Sane's STATS Academy of Statistics

Dimension Reduction

SVD & PCA

Dimension Reduction

- Dimension Reduction Algorithms are a part of unsupervised learning algorithms
- Goals of Dimension Reduction Algorithms are:
  - to find structure within predictors(features)
  - to aid in visualization
- We will learn in this section Singular Value Decomposition and Principal Component Analysis

Why Dimension Reduction?

- Statistical Purpose: For finding a new set of variables that are uncorrelated and explain as much as variance as possible
- Data Compression Purpose: For finding the best matrix created with fewer variables that explain the original data

Singular Value Decomposition

- SVD is a method for identifying and ordering the dimensions for which the data points exhibit the most variation.
- Once we have identified where the most variation is, it's possible to find the best approximation of the original data points using fewer dimensions.
- What makes SVD practical for NLP like applications is that you can simply ignore variation below a particular threshold to massively reduce your data but be assured that the main relationships of interest have been preserved.

SVD & PCA

- If M is a matrix, then the matrix M ($m \times n$) can be factorized as

  $$M = U\Sigma V^T,$$

  where U is $m \times m$ where $U^T U = I$ and the columns of *U* are orthonormal eigenvectors of AA^T

  Σ is $m \times n$ diagonal matrix

  V is $n \times n$ where $V^T V = I$ and the columns of *V* are orthonormal eigenvectors of $A^T A$

  V^T is transpose of matric V.

- Principal Components are obtained by first scaling and centering the data and then extracting the V matrix and multiplying it with the scaled data

Principal Component Analysis

- Finds a linear combination of variables to create principal component
- PCA maintains most of variance from the original data in Principal Components created
- Principal Components are uncorrelated (i.e. orthogonal to each other) with each other
- If our data is having n observations and p variables then we can have at most $\mathrm{min}imum\ of\ (n, p)$ principal components

A two dimensional example

- Consider the data of two variables in which we find fit a regression line to it.
- This regression line is determined such that we have minimum residuals.
- This line can be said to be the first component of this data.
- Once the line is fitted, each point can be mapped on the line and perpendicular projection can be drawn of all the points on the line.
- This projected value can be referred to as component scores or factor scores.

Steps in PCA

1. Data Preparation: Make the data completely numerical. If, it is not numerical then dummy variables can be replaced with categorical variables
2. Centering and Scaling(if needed): Subtract each value mean of its respective column and (optional) divide values by their respective column Standard Deviations
3. Calculate Covariance Matrix / Correlation matrix
4. Calculate eigenvalues and eigenvectors of the covariance matrix / correlation matrix
5. Extract the components with maximum variation
6. Calculate the scores

Example: Milk (dataset in package flexclust in R)

- The data set contains the ingredients of mammal's milk of 25 animals.
- A data frame with 25 observations on the following 5 variables (all in percent)
  - water
  - protein
  - fat
  - lactose
  - ash

PCA in Python

sklearn.decomposition.PCA()

```
from sklearn.decomposition import PCA
pca = PCA()
principalComponents = pca.fit_transform(milkscaled)
```

Python Program

```
In [78]: milk.head()
Out[78]:
           water  protein  fat  lactose   ash
HORSE       90.1      2.6  1.0      6.9  0.35
ORANGUTAN   88.5      1.4  3.5      6.0  0.24
MONKEY      88.4      2.2  2.7      6.4  0.18
DONKEY      90.3      1.7  1.4      6.2  0.40
HIPPO       90.4      0.6  4.5      4.4  0.10

In [79]: from sklearn.preprocessing import StandardScaler
    ...: scaler = StandardScaler()
    ...: scaler.fit(milk)
    ...: milkscaled=scaler.transform(milk)
    ...:
    ...: from sklearn.decomposition import PCA
    ...: pca = PCA(svd_solver = 'auto')
    ...: principalComponents = pca.fit_transform(milkscaled)
```

Sane's STATS Academy of Statistics

Principal Components Generated

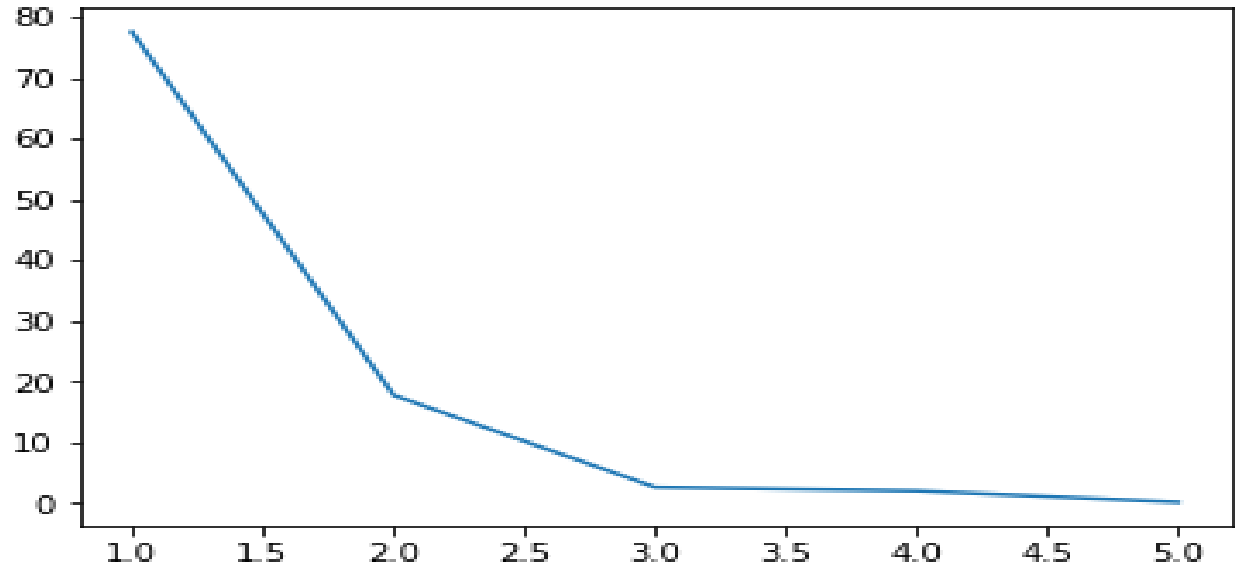
```
In [81]: df_plot = pd.DataFrame(principalComponents,
    ...:                        columns = ['PC1', 'PC2','PC3','PC4','PC5'],
    ...:                        index = milk.index)

In [82]: df_plot
Out[82]:
                PC1       PC2       PC3       PC4       PC5
HORSE     -2.435175 -0.231478  0.330150  0.307664 -0.048184
ORANGUTAN -2.258736 -0.693148  0.029035 -0.081057  0.000532
MONKEY    -2.335861 -0.663328  0.101257  0.247997 -0.010109
DONKEY    -2.318677 -0.273815  0.109881 -0.092873  0.006769
HIPPO     -2.060294 -1.019199 -0.762436 -0.510091 -0.029382
CAMEL     -1.301536  0.137303 -0.198742 -0.282160  0.014668
BISON     -1.282742  0.622667  0.253785  0.065148  0.044486
BUFFALO   -0.505068  0.088235  0.008383  0.133774 -0.083806
GUINEA PIG 0.255502  0.287373 -0.881857 -0.013100  0.016484
CAT        0.051008  0.470136 -0.355676  1.050970 -0.159793
FOX       -0.429017  0.459312  0.135181  0.236941  0.024309
LLAMA     -1.364034  0.309224  0.228597 -0.088242  0.024345
MULE      -2.012658 -0.192807 -0.131325 -0.242899  0.025286
PIG        0.000100  0.788138 -0.334070 -0.056860  0.056423
ZEBRA     -1.390928 -0.012660  0.116929 -0.287079  0.004502
SHEEP     -0.513120  0.321365  0.078943 -0.020171  0.035616
DOG        0.982829  0.725239 -0.331552  0.169590  0.063456
ELEPHANT  -0.302294 -1.079878  1.004594 -0.114093 -0.000905
RABBIT     2.798474  2.288793  0.161914 -0.394421  0.021719
RAT        1.309609  0.774032  0.159709  0.015485  0.064371
DEER       2.212053  0.345036  0.211472  0.064829 -0.044332
REINDEER   2.345616  0.310003  0.205686  0.108605 -0.031051
WHALE      2.892787  0.721121  0.055943 -0.276172 -0.054319
SEAL       4.133773 -2.200801 -0.067215 -0.374615 -0.190501
DOLPHIN    3.528390 -2.280863 -0.128585  0.432830  0.249417
```

Sane's STATS Academy of Statistics

Variance Explained

```
In [84]: import matplotlib.pyplot as plt
    ...: y = pca.explained_variance_ratio_ * 100
    ...: x = np.arange(1,6)
    ...: plt.plot(x,y)
    ...: plt.show()
```



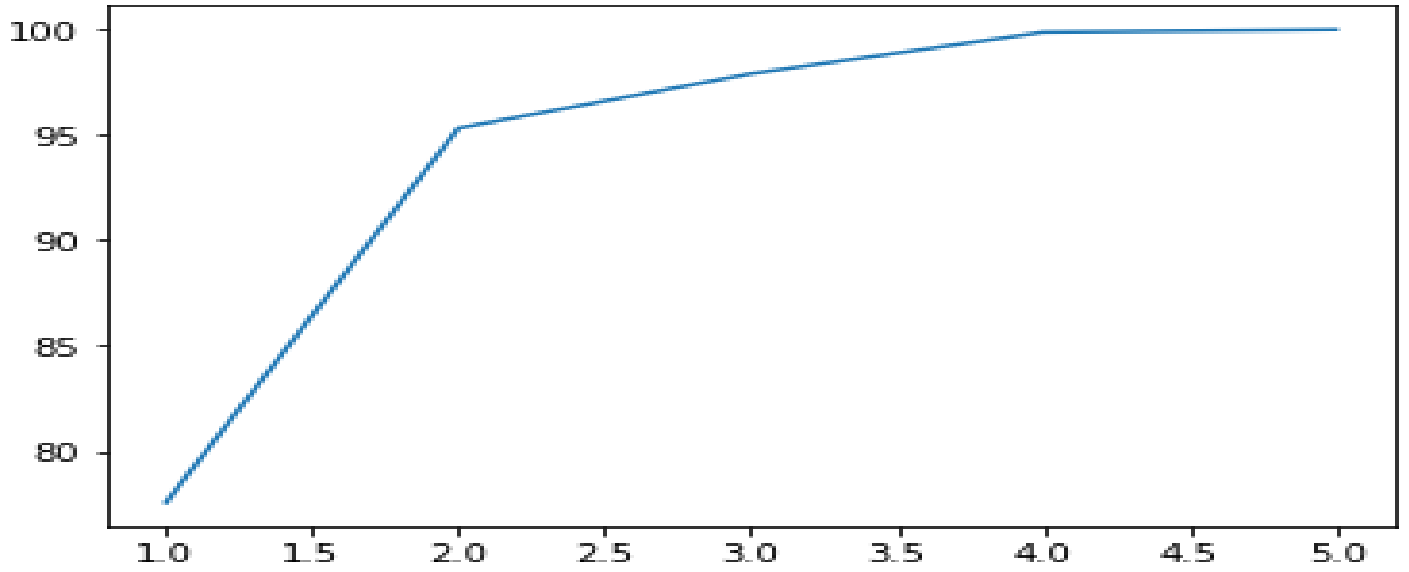

```
In [85]: print(pca.explained_variance_ratio_ * 100)
[77.57590469 17.74794969  2.57838496  1.97162103  0.12613963]
```

- We can observe that the maximum variance(77.57% + 17.74 = 96.67%) has been already explained by the first principal component

Sane's STATS Academy of Statistics

Cumulative Variation Explained

```
In [86]: import matplotlib.pyplot as plt
    ...: y = np.cumsum(pca.explained_variance_ratio_ * 100)
    ...: x = np.arange(1,6)
    ...: plt.plot(x,y)
    ...: plt.show()
```




```
In [87]: np.cumsum(pca.explained_variance_ratio_ * 100)
Out[87]:
array([ 77.57590469,  95.32385438,  97.90223934,  99.87386037,
       100.        ])
```

Sane's STATS Academy of Statistics