Lecture 18 - Mar 11

Principal Components Analysis

Reading

- *Data Mining and Machine Learning*
  - 7.1-2 Principal Components Analysis

- *Elements of Statistical Learning*
  - 14.5.1 Principal Components

Upcoming Deadlines

Homework 3 [Mar 15]

Midterm Exam [Take-home exam: Mar 14-16]
Material: Lectures 1-17
Topics: Regression and Classification

Columns are features (samples of random variables) ← AKA attributes, properties, dimensions, variables, fields

$$D = \begin{pmatrix} X_1 & X_2 & \cdots & X_d \\ x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & & & \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix} \begin{matrix} \\ \leftarrow x_1 \\ \leftarrow x_2 \\ \\ \leftarrow x_n \end{matrix}$$

rows are datapoints

AKA examples, records, instances, feature-vectors, tuples, objects

$$x_i = \begin{pmatrix} x_{i1} \\ \vdots \\ x_{id} \end{pmatrix} = (x_{i1} \cdots x_{id})^T \in \mathbb{R}^d$$

If $e_j = (0, \ldots, 0, \underbrace{1}_{j\text{th entry}}, 0, \ldots, 0)$, we call $e_1, \ldots, e_d$ the **standard basis** of the d-dimensional space

vectors of the directions of the standard coordinate axes.

In 2D: $\begin{pmatrix}1\\0\end{pmatrix}$ $\begin{pmatrix}0\\1\end{pmatrix}$

In 3D: $\begin{pmatrix}1\\0\\0\end{pmatrix}$ $\begin{pmatrix}0\\1\\0\end{pmatrix}$ $\begin{pmatrix}0\\0\\1\end{pmatrix}$

Coordinates in each dimension

$$\Rightarrow \quad x_i = x_{i1}e_1 + x_{i2}e_2 + \ldots + x_{id}e_d = \sum_{j=1}^{d} x_{ij} e_j$$

$$\Rightarrow \quad x_i = x_{i1}e_1 + x_{i2}e_2 + \dots + x_{id}e_d = \sum_{j=1}^{} x_{ij} e_j$$

all points x_i are linear combinations of the basis vectors.

For $v_1, \dots, v_k \in \mathbb{R}^m$, a linear combination is

$$c_1 v_1 + \dots + c_k v_k$$

where $c_1, \dots, c_k \in \mathbb{R}$ are scalars

$$\text{span}(v_1, \dots, v_k) = \{\text{all linear combinations of } v_1, \dots, v_k\}$$

$$\text{if } = \mathbb{R}^m, \quad v_1, \dots, v_k \text{ span } \mathbb{R}^m$$

if $c_1 v_1 + \dots + c_k v_k = 0$ implies $c_1 = c_2 = \dots = c_k = 0$, the vectors are linearly independent

↳ if some v_i can be written as a linear combination of the others, they are linearly dept

A basis of a subspace $S \in \mathbb{R}^m$ is a linearly indept

A basis of a subspace $S \subseteq \mathbb{R}^m$ is a linearly independent set of vectors $v_1, \ldots, v_k$ s.t. $\text{span}(v_1, \ldots, v_k) = S$

If $v_i^T v_j = 0$ for all $i \neq j$, $v_1, \ldots, v_k$ is an orthogonal basis of S

If v_i are normalized to be unit vectors, orthonormal basis of S

e.g. $e_1, \ldots, e_m$

For S, all bases have some number k vectors, this is the dimension of S, $\dim(S) = m$

For an orthonormal basis $u_1, \dots, u_d$, a point x can be expressed

coordinates

$$x = a_1 u_1 + \dots + a_d u_d = Ua$$

where $a = \begin{pmatrix} a_1 \\ \vdots \\ a_d \end{pmatrix}$, $U = \begin{pmatrix} | & & | \\ u_1 & \dots & u_d \\ | & & | \end{pmatrix}$

Since U is orthogonal, $U^{-1} = U^T$

$$\Rightarrow U^T x = U^T U a$$

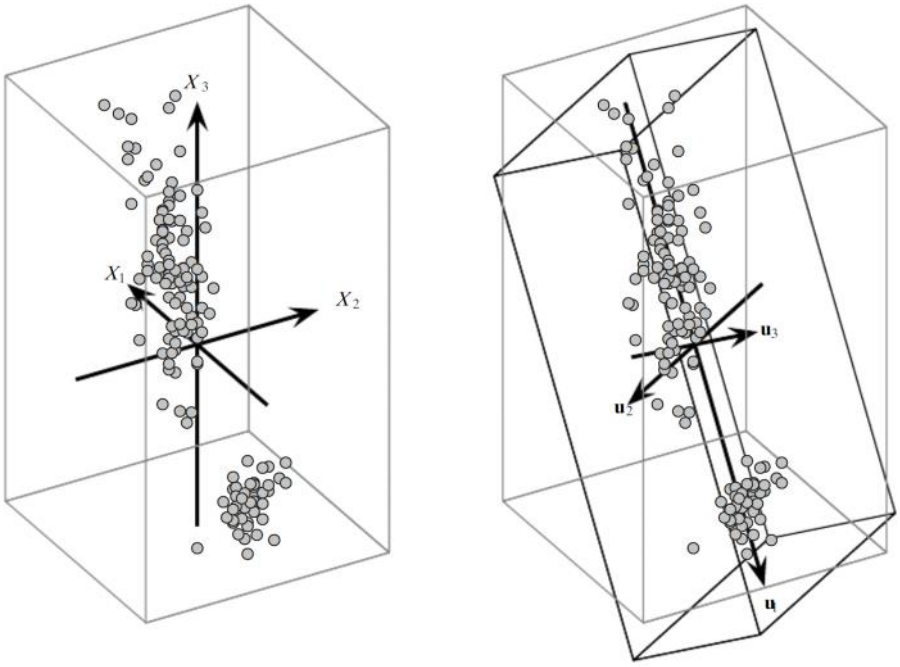
$$U^T x = a$$

← this is how we get the coordinates with respect to the new basis

original coordinates

new coordinates

Question: Is there an "optimal" basis?



(a) Original Basis (b) Optimal Basis

Question: Can we find $r << d$ s.t. there is a r-dim subspace that preserves the essential properties of the data?

↳ $x' = \sum_{i=1}^{r} a_i u_i = \begin{pmatrix} | & & | \\ u_1 & \cdots & u_r \\ | & & | \end{pmatrix} \begin{pmatrix} a_1 \\ \vdots \\ a_r \end{pmatrix} = U_r a_r$

As before, $x' = U_r a_r$

$U_r^T x' = U_r^T U_r a_r = a_r$

$U_r x$ ...

$$\Rightarrow \quad x' = U_r U_r^T x' = P_r x$$

where $P_r = U_r U_r^T$ is the orthogonal projection matrix for the subsp. spanned by the first r basis vectors

$$\Rightarrow \quad P_r^T = \left(U_r U_r^T\right)^T = U_r U_r^T = P_r$$

(reverse order & transpose)

(P_r is symmetric)

$$P_r^2 = \left(U_r U_r^T\right)\left(U_r U_r^T\right) = U_r \underbrace{\left(U_r^T U_r\right)}_{I_r} U_r^T = U_r U_r^T = P_r$$

(assoc. prop)

And, $$P_r = U_r U_r^T = \sum_{i=1}^{r} u_i u_i^T$$

$$\varepsilon = x - x' = \sum_{i=1}^{d} a_i u_i - \sum_{i=1}^{r} a_i u_i = \sum_{i=r+1}^{d} a_i u_i$$

Note... $$x'^T \varepsilon = \underbrace{\sum_{i=1}^{r} \sum_{j=r+1}^{d}}_{i \text{ is strictly less than } j} a_i a_j \cancel{u_i^T u_j}^{\,0} = 0$$

orthogonal

Goal: Find r-dim basis s.t. each x_i' approximates

Goal: Find r-dim basis s.t. each x_i [illegible]
$x_i \in D$ well, or minimize $\varepsilon = x_i - x_i'$ $\forall$ points

a measure of dispersion

Total variance of D is the average squared distance of each point (row) to μ

$$var(D) = \frac{1}{n}\sum_{i=1}^{n}\|x_i - \mu\|^2 = \frac{1}{n}\sum_{i=1}^{n}\|x_i\|^2 - \|\mu\|^2$$

Let's find the best $r=1$ dimensional subspace u (a line) in terms of maximizing variance from D.

Geometry...

https://i.stack.imgur.com/Q7HIP.gif

Assume u is a unit vector $\|u\|^2 = u^T u = 1$

Center D to $\bar{D}$ with mean $\bar{\mu} = 0 \in \mathbb{R}^d$

$$\bar{D} = D - \mathbb{1} \mu^T$$

Project $\bar{x}_i \in \bar{D}$ onto u

$$\bar{x}_i' = \left(\frac{u^T \bar{x}_i}{u^T u} \right) u = \underbrace{(u^T \bar{x}_i)}_{a_i} u = a_i u$$

a_i is scalar projection of x_i on u (projected point)

Mean of a_i...

$$\bar{\mu} = \mu - \mu = 0$$

$$\mu_a = \frac{1}{n} \sum_{i=1}^{n} a_i = \frac{1}{n} \sum_{i=1}^{n} u^T \bar{x}_i = u^T \bar{\mu} = 0$$

projected variance along u

$$\frac{1}{n} \sum_{i=1}^{n} (a_i - \mu_a)^2 = \frac{1}{n} \sum_{i=1}^{n} (u^T \bar{x}_i)^2 = \frac{1}{n} \sum_{i=1}^{n} (u^T \bar{x}_i)(u^T \bar{x}_i)^T$$

$$\sigma_u^2 = \frac{1}{n}\sum_{i=1}^{n}\left(a_i - \cancel{\mu_a}^{0}\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(u^T\bar{x}_i\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(u^T\bar{x}_i\right)\left(u^T\bar{x}_i\right)^T$$

($a_i = u^T\bar{x}_i$)

$$= \frac{1}{n}\sum_{i=1}^{n} u^T\bar{x}_i\bar{x}_i^T u = u^T\left(\frac{1}{n}\sum_{i=1}^{n}\bar{x}_i\bar{x}_i^T\right)u = u^T\Sigma u$$

(Σ: covariance matrix for D)

Goal: choose u to maximize σ_u subject to $u^Tu = 1$. ($u^Tu - 1 = 0$)

$$\underset{u}{\text{Maximize}}\; u^T\Sigma u - \alpha(u^Tu - 1)$$

(α: Lagrange multiplier)

Unconstrained maximization problem

We need to set the deriv. w.r.t. u to 0 and solve

$$\frac{\partial}{\partial u}\left(u^T\Sigma u - \alpha(u^Tu - 1)\right) = 0$$

$$2\Sigma u - 2\alpha u = 0$$

$$\Sigma u = \alpha u$$

(α: eigenvalue!)

$$u^T\Sigma u = u^T\alpha u = \alpha \quad (\text{since } u^Tu = 1)$$

We need an eigenvalue α that is largest.

($\lambda_1 \geq \lambda_2 \geq \ldots$)

1 with eigenvector $u = u_1$

We ... $\lambda_1 \geq \lambda_2 \geq \dots$

$\Rightarrow \sigma_u^2 = a = \lambda$ with eigenvector $u = u_1$

the first principal component

Similarly,

$$MSE(u) = \frac{1}{n}\sum_{i=1}^{n}\|\varepsilon_i\|^2 = \dots = \frac{1}{n}\sum_{i=1}^{n}\|\bar{x}_i\|^2 - u^T\Sigma u \quad \text{(2.35)}$$

$$= \text{Var}(D) - u^T\Sigma u = \text{Var}(D) - u^T\Sigma u = \sum_{i=1}^{n}\sigma_i^2 - u^T\Sigma u$$

adding a constant does not affect var

Constant $\Rightarrow$ minimizing $MSE(u)$ $\Leftrightarrow$ maximizing $u^T\Sigma u$

we use this approach in multidim. PCA below...

Through a similar approach, we find for

$$\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_r \geq \lambda_{r+1} \geq \dots \geq \lambda_d \geq 0$$

u_1 → first PC, u_2 → second PC, u_r → rth PC

$$U_r = \begin{pmatrix} | & & | \\ u_1 & \cdots & u_r \\ | & & | \end{pmatrix} = \text{basis}$$

take these as the basis of r-dim space

Total projected variance

Total projected variance

$$A = \{U_r^T \bar{x}_i : \bar{x}_i \in \bar{D}\} \text{ - new dataset}$$

$$var(A) = \frac{1}{n}\sum_{i=1}^{n} \|a_i - 0\|^2 = \frac{1}{n}\sum_{i=1}^{n} (U_r^T \bar{x}_i)^T (U_r^T \bar{x}_i)$$

$$= \frac{1}{n}\sum_{i=1}^{n} \bar{x}_i^T U_r U_r^T \bar{x}_i = \frac{1}{n}\sum_{i=1}^{n} \bar{x}_i^T P_r \bar{x}_i$$

$$= \frac{1}{n}\sum_{i=1}^{n} \bar{x}_i^T \left(\sum_{j=1}^{r} u_j u_j^T\right) \bar{x}_i = \sum_{j=1}^{r} u_j^T \left(\frac{1}{n}\sum_{i=1}^{n} \bar{x}_i^T \bar{x}_i\right) u_j$$

$$= \sum_{j=1}^{r} u_j^T \Sigma u_j = \sum_{j=1}^{r} \lambda_j$$

MSE $\frac{1}{n}\sum_{i=1}^{n} \|\bar{x}_i - \bar{x}_i'\|^2 = Var(D) - Var(A) = var(D) - \sum_{j=1}^{r} \lambda_j$

Total var $var(D) = \sum_{i=1}^{d} \sigma_i^2 = \sum_{i=1}^{d} \lambda_i$

invariant to a change of basis

Choosing r

$$f(r) = \frac{var(A)}{var(D)} = \frac{\lambda_1 + \ldots + \lambda_r}{\lambda_1 + \ldots + \lambda_r + \ldots + \lambda_d}$$

fraction of variance captured by the lower-dimension data

Choose $r = \min\{r' \mid f(r') \geq \alpha\}$

typical: $\alpha = 0.9$ or higher

The Geometry of PCA

https://i.stack.imgur.com/Q7HIP.gif

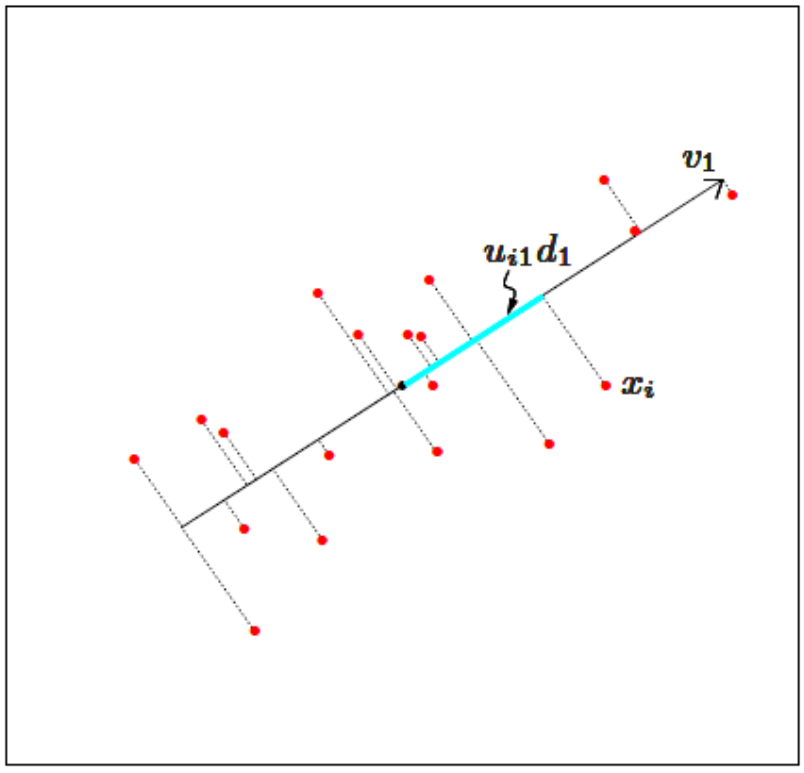


FIGURE 14.20. *The first linear principal component of a set of data. The line minimizes the total squared distance from each point to its orthogonal projection onto the line.*

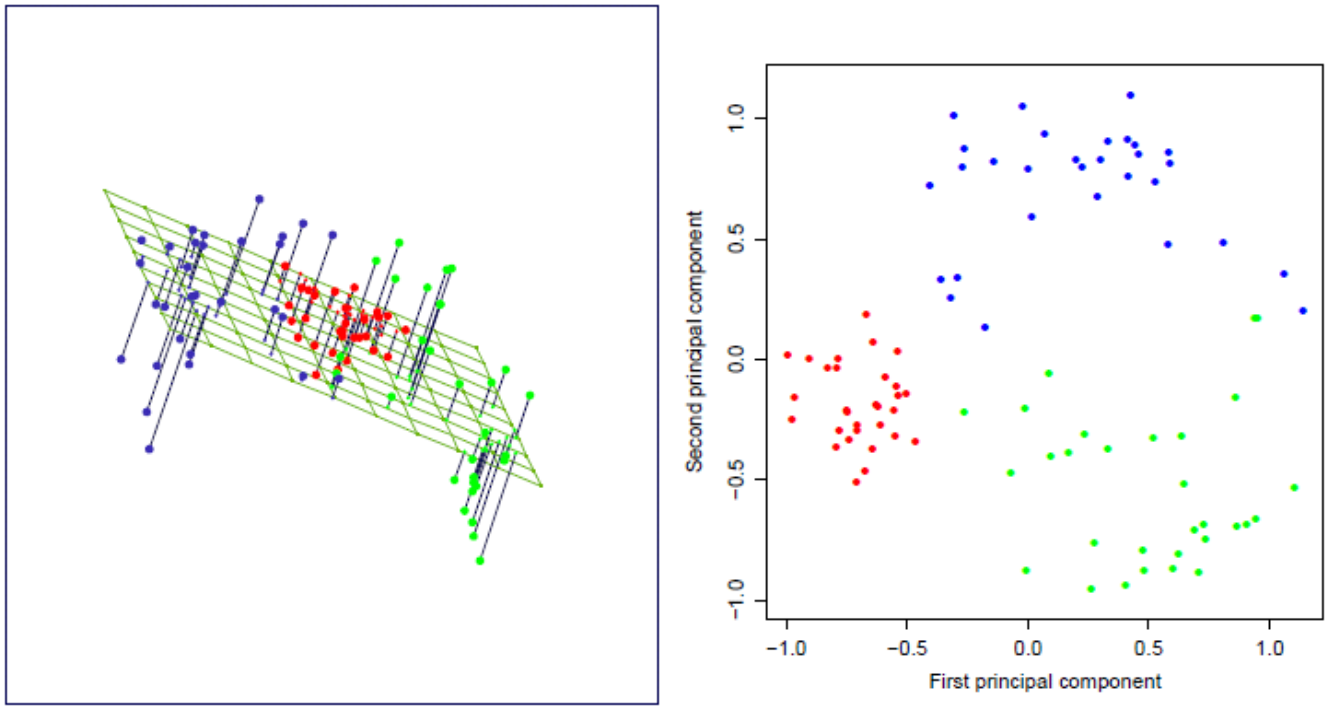
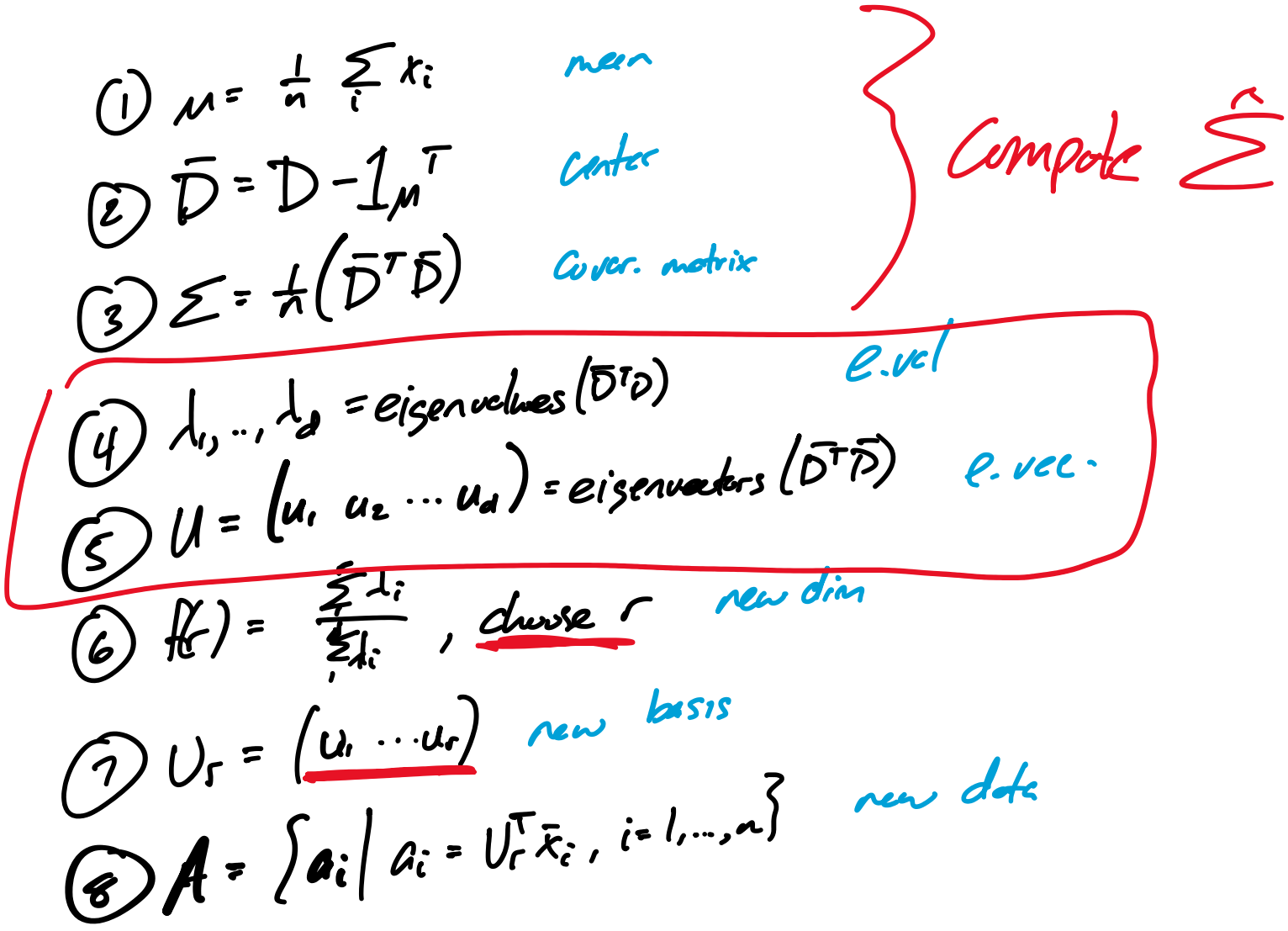


FIGURE 14.21. *The best rank-two linear approximation to the half-sphere data. The right panel shows the projected points with coordinates given by* $\mathbf{U}_2\mathbf{D}_2$*, the first two principal components of the data.*

Algorithm




```python
# Principal component analysis
def PCA(X, alpha):
    # find the original dimension and print it
    dimension = X.shape[1]
    print('The original dimension of the data is', dimension)

    # center D to have mean 0
    X -= np.mean(X, axis=0)

    # compute the covariance matrix
    Sigma = (1/X.shape[0]) * X.T @ X

    # compute the eigenvalues and eigenvectors of D^T D
    (eValues, eVectors) = np.linalg.eigh(Sigma)

    # compute the total variance
    varX = np.sum(eValues)

    # initialize the variance for A to 0
    varA = 0.0

    # reverse eValues and eVectors
    eValues = np.flip(eValues)
    eVectors = np.flip(eVectors, axis=1)
```

```
    # find the minimum dimension consisting of fraction at least alpha of the total variance
    for r in np.arange(0, eValues.shape[0]):
        varA += eValues[r]
        ratio = varA/varX

        if ratio > alpha:
            dimension = r + 1
            print('The new dimension of the data is', dimension, 'and it explains', ratio, 'of the variance')
            break

    # construct the new basis
    basis = eVectors[:,:dimension]

    # create the datapoints in the new space
    A = X @ basis

    # return points
    return A
```

Example: Image Compression

Original (512-by-512 image)

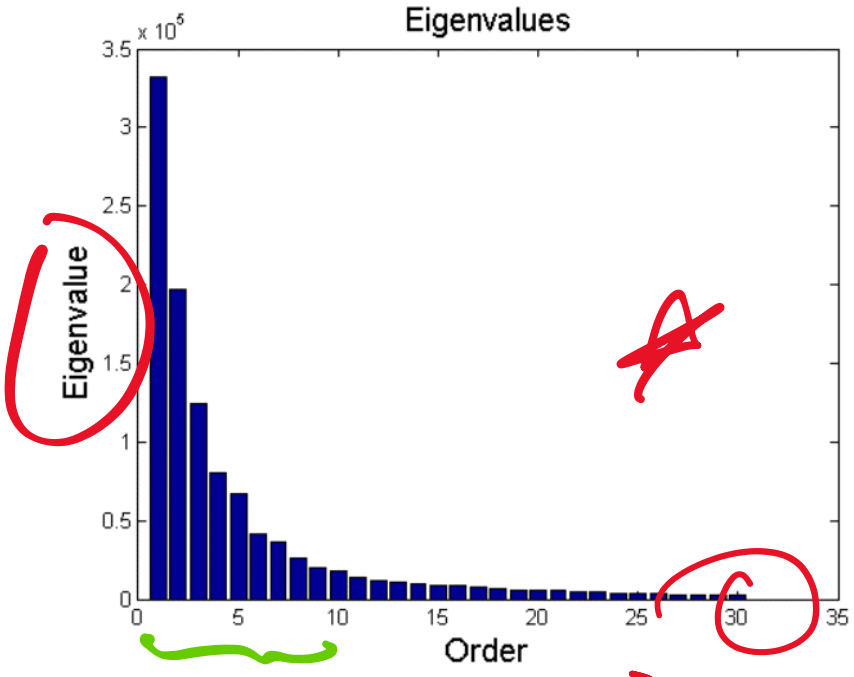
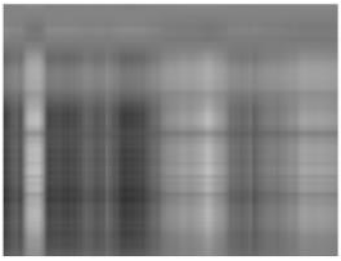


Figure 6. Ordered eigenvalues.



(a) 1 principal component

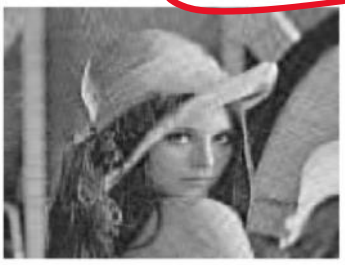
(b) 5 principal component

(c) 9 principal component

(d) 13 principal component

(e) 17 principal component

(f) 21 principal component



(g) 25 principal component

(h) 29 principal component

Example

```
# import MNIST data
(trainX, trainY), (testX, testY) = mnist.load_data()

# reshape the data
trainX = trainX.reshape(trainX.shape[0], trainX.shape[1] * trainX.shape[2])
testX = testX.reshape(testX.shape[0], testX.shape[1] * testX.shape[2])

# normalize coordinates
trainX = trainX.astype('float')/255.0
testX = testX.astype('float')/255.0

# fit the model to the training data
print('Now training QDA...')

start = time.time()

model = QuadraticDiscriminantAnalysis(reg_param = 0.05)
model.fit(trainX, trainY)

end = time.time()

print('QDA training is finished after', end - start, 'seconds')

# predict the labels of the test set
print('Now testing QDA...')

start = time.time()

predictedY = model.predict(testX)

end = time.time()

print('QDA prediction is finished after', end - start, 'seconds')
```

```
# print quality metrics
print('\nTest Classification Report for reg_param =', 0.05, classification_report(testY, predictedY))

print('\nTest Confusion Matrix:\n')
sn.heatmap(confusion_matrix(testY, predictedY))
```

```
# import MNIST data
(trainX, trainY), (testX, testY) = mnist.load_data()

X = np.vstack((trainX, testX))
Y = np.concatenate((trainY, testY))

# reshape the data
X = X.reshape(X.shape[0], X.shape[1] * X.shape[2]).astype('float')

# Use PCA
print('Now using PCA dimensionality reduction...')

start = time.time()

# Apply PCA to the data matrices
X = PCA(X, 0.8)

end = time.time()

print('PCA finished in', end - start, 'seconds')

# normalize the data
X = X / 255.0

# train-test split
trainX, testX, trainY, testY = train_test_split(X, Y, test_size = 0.25, random_state = 1)

# fit the model to the training data
print('Now training QDA...')

start = time.time()
model = QuadraticDiscriminantAnalysis(reg_param = 0.05)
model.fit(trainX, trainY)

end = time.time()

print('QDA training is finished after', end - start, 'seconds')

# predict the labels of the test set
print('Now testing QDA...')

start = time.time()

predictedY = model.predict(testX)

end = time.time()

print('QDA prediction is finished after', end - start, 'seconds')

# print quality metrics
print('\nTest Classification Report for reg_param =', 0.05, classification_report(testY, predictedY))

print('\nTest Confusion Matrix:\n')
sn.heatmap(confusion_matrix(testY, predictedY))
```

Principal Components Analysis

- Invented by Karl Pearson as an analogue of the principal axis theorem
- Independently developed in several fields:
  - Karhunen-Loeve transform (KLT) in signal processing
  - Hotelling transform in quality control
  - Proper orthogonal decomposition (POD) in mechanical engineering
  - Singular value decomposition (SVD) of X
  - Eigenvalue decomposition of X^TX
  - Factor analysis
  - Eckart-Young theorem
  - Empirical orthogonal functions (EOF) in meteorology
  - Spectral decomposition in noise and vibration
  - Empirical modal analysis in structural dynamics

Geometry…

https://i.stack.imgur.com/Q7HIP.gif

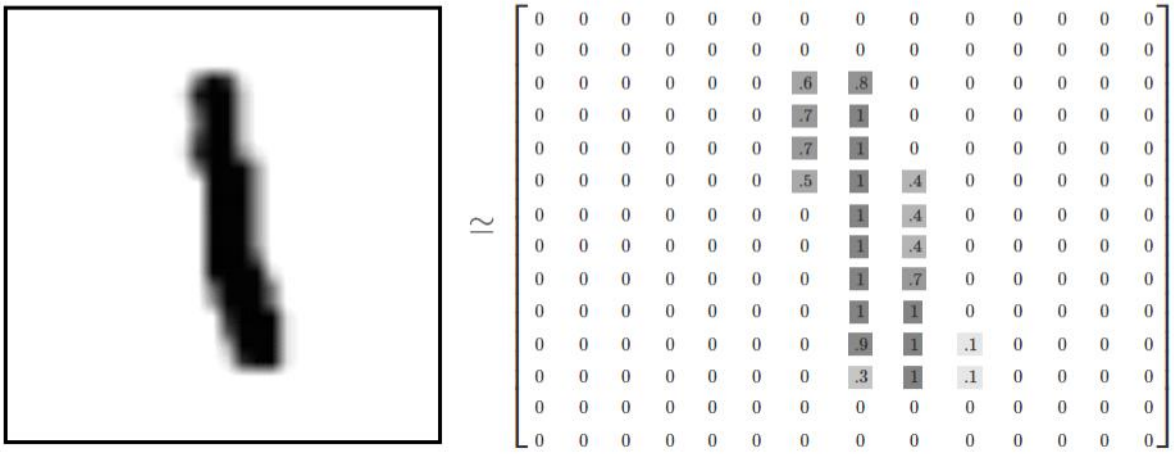
Visualizing MNIST

Most of the content here is from chris Olah's amazing blog post:

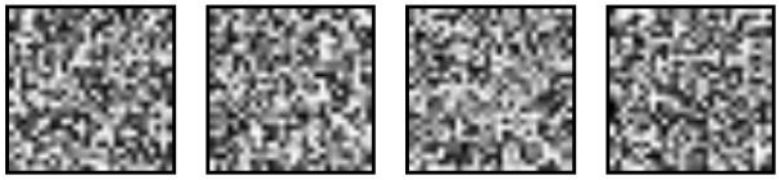
[Visualizing MNIST: An Exploration in Dimensionality Reduction]

MNIST is a simple computer vision dataset. It consists of 28x28 pixel images of handwritten digits, such as:

Every MNIST data point, every image, can be thought of as an array of numbers describing how dark each pixel is. For example, we might think of \ as something like:

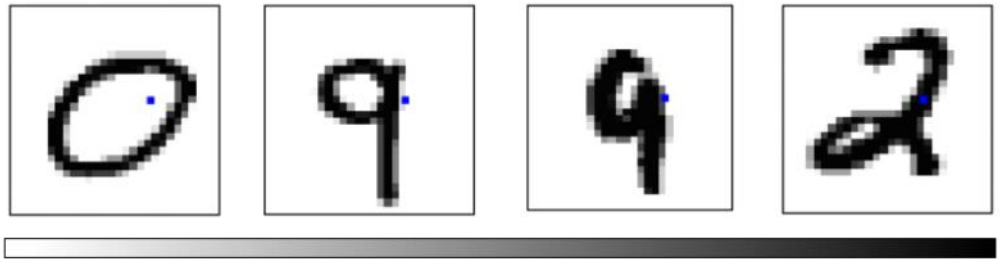


Not all vectors in this 784-dimensional space are MNIST digits. Typical points in this space are very different! To get a sense of what a typical point looks like, we can randomly pick a few points and examine them. In a random point – a random 28x28 image – each pixel is randomly black, white or some shade of gray. The result is that random points look like noise.



Images like MNIST digits are very rare. While the MNIST data points are *embedded* in 784-dimensional space, they live in a very small subspace. With some slightly harder arguments, we can see that they occupy a lower dimensional subspace.

We can think of the MNIST data points as points suspended in a 784-dimensional cube. Each dimension of the cube corresponds to a particular pixel. The data points range from zero to one according to the pixels intensity. On one side of the dimension, there are images where that pixel is white. On the other side of the dimension, there are images where it is black. In between, there are images where it is gray.



If we think of it this way, a natural question occurs. What does the cube look like if we look at a particular two-dimensional face? Like staring into a snow-globe, we see the data points projected into two dimensions, with one dimension corresponding to the intensity of a particular pixel, and the other corresponding to the intensity of a second pixel. Examining this allows us to explore MNIST in a very raw way.

(See the Olah blog viz)

Exploring this visualization, we can see some glimpses of the structure of MNIST. Looking at the pixels $p_{18,16}$ and $p_{7,12}$, we are able to separate a lot of zeros to the bottom right and a lot of nines to the top left. Looking at pixels $p_{5,6}$ and $p_{7,9}$ we can see a lot of twos at the top right and threes at the bottom right.

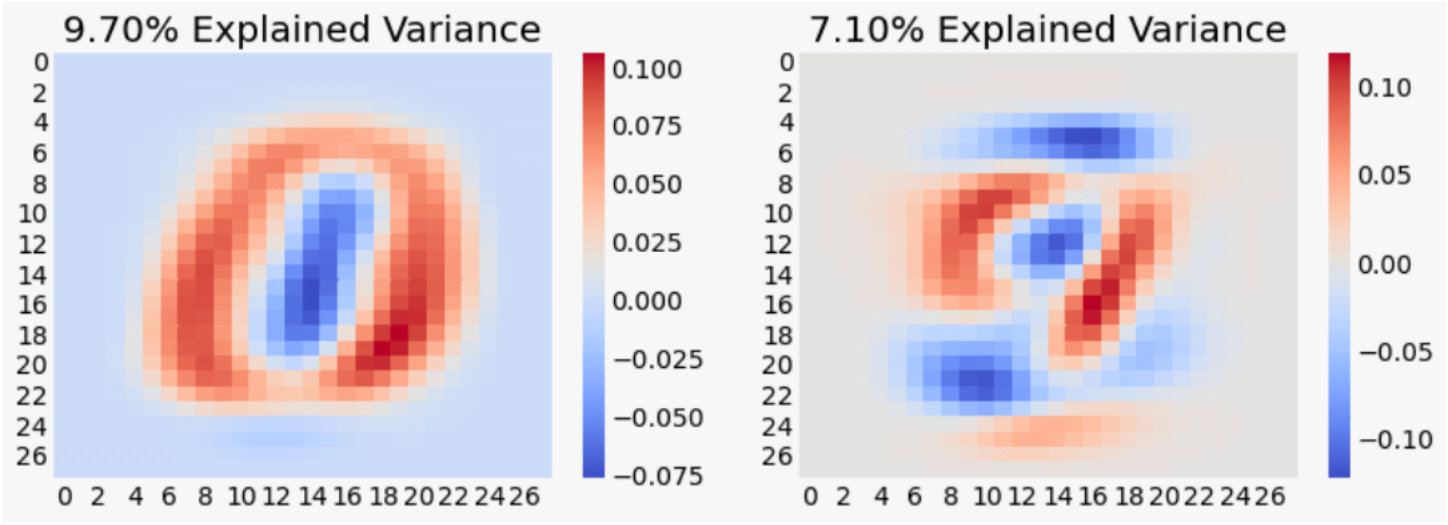
Despite minor successes like these, one can't really can't understand MNIST this way. The small insights one gains feel very fragile and feel a lot like luck. The truth is, simply, that very little of MNIST's structure is visible from these perspectives. You can't understand images by looking at just two pixels at a time.

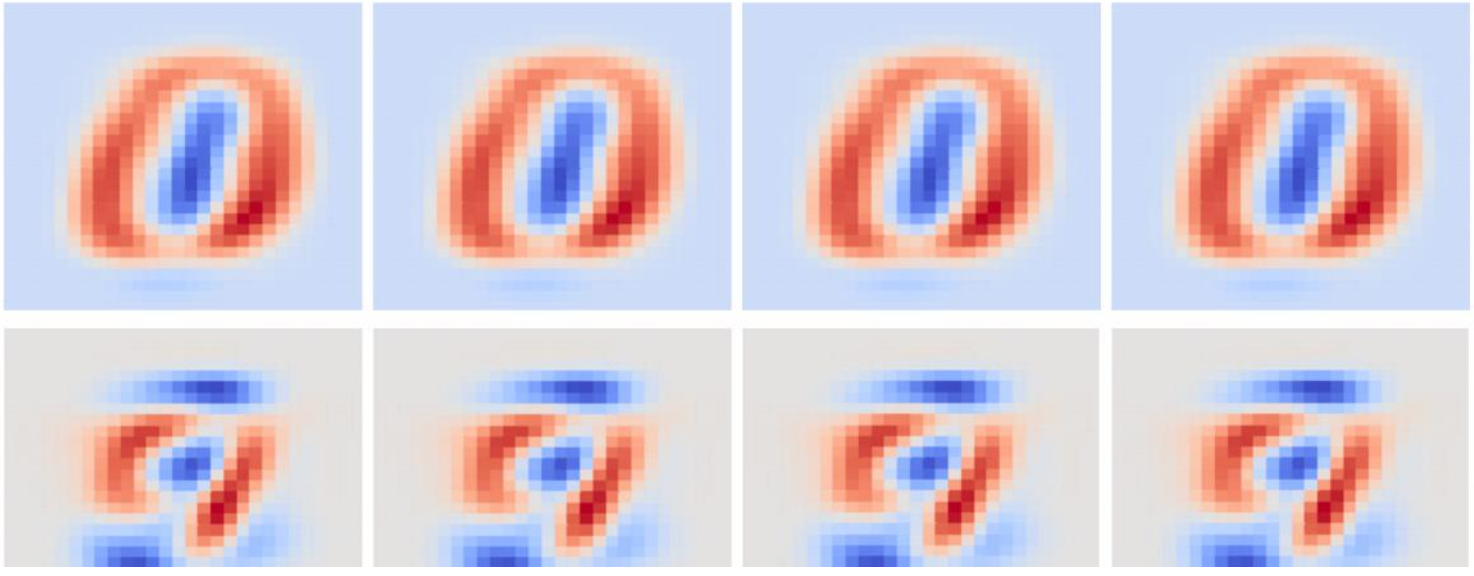
But there's lots of other perspectives we could look at MNIST from! In these perspectives, instead of looking a face straight on, one looks at it from an angle.

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

from sklearn.decomposition import PCA

from tensorflow.keras.datasets import mnist
```

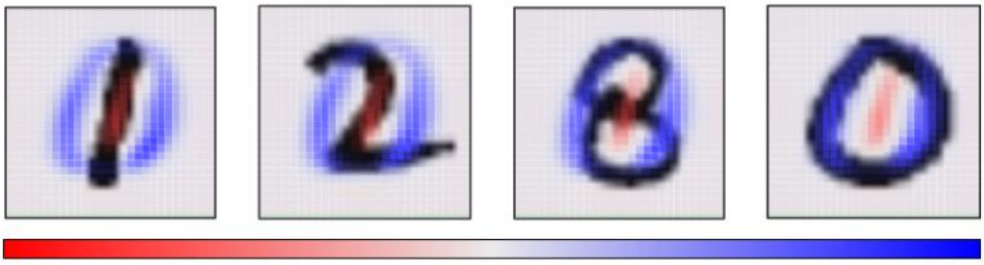
```
# read the MNIST data
(X, Y), _ = mnist.load_data()

# reshape the datapoints into rows
X = X.reshape(X.shape[0], X.shape[1] * X.shape[2])

# create a PCA model with two components
pca = PCA(n_components = 2)

# transform the data
A = pca.fit_transform(X)
```

```
# set the style
plt.style.use('fivethirtyeight')

# declare a 1-by-2 plot
fig, axarr = plt.subplots(1, 2, figsize=(12, 4))

# plot heatmaps of PCA components and shape into a square
sns.heatmap(pca.components_[0, :].reshape(28, 28), ax=axarr[0], cmap='coolwarm')
sns.heatmap(pca.components_[1, :].reshape(28, 28), ax=axarr[1], cmap='coolwarm')

# set chart titles
axarr[0].set_title("{0:.2f}% Explained Variance".format(pca.explained_variance_ratio_[0]*100))
axarr[1].set_title("{0:.2f}% Explained Variance".format(pca.explained_variance_ratio_[1]*100))
```





If an MNIST digit primarily highlights red, it ends up on one side. If it highlights blue, it ends up on a different side. The first angle – the "first principal component" – will be our horizontal angle, pushing ones (which highlight lots of red and little blue) to the left and zeros (which highlight lots of blue and little red) to the right.



Now that we know what the best horizontal and vertical angle are, we can try to look at the cube from that perspective.

Threes with PCA

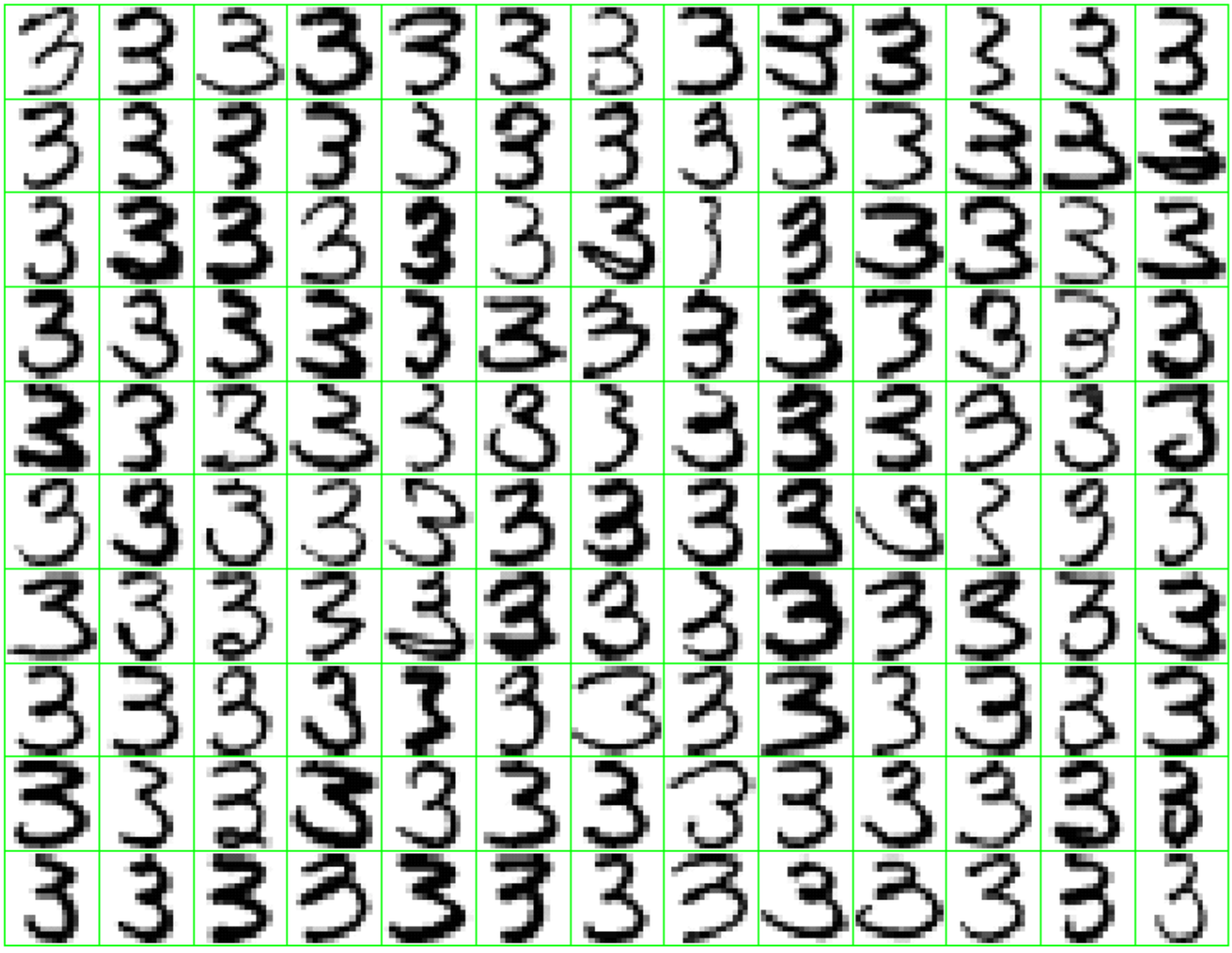
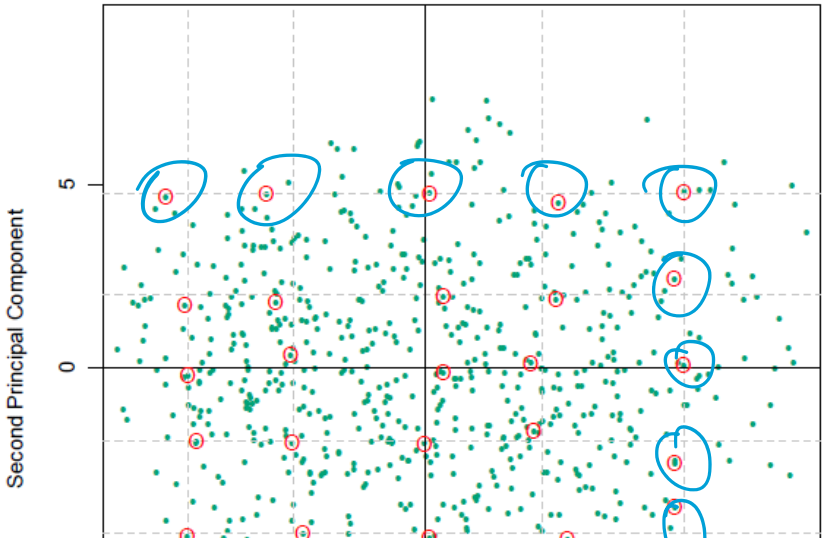
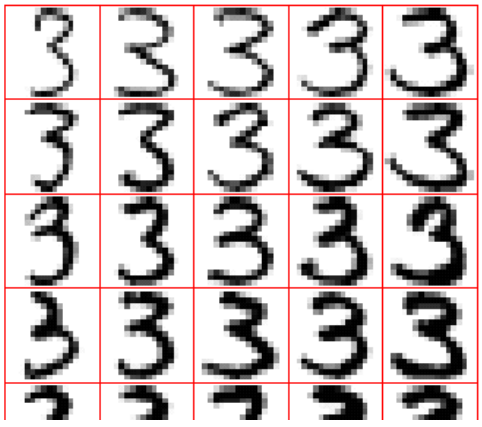


FIGURE 14.22. *A sample of* 130 *handwritten 3's shows a variety of writing styles.*





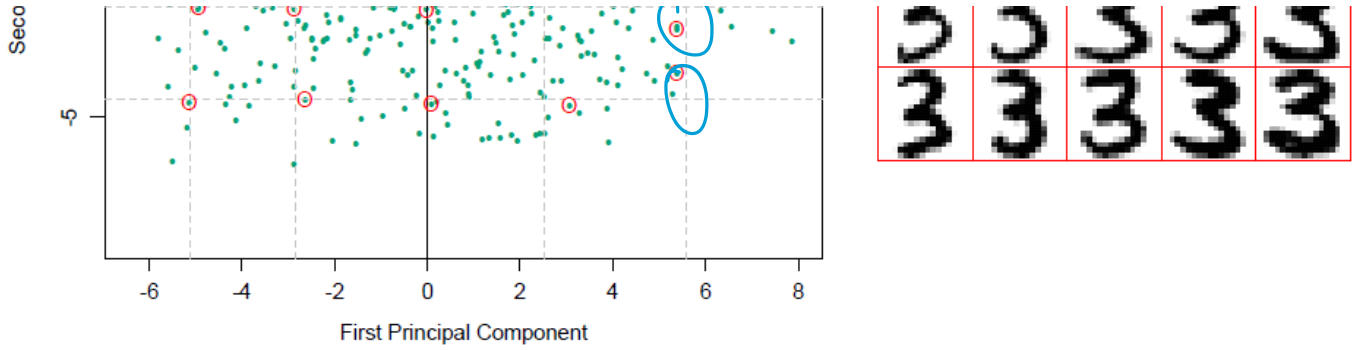
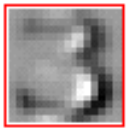


FIGURE 14.23. *(Left panel:) the first two principal components of the hand-written threes. The circled points are the closest projected images to the vertices of a grid, defined by the marginal quantiles of the principal components. (Right panel:) The images corresponding to the circled points. These show the nature of the first two principal components.*

Figure 14.23 shows the first two principal components of these data. For each of these first two principal components u_{i1} and u_{i2}, we computed the 5%, 25%, 50%, 75% and 95% quantile points, and used them to define the rectangular grid superimposed on the plot. The circled points indicate those images close to the vertices of the grid, where the distance measure focuses mainly on these projected coordinates, but gives some weight to the components in the orthogonal subspace. The right plot shows the images corresponding to these circled points. This allows us to visualize the nature of the first two principal components. We see that the v_1 (horizontal movement) mainly accounts for the lengthening of the lower tail of the three, while v_2 (vertical movement) accounts for character thickness. In terms of the parametrized model (14.49), this two-component model has the form

$$\begin{aligned} \hat{f}(\lambda) &= \bar{x} + \lambda_1 v_1 + \lambda_2 v_2 \\ &= \text{  } + \lambda_1 \cdot \text{  } + \lambda_2 \cdot \text{  }. \end{aligned}$$

Avg. 3 PC 1 PC 2