COMP755-Lect15

October 9, 2018

1 COMP 755

Plan for today

1. Principal Component Analysis (PCA)
2. Eigen-decomposition and Singular Value Decomposition (SVD)
3. Partial Correlation and Partial Least Squares

```
In [1]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline
        from sklearn.datasets import fetch_lfw_people
        lfw_people = fetch_lfw_people(min_faces_per_person=70, data_home='.')
        T, h, w = lfw_people.images.shape
        print ("Samples: ",T,"Height: ",h,"Width: ",w,"Features (Height*Width):",h*w)
        X = lfw_people.data
        L= 25
        data = X.reshape((T,h*w)).transpose()
        print ("Size of Data (features,samples): ", data.shape)
        def show(x):
            plt.imshow(x,cmap=plt.cm.gray)
            plt.axis('off')

        for i in range(25):
            plt.subplot(5,5,i+1)
            show(data[:,i].reshape(h,w))
```

```
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976012
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976009
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976006
Downloading LFW data (~200MB): https://ndownloader.figshare.com/files/5976015
```

```
Samples:  1288 Height:  62 Width:  47 Features (Height*Width): 2914
Size of Data (features,samples):  (2914, 1288)
```

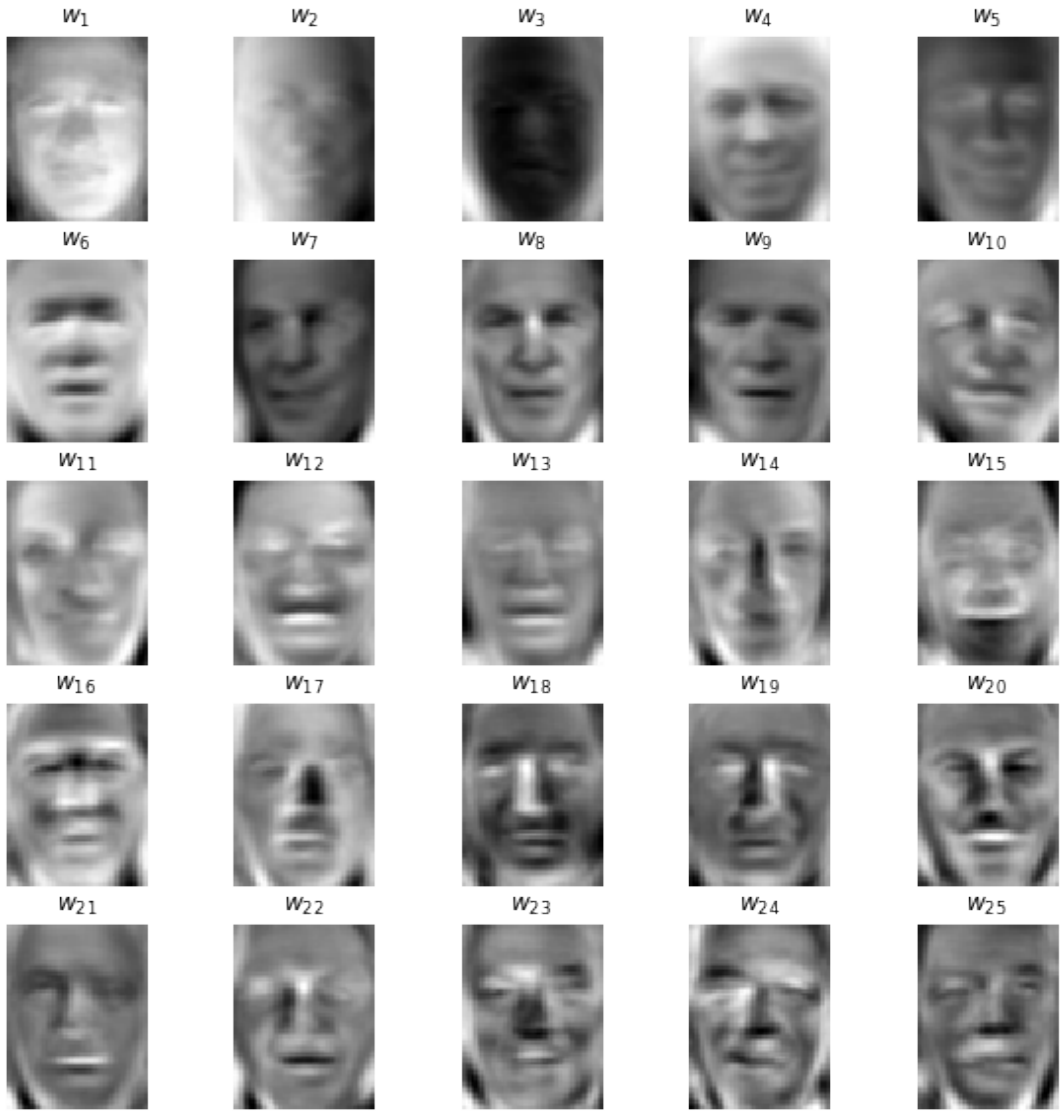
```
In [5]: def pca(X,L):
            print ("Data size (features,samples):", X.shape)
            T = X.shape[1]
            mu = np.mean(X,axis=1)
            print ("Mean size:", mu.shape)
            X = X - mu[:,np.newaxis]
            C = 1./float(T)*np.dot(X,X.T)
            print ("Covariance size:", C.shape)
            evals,evecs = np.linalg.eig(C,)
            W = evecs[:,:L]
            z = np.dot(W.T,X)
            return W,z,mu,evals

        def project(x,mu,W):
            z = np.dot(x - mu,W)
            reconstruction = np.dot(W,z) + mu
            return z,reconstruction

        W,z,mu,evals = pca(data,  L)
        print ("Size of matrix of principal components (features,L):", W.shape)
        eigenfaces = np.reshape(W,(h,w,L))
        plt.figure(figsize=(10,10))
        subplot_d = np.sqrt(L)
        for i in range(L):
            plt.subplot(subplot_d,subplot_d,i+1)
            plt.imshow(eigenfaces[:,:,i],cmap=plt.cm.gray)
```

```
            plt.title('$w_{'+str(i+1)+'}$')
            plt.axis('off')

Data size (features,samples): (2914, 1288)
Mean size: (2914,)
Covariance size: (2914, 2914)
Size of matrix of principal components (features,L): (2914, 25)
```

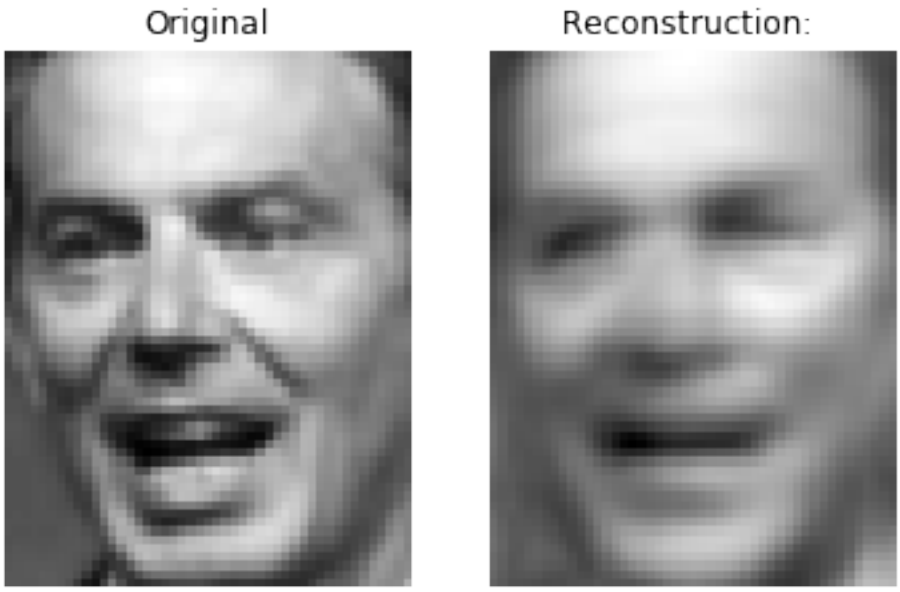



```
In [7]: xt = data[:,1]
        print ("Size of data sample: ",xt.shape)
        z,reconstruction = project(xt,mu,W)
```

```
        print ("Size of z: ",z.shape)
        print ("Compression rate: ", float(xt.shape[0])/float(z.shape[0]))
        print ("Size of reconstruction: ", reconstruction.shape)

        plt.subplot(1,2,1)
        show(xt.reshape(h,w))
        plt.title('Original')
        plt.subplot(1,2,2)
        show(reconstruction.reshape(h,w))
        plt.title('Reconstruction: ');
```

```
Size of data sample:  (2914,)
Size of z:  (25,)
Compression rate:  116.56
Size of reconstruction:  (2914,)
```



2 Principal Component Analysis

Principal Component Analysis optimizes objective

$$J(\mathbf{W}, \mathbf{Z}) = \frac{1}{N} \sum_{t=1}^{N} \underbrace{\|\mathbf{x}_t - \mathbf{W}\mathbf{z}_t\|^2}_{\text{reconstruction error}}$$

to find optimal $\mathbf{W}$ of size $d \times L$ and $\mathbf{z}_t$ vectors of length l.

Intuitively, reconstruction $\mathbf{W}\mathbf{z}_t$ approximates $\mathbf{x}_t$.

Optimal solutions for $\mathbf{W}$ and $\mathbf{z}_t$ are given by: 1. $\mathbf{W}^*$ composed of top L eigenvectors of data covariance matrix 2. $\mathbf{z}_t^*$ is projection of $\mathbf{x}_t$ onto space spanned by columns of $\mathbf{W}^*$

3 Principal Component Analysis -- implementation details

Given Data $= \{\mathbf{x}_t : t = 1, ...T\}$, where samples $\mathbf{x}_t$ are d-long vectors, if you want to learn how to compress these vectors into shorter L long representations $\mathbf{z}_t$ 1. compute covariance of the dataset 1. centered data : $\hat{\Sigma} = \frac{1}{N}\sum_{t=1}^{N}\mathbf{x}_t\mathbf{x}_t^T$, 1. non-centered data: $\hat{\Sigma} = \frac{1}{N}\sum_{t=1}^{N}(\mathbf{x}_t - \bar{})(\mathbf{x}_t^T - \bar{})$, where $\bar{} = \frac{1}{N}\sum_t \mathbf{x}_t$, 2. extract top L eigenvectors of $\hat{\Sigma}$ and store them in $\mathbf{W}^*$ 3. project data to the space spanned by those eigenvectors
1. centered data: $\mathbf{z}_t^* = (\mathbf{W}^*)^T\mathbf{x}_t$ 2. non-centered data: $\mathbf{z}_t^* = (\mathbf{W}^*)^T(\mathbf{x}_t - \bar{}) + \bar{}$

```
In [8]: def pca(X,L):
            T = X.shape[1]
            mu = np.mean(X,axis=1)
            X = X - mu[:,np.newaxis]
            C = 1./float(T)*np.dot(X,X.T)
            evals,evecs = np.linalg.eig(C)
            W = evecs[:,:L]
            z = np.dot(X.T,W)
            return W,z,mu,evals
```

4 Projection onto an orthonormal basis is easy

$$J(\mathbf{W},\mathbf{Z}) = \frac{1}{N}\sum_{t=1}^{N}\underbrace{\|\mathbf{x}_t - \mathbf{W}\mathbf{z}_t\|^2}_{\text{reconstruction error}}$$

Finding optimal $\mathbf{z}_t$ is really just regression onto $\mathbf{W}$

$$\|\mathbf{x}_t - \mathbf{W}\mathbf{z}_t\|^2$$

but we also know that $\mathbf{W}$ is orthonormal, $\mathbf{W}^T\mathbf{W} = I$

If you take gradient in matrix form and equate it to zero vector

$$\begin{aligned}
(\mathbf{x}_t - \mathbf{W}\mathbf{z})^T\mathbf{W} &= 0 \\
\mathbf{x}_t^T\mathbf{W} - \mathbf{z}_t^T\underbrace{\mathbf{W}^T\mathbf{W}}_{I} &= 0 \\
\mathbf{x}_t^T\mathbf{W} - \mathbf{z}_t^T &= 0 \\
\mathbf{z}_t &= \mathbf{W}^T\mathbf{x}_t
\end{aligned}$$

Hence, if you are minimizing

$$\|\mathbf{x}_t - \mathbf{W}\mathbf{z}_t\|^2$$

and $\mathbf{W}^T\mathbf{W} = I$ then

$$\mathbf{z}_t = \mathbf{W}^T\mathbf{x}_t$$

```
In [9]: def project(x,mu,W):
            z = np.dot(W.T,x - mu[:,np.newaxis])
            reconstruction = np.dot(W,z) + mu[:,np.newaxis]
            return z,reconstruction
```

5 PCA and eigen-decomposition

PCA uses eigen-decomposition of covariance of a centered data matrix

$$\frac{1}{N}\mathbf{X}\mathbf{X}^T = \hat{\Sigma} = \mathbf{V}\text{diag}\left(\mathbf{d}\right)\mathbf{V}^T$$

where eigenvectors are columns of $\mathbf{V}$ and $\mathbf{d}$ is a vector of eigenvalues (eigenvector $\mathbf{v}_i$ corresponds to eigenvalue d_i)

However, we do not need all of the eigenvectors -- we just need the top L of them.

Efficient methods to compute top L eigenvectors exist. They use *Lanczos* power method.

6 Power method for computing top eigenvector

To obtain top eigenvector of matrix $\mathbf{A}$, start with a randomly generated vector $\mathbf{b}^{(0)}$ and iterate

$$\mathbf{b}^{(k+1)} = \frac{\mathbf{A}\mathbf{b}^{(k)}}{\left\|\mathbf{A}\mathbf{b}^{(k)}\right\|}$$

The way to think about this is:

$$\begin{aligned}\mathbf{A}\mathbf{b}^{(k)} &= \frac{1}{C_k}(\mathbf{V}\mathbf{D}\mathbf{V}^T)^k\mathbf{b}^{(0)} = \frac{1}{C_k}\mathbf{V}\mathbf{D}^k\mathbf{V}^T\mathbf{b}^{(0)} \\ &= \frac{1}{C_k}\mathbf{V}\text{diag}\left(\mathbf{d}^k\right)\mathbf{V}^T\mathbf{b}^{(0)} = \sum_i \frac{d_i^k}{C_k}\mathbf{v}_i\left(\mathbf{v}_i^T\mathbf{b}^{(0)}\right)\end{aligned}$$

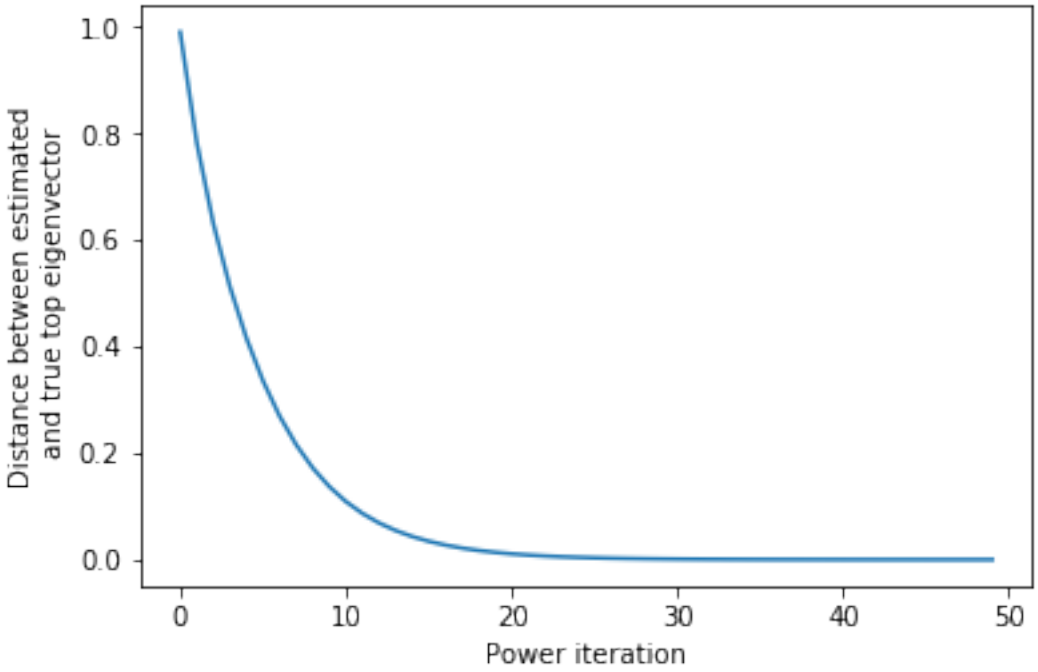
Note that $(\frac{\mathbf{d}_j}{\mathbf{d}_1})^k$, for $j > 1$ tends to 0 as k grows, hence contribution from $\mathbf{v}_1$ gets proportionally larger.

More sophisticated methods based on Lanczos method.

```
In [11]: np.random.seed(1)
         A = np.random.randn(20,20)
         S = np.dot(A.transpose(),A)
         evals,evecs = np.linalg.eig(S)
         dist = []
         v = np.random.randn(20,1)
         for i in range(50):
             pv = v
             t = np.dot(S,v)
             v = t/np.linalg.norm(t)
             dist.append(np.min([np.linalg.norm(v.T-evecs[:,0]),np.linalg.norm(v.T+evecs[:,0])])
```

```
        plt.plot(dist)
        plt.xlabel('Power iteration')
        plt.ylabel('Distance between estimated\n and true top eigenvector')
        print (v.T)
        print (evecs[:,0])
```

```
[[-0.22590363  0.23195047 -0.03786408  0.37374716 -0.05109231 -0.40041844
  -0.11526983 -0.06139302 -0.24907054  0.27717813 -0.15557454 -0.20383893
   0.01677933 -0.19183472 -0.34136459  0.15406141  0.05169526  0.05617754
   0.39262341  0.17916044]]
[-0.22590443  0.23195115 -0.0378667   0.37374746 -0.05109332 -0.40042063
 -0.11526652 -0.06139742 -0.24906632  0.27717669 -0.15557383 -0.20384345
  0.01678263 -0.19183597 -0.3413595   0.15406124  0.05170373  0.05617577
  0.39262511  0.17915895]
```



7 Using Singular Vector Decomposition to compute PCA

Even with power methods, we still need to compute the covariance matrix.

This matrix is of size $d \times d$ (2914×2914 in our faces example). This can be expensive.

We will now see how to bypass computation of covariance matrix using a different decomposition.

For brevity, we will assume centered data, so the covariance is given by:

$$\hat{\Sigma} = \frac{1}{N}\mathbf{X}\mathbf{X}^T$$

8 Singular Vector Decomposition

Frequently used in machine learning as a means to reduce computation time.

Let $\mathbf{X}$ be a matrix of size $d \times N$ and $K = \min(d, N)$ then we can find decomposition

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

where $\mathbf{U} : d \times K$ and $\mathbf{V} : K \times N$ are orthonormal and $\mathbf{S} : K \times K$ is diagonal.

These matrices have their names 1. columns of $\mathbf{U}$ left singular vectors 2. diagonal of $\mathbf{S}$ contains singular values 3. columns of $\mathbf{V}$ right signular vectors

9 Singular Vector Decomposition

$$\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

Why is this useful?

Remember we want to compute eigenvectors of covariance matrix

$$\begin{aligned} \mathbf{X}\mathbf{X}^T &= (\mathbf{U}\mathbf{S}\mathbf{V}^T)(\mathbf{U}\mathbf{S}\mathbf{V}^T)^T \\ &= \mathbf{U}\mathbf{S}\underbrace{\mathbf{V}^T\mathbf{V}}_{I}\mathbf{S}^T\mathbf{U}^T \\ &= \mathbf{U}\mathbf{S}\mathbf{S}^T\mathbf{U}^T = \mathbf{U}\mathbf{S}^2\mathbf{U}^T \end{aligned}$$

Hence, 1. left singular vector of $\mathbf{X}$, $\mathbf{U}$ are eigenvectors of $\mathbf{X}\mathbf{X}^T$ 2. right singular vector of $\mathbf{X}$, $\mathbf{V}$ are eigenvectors of $\mathbf{X}^T\mathbf{X}$

Importantly, computation of top k singular values can be accomplished in $O(dN\log(k))$ time.

10 Singular Vector Decomposition and low-rank representation

Low-rank representation of a matrix is obtained by using only k top singular vectors

1. Full rank uses all singular vectors (1..K):

$$\mathbf{X} = \sum_{i=1}^{k} \mathbf{u}_i s_i \mathbf{v}_i^T + \sum_{i=k+1}^{K} \mathbf{u}_i s_i \mathbf{v}_i^T$$

2. Low rank approximation

$$\hat{\mathbf{X}} = \sum_{i=1}^{k} \mathbf{u}_i s_i \mathbf{v}_i^T$$

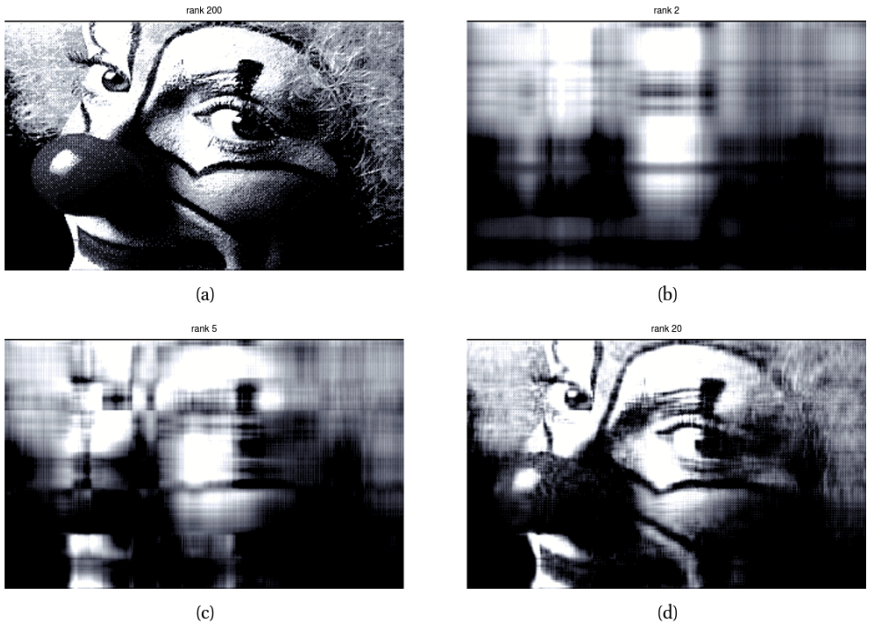


Figure 12.9 Low rank approximations to an image. Top left: The original image is of size 200×320, so has rank 200. Subsequent images have ranks 2, 5, and 20. Figure generated by `svdImageDemo`.

11 Model selection for subspace models

Since the subspace models are unsupervised there is no labeling available to evaluate test error.

However, we can evaluate reconstruction error (negative log-likelihood) on a held out validation set.

```
In [13]: from sklearn.decomposition import TruncatedSVD

         def pca(X,L):
             print ("Data size (features,samples):", X.shape)
             T = X.shape[1]
             mu = np.mean(X,axis=1)
             print ("Mean size:", mu.shape)
             X = X - mu[:,np.newaxis]
             C = 1./float(T)*np.dot(X,X.T)
             print ("Covariance size:", C.shape)
             evals,evecs = np.linalg.eig(C,)
             W = evecs[:,:L]
             z = np.dot(W.T,X)
             return W,z,mu,evals

         from sklearn.cross_validation import train_test_split
         print (data.shape)
         X_train, X_test = train_test_split(X,  test_size=0.75, random_state=1)
```

```
        X_train = X_train.T
        X_test = X_test.T
        n_test = X_test.shape[1]

        W,_,mu,evals = pca(X_train,200)
```

```
/Users/jgs/anaconda3/lib/python3.6/site-packages/sklearn/cross_validation.py:41: DeprecationWarn
  "This module will be removed in 0.20.", DeprecationWarning)
```

```
(2914, 1288)
Data size (features,samples): (2914, 322)
Mean size: (2914,)
Covariance size: (2914, 2914)
```

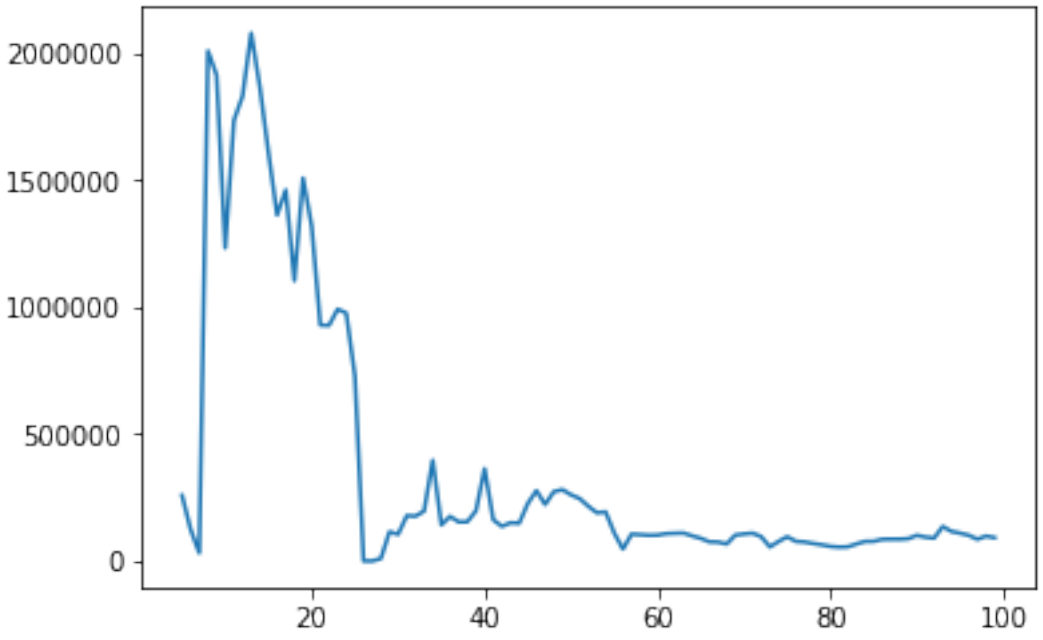
```
In [16]: err = []
         for L in range(5,100):
             _,reconstruction = project(X_test,mu,W[:,:L])
             residual = X_test - reconstruction
             err.append(1.0/n_test*np.sum(np.sum(residual)**2.0))
             print (str(L),str(err[-1]))
         plt.plot(range(5,100),err)
```

```
5 258536.660935
6 123849.620199
7 31704.1006288
8 2009500.95888
9 1913703.4588
10 1233100.63248
11 1735587.44837
12 1831167.25135
13 2079809.2403
14 1868607.45403
15 1609051.30575
16 1362993.46202
17 1461486.02011
18 1104221.42485
19 1508481.81866
20 1318684.42445
21 929273.800391
22 927974.01384
23 991649.784732
24 976703.119252
25 728526.444532
26 252.881516735
27 26.5800277204
28 9312.41559047
```

29 114817.117593
30 105030.055355
31 178246.973554
32 176435.952599
33 196372.990799
34 396685.915579
35 142212.060521
36 175532.843522
37 153981.823665
38 154424.030731
39 197259.475947
40 363651.30777
41 162681.132346
42 136204.441427
43 150392.411962
44 148599.214754
45 226541.77006
46 276996.683808
47 222499.272912
48 272817.24225
49 281058.525086
50 260561.723683
51 245882.182782
52 215489.020166
53 189823.524368
54 193356.029056
55 109870.086838
56 46976.7605798
57 105659.295659
58 103610.349172
59 101202.93818
60 101816.915556
61 107725.374924
62 109063.550093
63 110202.194828
64 98938.5153223
65 89213.3628886
66 75369.0883256
67 74519.0112211
68 66741.1519877
69 101862.558365
70 106374.594468
71 109596.7018
72 95567.1506707
73 55302.0726675
74 76721.0772501
75 95529.0978379
76 77327.6868741

```
77 75020.7436559
78 68759.1820176
79 63725.6684926
80 57385.1399005
81 54623.8871036
82 55645.3619488
83 66774.0389729
84 77098.4495505
85 77399.1573067
86 85232.8955739
87 85765.7091755
88 85820.0446469
89 87746.9588513
90 100841.980332
91 93995.9347127
92 90174.1269373
93 135817.040789
94 116326.551884
95 109517.599673
96 101174.632675
97 85515.3113658
98 98049.8134724
99 91785.2106327
```

Out[16]: [<matplotlib.lines.Line2D at 0x1a1d18a470>]



12 Correlation

Correlation of two vectors

$$\text{corr}(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{N} \frac{x_j - \mu_x}{s_x} \frac{y_j - \mu_y}{s_y}$$

where $\mu_x = \frac{1}{N}\sum_j x_j$, $\mu_y = \frac{1}{N}\sum_j y_j$, $s_x = \sqrt{(\sum_j x_j - \mu_x)^2}$, $s_y = \sqrt{(\sum_j y_j - \mu_y)^2}$

Complicated!

Let's assume that $\mathbf{x}$ and $\mathbf{y}$ are centered, means are 0. Further, assume that norm of $\mathbf{x}$ and $\mathbf{y}$ are 1.

More compactly: $\mu_x = \mu_y = 0$ and $s_x = s_y = 1$.

Then correlation is

$$\text{corr}(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y}$$

Also

$$\text{corr}(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T\mathbf{y} = \cos(\text{angle}(\mathbf{x}, \mathbf{y}))$$

So, here are couple of observations about correlation: 1. Correlation between normalized vectors is equivalent to cosine of angle between them 2. Orthonormal vectors have pairwise correlation of 0.

13 Problems with correlation

Correlation only tells you about a **linear** relationship between **two** variables.

However, the correlation between two variables may arise due to a third variable.

For example 1. eye-sight gets worse with age 2. hair becomes gray with age

Correlation between eye-sight test and number of gray hairs may suggest that one drives the other. But we know that age is the cause of both.

Common saying "Correlation does not imply causation" reflects the idea that the correlation might arise to a shared cause.

So, how do we fix correlation to account for additional factors?

14 Partial correlation

Partial correlation asks whether two variables are correlated after we remove effects of a third

If we fit model:

$$\begin{aligned} x &= \beta_x \mathbf{z} + r_x \\ y &= \beta_y \mathbf{z} + r_y \\ r_x &\sim \mathcal{N}(\mu_x, \sigma_x^2) \\ r_y &\sim \mathcal{N}(\mu_y, \sigma_y^2) \end{aligned}$$

then we can ask how correlated are $\mathbf{x} - \beta_x\mathbf{z}$ and $\mathbf{y} - \beta_y\mathbf{z}$.

15 Today

1. PCA in greter detail
2. Power methods
3. SVD and low-rank representations
4. Model selection for PCA
5. Correlation and partial correlation