PCA_on_MNIST_dataset

July 30, 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
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.preprocessing import StandardScaler

        import plotly.offline as py
        py.init_notebook_mode(connected=True)
        import plotly.graph_objs as go
        import plotly.tools as tls

        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]: 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

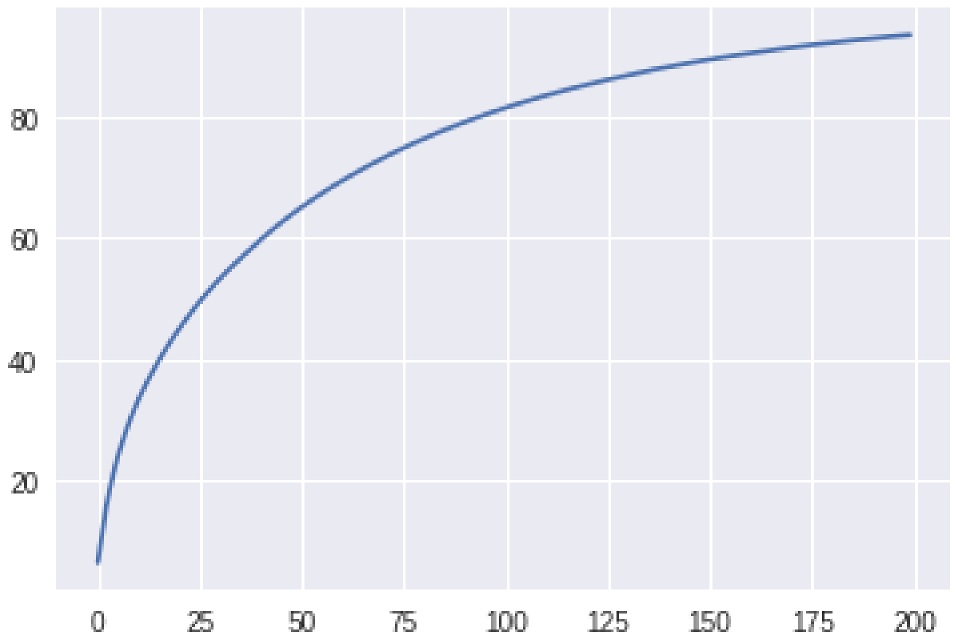
```
In [9]: pca = PCA(n_components=200)
```

```
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 0x7f1b87a5dc50>]
```



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=75)
```

```
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]: trace0 = go.Scatter(
             x = df_pca[:,0],
             y = df_pca[:,1],
             name = label,
             hoveron = label,
             mode = 'markers',
             text = label,
             showlegend = False,
             marker = dict(
                 size = 8,
                 color = label,
                 colorscale ='Jet',
                 showscale = False,
                 line = dict(
                     width = 2,
                     color = 'rgb(255, 255, 255)'
                 ),
                 opacity = 0.8
             )
         )
         data = [trace0]

         layout = go.Layout(
             title= 'Principal Component Analysis (PCA)',
             hovermode= 'closest',
             xaxis= dict(
                  title= 'First Principal Component',
                 ticklen= 5,
                 zeroline= False,
                 gridwidth= 2,
             ),
             yaxis=dict(
                 title= 'Second Principal Component',
                 ticklen= 5,
                 gridwidth= 2,
             ),
             showlegend= True
         )


         fig = dict(data=data, layout=layout)
         py.iplot(fig, filename='styled-scatter')
```

The above plot however does not show anthing significant and all the efforts put by us might feel wasted, but we actually learned where not to use PCA. A wise person once said, "It's important what not to do, rather than doing what all to do!"

We will see the t-SNE notebook which makes it much more visible and the distinction between the points is very clear.

3 Bonus (Math behind PCA)

3.0.1 Covariance matrix

Let's start by calculating the covariance matrix, which is nothing but the normalized/standardized matrix, remember we talked above why normalization is necessary.

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

3.0.2 Eigen Decomposition

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)**

3.0.3 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)

3.0.4 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.

3.0.5 Transformation in another space/dimension

Now, depending on the components, we will transform the normalized data into that space, ex 5d, 6d, etc and plot a scatter plot.

Note: (In process)

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/ http://www.math.union.edu/~jaureguj/PCA.pdf

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 but the features are not too many, and may not require reduction using PCA but, to make the plot more interesting, I had to choose this example. :) One more key point is that, PCA works on numerical data, another reason the example is not a really great fit.

```
In [ ]:
```