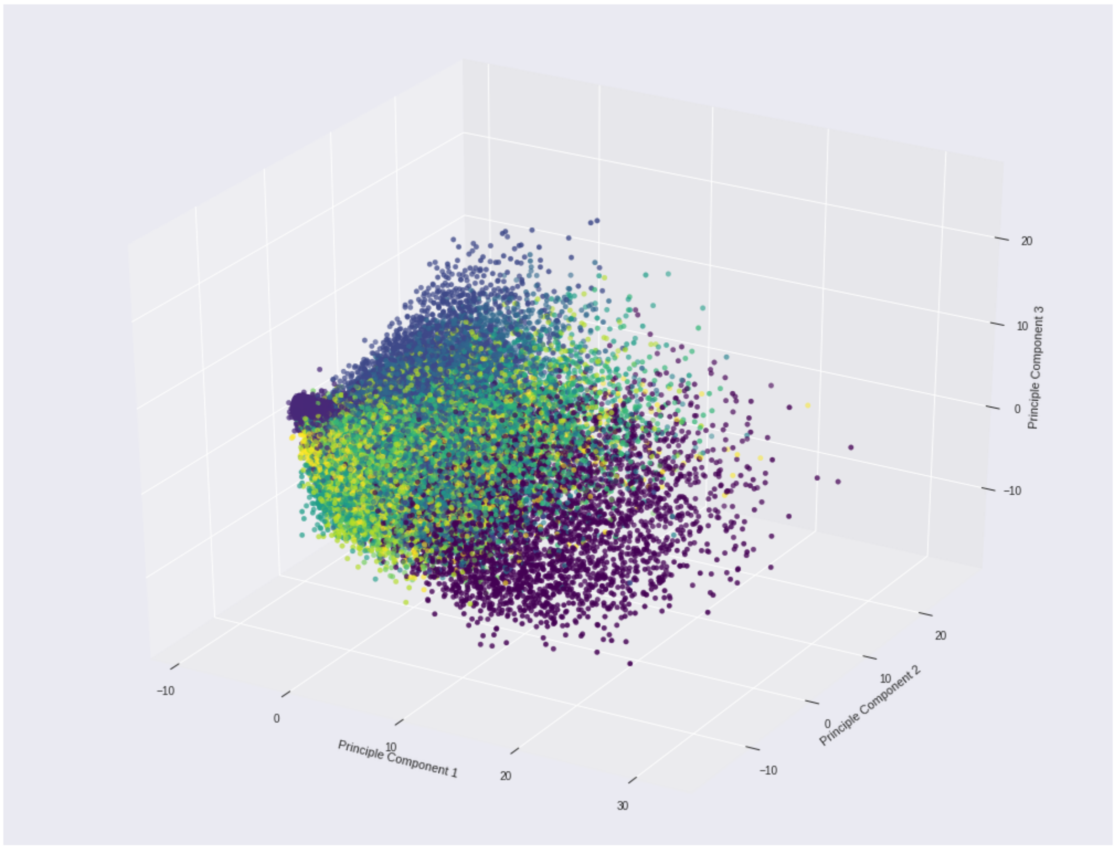
I consider this part the most important. The reason is, unless you can project the eigen vectors with the standardized matrix on a particular visible dimension (1,2,3D) the point of above explanation is useless.

Make sure to see each of the below step and re-read if unclear. I can't emphasize enough; this is the most important of all the parts.

```
In [37]: df_std.shape

Out[37]: (42000, 784)

In [42]: eigen_val_vec_pair[1][1].shape

Out[42]: (784,)

In [59]: eigen_array = np.zeros((180,784))

In [60]: eigen_array.shape

Out[60]: (180, 784)

In [61]: for i in range(180):
             eigen_array[i] = eigen_val_vec_pair[i][1]

In [63]: df_std.shape

Out[63]: (42000, 784)

In [65]: eigen_array.T.shape

Out[65]: (784, 180)

In [67]: W = df_std.dot(eigen_array.T)

In [71]: plt.scatter(W[:,0], W[:,1], c=label, cmap='viridis')
         plt.show()
```



```
In [68]: fig = plt.figure(figsize=(18, 14))
         ax = fig.add_subplot(111, projection='3d')
         ax.set_xlabel('Principle Component 1')
         ax.set_ylabel('Principle Component 2')
         ax.set_zlabel('Principle Component 3')
         ax.scatter(W[:,0], W[:,1], W[:,2], c=label, cmap='viridis')
         plt.show()
```



And with that, we are done with Math part. If it was too heavy, make sure to get the Linear Algebra basics sorted out. This is one of the simplest concepts in Machine Learning with actually very less math behind it.

So, I recommend going through Khan Academy's Linear Algebra course, at least the vectors and eigen vectors part. You can reserve matrices for later if you know the basic properties of matrices.

For, comprehensive stude, you may want to check Gilbert Strang's videos which will get you from Zero to Hero.

4 References

PCA http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html https://www.analyticsvidhya.com/blog/2016/03/practical-guide-principal-component-analysis-python/

Data warehousing and DBMS https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb/ http://highscalability.com/blog/2013/4/15/scaling-pinterest-from-0-to-10s-of-billions-of-page-views-a.html https://medium.com/@Pinterest_Engineering/shardin pinterest-how-we-scaled-our-mysql-fleet-3f341e96ca6f

Digital Marketing terms http://www.business2community.com/digital-marketing/20-must-know-digital-marketing-definitions-0797241#lva7eV0FqIrBA2a9.97

5 Note

The tracking data is certainly very huge in terms of data points but the features/dimensions may not be too many, and may not require reduction using PCA. However, I being from a **digital marketing background** wanted to explain the concepts with what best I can do and also keep the users engaged. :)

5.0.1 PS:

I will also make sure to add some custom images so that the sections are more crisp and clear.

Thanks for reading. :)