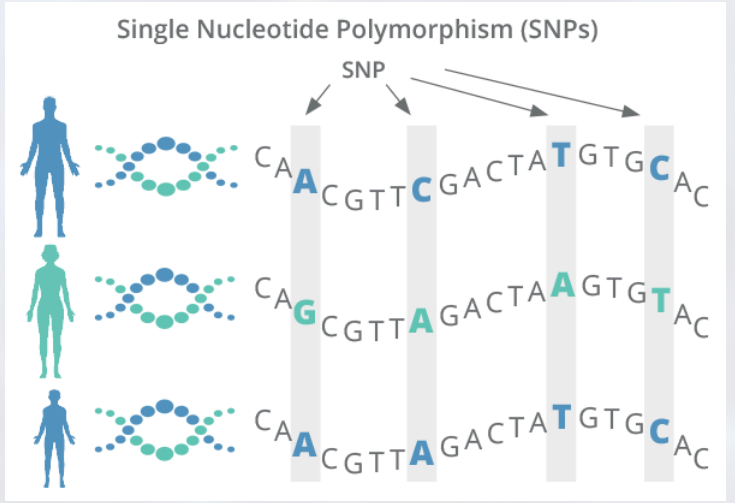
Linear Algebra Application: Linear Dimensionality Reduction via PCA

David I. Inouye

Unsupervised Dimensionality Reduction via PCA

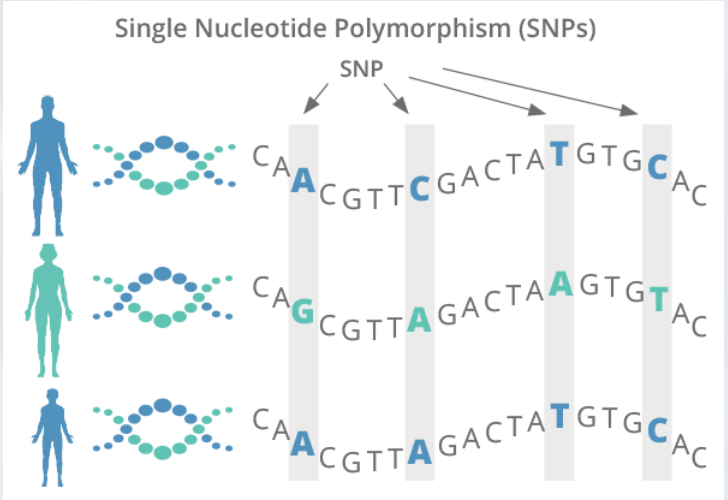
Very High-Dimensional Data is Becoming Ubiquitous

- Images (1 million pixels)
- Text (100k unique words)
- Genetics (4 million SNPs)
- Business data (12 million products)



Why Dimensionality Reduction? Lower Computation Costs

- Suppose original dimension is large like $d = 100000$ (e.g., images, DNA sequencing, or text).
- If we reduce to $k = 100$ dimensions, the training algorithm can be sped up by $1000\times$.



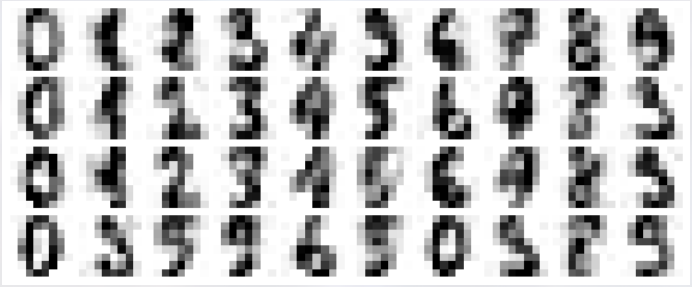
4-5 million SNPs in human genome. https://www.diagnosticsolutionslab.com/tests/genomicinsight

Why Dimensionality Reduction? Visualization

- Allows 2D scatterplot visualizations even of high-dimensional data (2D projection of digits).

https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

Why Dimensionality Reduction? Noise Reduction Via Reconstruction



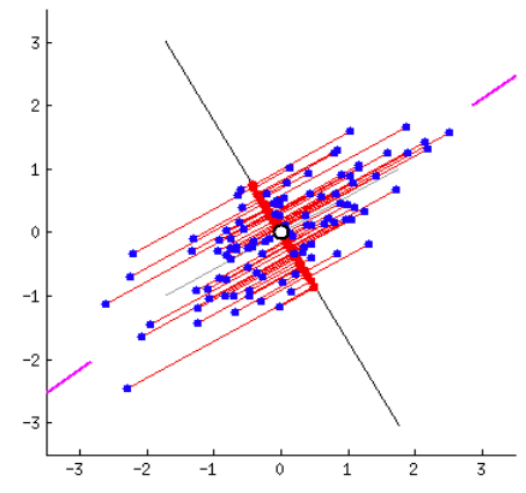
Outline of Principal Components Analysis (PCA)

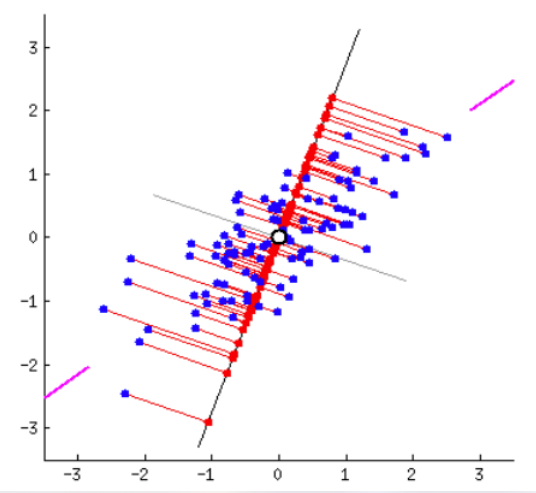
- Motivation for dimensionality reduction
- Formal PCA problem: Min reconstruction
- Derive PCA formulation for 1D
  - Least error 1D projection is orthogonal
  - Sum over all data points
- Solution is based on truncated SVD
- Alternative problem: Max variance

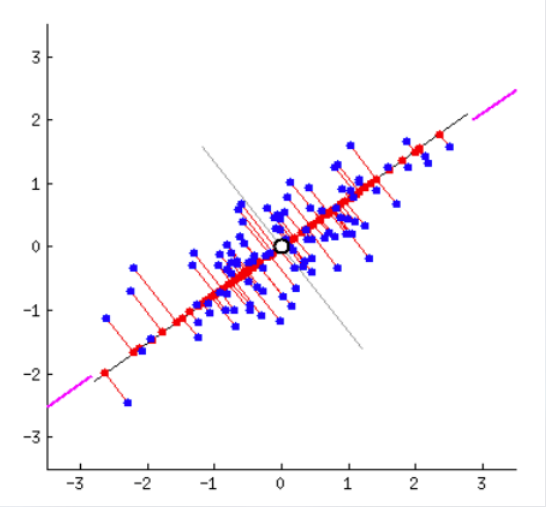
PCA Intuition and Math

The Goal: Finding the Best Lower-Dimensional Representation

Before diving into equations, let's start with the core visual intuition. What does it mean to find the "best" lower-dimensional projection of our data?

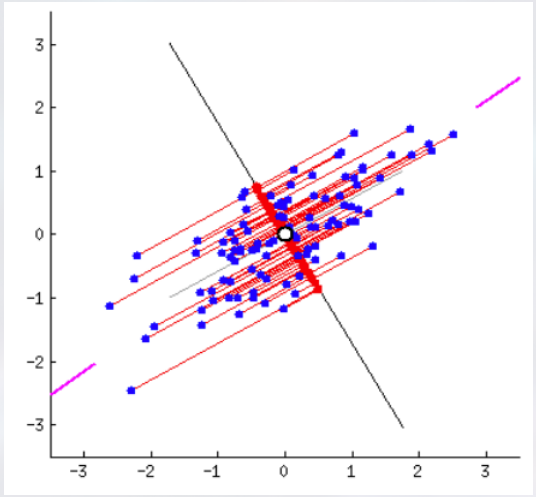




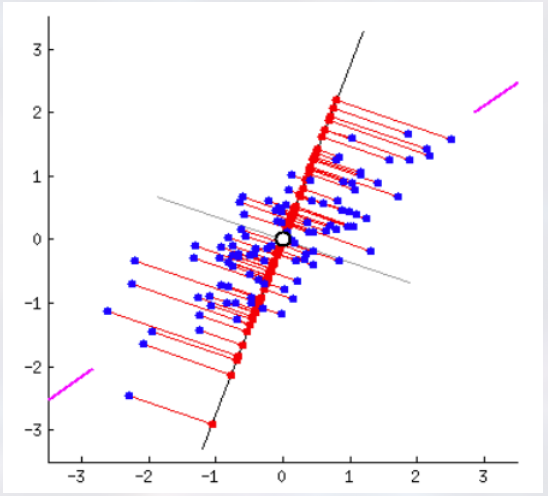


Which is the best projection of the 2D data onto a 1D space (i.e., a line)?

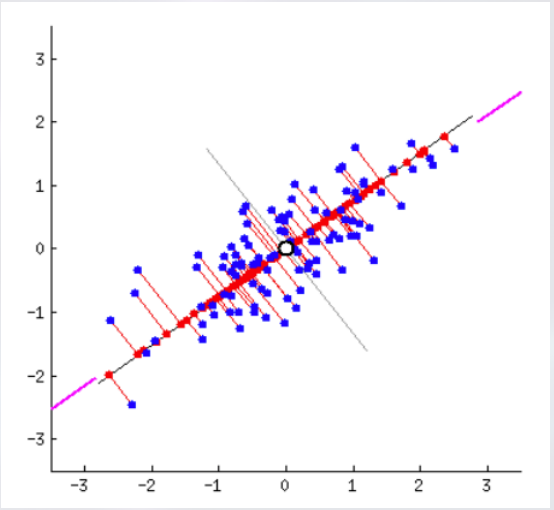
Intuition: Principal Component Analysis Finds The Best Linear Projection Onto a Lower-Dimensional Space



Poor reconstruction



Okay reconstruction



Best reconstruction

- PCA finds the line that has the smallest projection error.

https://stats.stackexchange.com/questions/2691/making-sense-of-principal-component-analysis-eigenvectors-eigenvalues

Viewpoint 1: Minimizing Reconstruction Error

Now, let's formalize this idea of "smallest projection error." This is the first of two ways we will define PCA.

Math: PCA Minimizes The Linear Reconstruction Error Using Only $k \leq d$ Dimensions

PCA can be formalized as:

$$\min_{Z,W} \|X_c - ZW^T\|_F^2$$

where

- $X_c = X - \mathbf{1}_n \mu_x^T \in \mathbb{R}^{n \times d}$ (centered input data)
- $Z \in \mathbb{R}^{n \times k}$ (latent representation or "scores")
- $W^T \in \mathbb{R}^{k \times d}$ (principal components)
- $w_s^T w_t = 0, w_s^T w_s = \|w_s\|_2^2 = 1, \forall s, t \neq s$ (orthogonal constraint)
  - Can be written more compactly as simply: $W^T W = I_k$.

Math: PCA Minimizes The Linear Reconstruction Error Using Only $k \leq d$ Dimensions

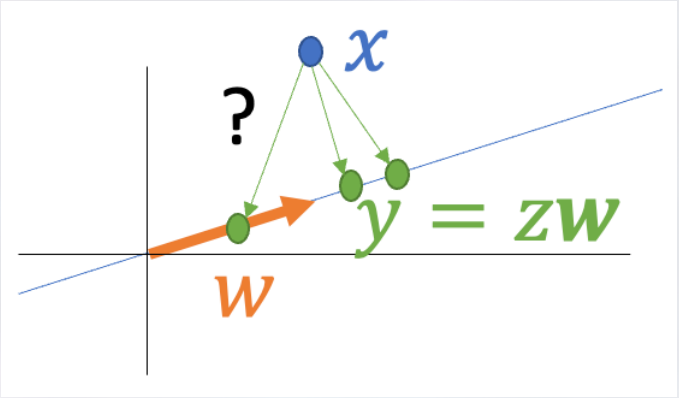
$$\min_{Z \in \mathbb{R}^{n \times k}, W \in \mathbb{R}^{d \times k}} \|X_c - ZW^T\|_F^2 \quad \text{s.t.} \quad W^T W = I_k$$

- Let's stare at this equation some more 🙂
- Why is this dimensionality reduction?
- What does the orthogonal constraint mean?
- Why minimize the squared Frobenius norm?
- $\|X_c - ZW^T\|_F^2 = \sum_{i=1}^n \|x_i^T - z_i^T W^T\|_2^2 = \sum_{i=1}^n \|x_i - W z_i\|_2^2$
- For analysis, let's simplify to a single dimension (i.e., $k = 1$):
  - $\sum_{i=1}^n \|x_i - z_i w\|_2^2 \quad \text{where } z_i \text{ is a scalar}$

What is The Best Projection Given a Fixed Subspace (Line in 1D Case)?

- If we are given w, what is the best z (i.e. minimum reconstruction error) for a given x?

$$\min_z \|x - zw\|_2^2$$



- The orthogonal projection!
  - $z = x^T w = \|x\|\|w\| \cos\theta = \|x\| \cos\theta$
  - $z = \|x\| \cos\theta = \text{hyp} \cdot \frac{\text{adj}}{\text{hyp}} = \text{adj}$
- zw is a scaled vector along the line defined by w.

Using this solution for z, we can simplify to only minimizing over w

$$\min_{z,w:\|w\|_2=1} \sum_{i=1}^{n} \|x_i - z_i w\|_2^2 = \min_{w:\|w\|_2=1} \sum_{i=1}^{n} \|x_i - (x_i^T w)w\|_2^2$$

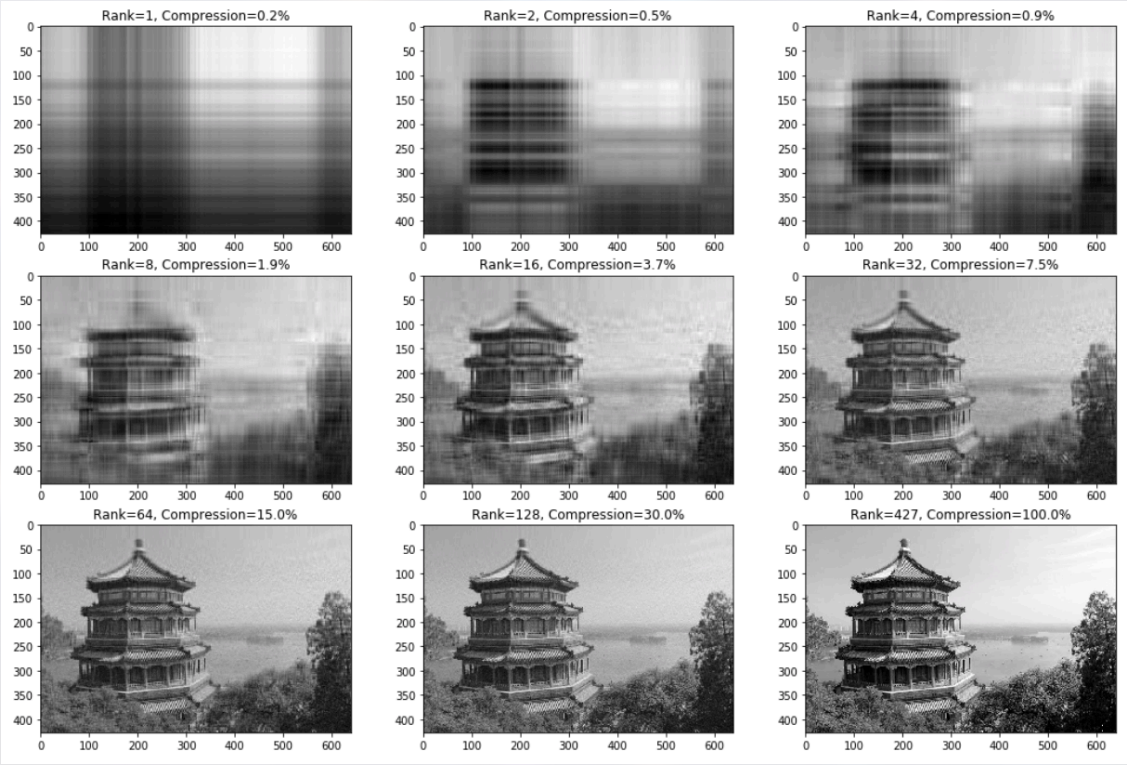
Now we can return to the Frobenius norm:

$$\min_{w:\|w\|_2=1} \|X_c - zw^T\|_F^2 \quad \text{where} \quad z = X_c w$$

- What is zw^T? Have we seen something like this before?
- This is the best low-rank approximation to X_c, which is given by the SVD!
- $w = v_1$ and $z = \sigma_1 u_1$, where σ_1, u_1, v_1 are the first singular value, left singular vector and right singular vector respectively.

For $k \geq 1$, The PCA Solution is The Top k Right Singular Vectors

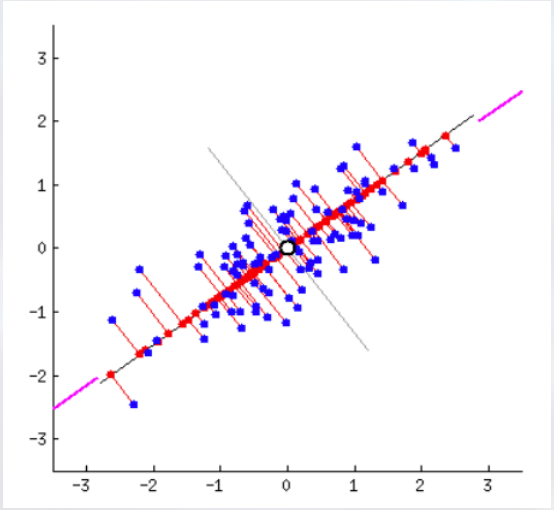
- If $X_c = USV^T$, then the general solution is $W^* = V_{1:k}$.
- Remember: SVD is best k dim. approximation



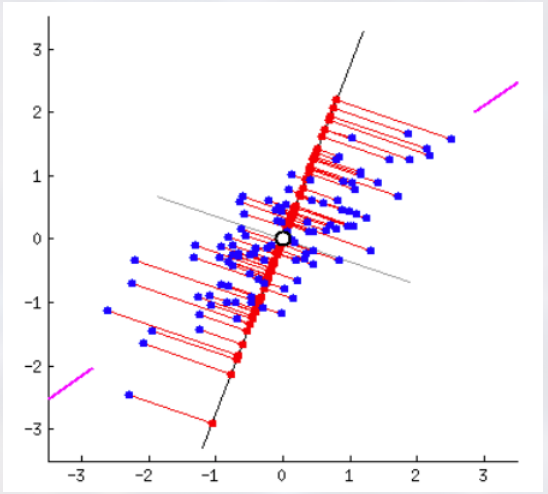
Viewpoint 2: Maximizing Sum of Squares of Projected Data

It turns out there's a completely different way to think about the "best" projection that leads to the same answer. Instead of minimizing error, we can try to maximize the information, or sum of squares, of the projected data.

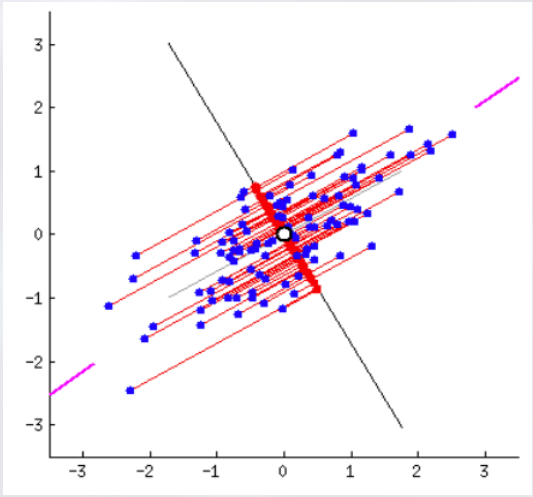
Minimizing Reconstruction Error (Red Lines) is Equivalent to Maximizing The Sum of Squares of the Projection (Spread of Red Points)



Min reconstruction error, Max variance



Middle error and variance



Max reconstruction error, Min variance

Maximizing variance problem formulation and intepretation

Let X_c be our centered data. We project it using orthonormal weights W to get the lower-dimensional projected data Z:

$$Z = X_c W$$

We measure the "amount of signal" in Z using the squared Frobenius norm (Sum of Squares):

$$\|Z\|_F^2 = \|X_c W\|_F^2$$

Goal: We want to preserve as much signal (i.e., sum of squares) as possible while satisfying orthogonality constraints:

$$\max_{W : W^T W = I} \|X_c W\|_F^2 \equiv \max_{W : W^T W = I} \|Z\|_F^2$$

Goal of Equivalence Derivation

We want to prove that minimizing the **Reconstruction Error**:

$$\min_{W:W^TW=I} \|X_c - (X_cW)W^T\|_F^2$$

Is mathematically equivalent to maximizing the **Projected Sum of Squares**:

$$\max_{W:W^TW=I} \|X_cW\|_F^2$$

Prerequisites

To derive this, we need three properties:

1. **Frobenius Norm as Trace**

$$\|A\|_F^2 = \mathrm{Tr}(A^T A)$$

2. **Cyclic Property of Trace**

$$\mathrm{Tr}(ABC) = \mathrm{Tr}(BCA)$$

3. **Orthonormality Constraint**

$$W^T W = I$$

Step 1: Expand Minimization Objective

Start with the reconstruction error $J(W)$:

$$\begin{aligned} J(W) &= \|X_c - X_c W W^T\|_F^2 \\ &= \mathrm{Tr}\left((X_c - X_c W W^T)^T (X_c - X_c W W^T)\right) \end{aligned}$$

Distribute the transpose $(AB)^T = B^T A^T$:

$$J(W) = \mathrm{Tr}\left((X_c^T - W W^T X_c^T)(X_c - X_c W W^T)\right)$$

Step 2: Multiply Terms

Expand the multiplication inside the trace:

$$
\begin{aligned}
J(W) &= \operatorname{Tr}\left((X_c^T - WW^T X_c^T)(X_c - X_c WW^T)\right) \\
&= \operatorname{Tr}(\underbrace{X_c^T X_c}_{1} - \underbrace{X_c^T X_c WW^T}_{2} \\
&\quad \underbrace{- WW^T X_c^T X_c}_{3} + \underbrace{WW^T X_c^T X_c WW^T}_{4})
\end{aligned}
$$

Separate into four traces:

$$
\begin{aligned}
J(W) = \; & \operatorname{Tr}(X_c^T X_c) - \operatorname{Tr}(X_c^T X_c WW^T) \\
& - \operatorname{Tr}(WW^T X_c^T X_c) + \operatorname{Tr}(WW^T X_c^T X_c WW^T)
\end{aligned}
$$

Step 3: Simplify Traces

Apply Cyclic Property & Constraint ($W^TW = I$):

- **Term 2:** $\mathrm{Tr}(X_c^T X_c W W^T) \to \mathrm{Tr}(W^T X_c^T X_c W)$
- **Term 3:** $\mathrm{Tr}(W W^T X_c^T X_c) \to \mathrm{Tr}(W^T X_c^T X_c W)$
- **Term 4:** $\mathrm{Tr}(W W^T X_c^T X_c W W^T)$
  - Cycle W^T to end $\to \mathrm{Tr}(W^T W W^T X_c^T X_c W)$
  - Cancel $W^T W = I \to \mathrm{Tr}(W^T X_c^T X_c W)$

Step 4: The Simplified Equation

Substitute the simplified terms back into $J(W)$:

$$J(W) = \mathrm{Tr}(X_c^T X_c) - \underbrace{\mathrm{Tr}(\ldots)}_{\text{Term 2}} - \underbrace{\mathrm{Tr}(\ldots)}_{\text{Term 3}} + \underbrace{\mathrm{Tr}(\ldots)}_{\text{Term 4}}$$

$$= \mathrm{Tr}(X_c^T X_c) - 2\mathrm{Tr}(W^T X_c^T X_c W) + \mathrm{Tr}(W^T X_c^T X_c W)$$

Result:

$$J(W) = \mathrm{Tr}(X_c^T X_c) - \mathrm{Tr}(W^T X_c^T X_c W)$$

Step 5: Min to Max

Analyze the optimization:

$$\min_W \left(\underbrace{\mathrm{Tr}(X_c^T X_c)}_{\text{Constant}} - \underbrace{\mathrm{Tr}(W^T X_c^T X_c W)}_{\text{Variable}} \right)$$

- To **minimize** the total error, we must **maximize** the variable term.

$$\iff \max_W \mathrm{Tr}(W^T X_c^T X_c W)$$

Equivalence Derivation Conclusion

Recall that $\|A\|_F^2 = \mathrm{Tr}(A^T A)$. Let's apply this to our Projected Data ($X_c W$):

$$\mathrm{Tr}(W^T X_c^T X_c W) = \mathrm{Tr}((X_c W)^T (X_c W)) = \|X_c W\|_F^2$$

This matches our maximization term exactly. Therefore:

$$\arg \min_{W : W^T W = I} \|X_c - (X_c W) W^T\|_F^2 \equiv \arg \max_{W : W^T W = I} \|X_c W\|_F^2$$

The Unifying Solution

We've now defined PCA in two different ways—minimizing reconstruction error and maximizing projected variance. Remarkably, they both have the same solution. This connects the Singular Value Decomposition (SVD) of the data matrix to the Eigendecomposition of its covariance matrix.

Equivalent Solutions: The Solution to Both Problems is The Top k Right Singular Vectors of X_c

Minimize reconstruction error

$$\min_{W:W^TW=I_k} \|X_c - (X_cW)W^T\|_F^2$$

- Singular value decomposition (SVD) of $X_c = USV^T$
- Solution: $W^* = V_{1:k}$

Maximize sum of squares of latent projection (equivalent solution)

$$\max_{W:W^TW=I} \|X_cW\|_F^2$$

- Eigendecomposition of $X_c^TX_c = Q\Lambda Q^T$
- Solution is top k eigenvectors $W^* = Q_{1:k}$
- This is equivalent to SVD solution because:
- $X_c^TX_c = (USV^T)^T(USV^T)$
- $= VSU^TUSV^T = VS(U^TU)SV^T = VS^2V^T$

PCA Demos and Examples

PCA Demo: Visualization and Dimensionality Reduction

(Edited by David I. Inouye for classroom use) The text has been removed and the code has been edited and reordered as seemed appropriate.

This notebook contains an excerpt from the Python Data Science Handbook by Jake VanderPlas; the content is available on GitHub.

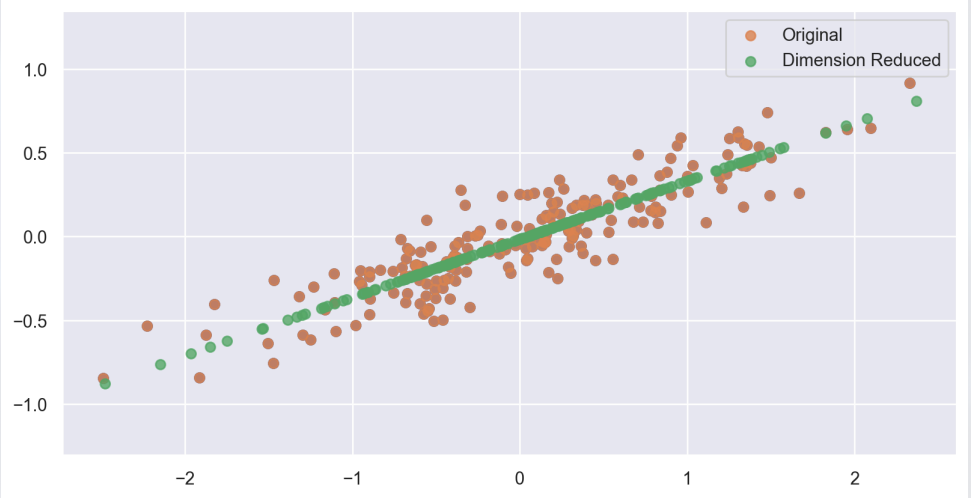

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.decomposition import PCA
```

PCA Demo: Minimum reconstruction error / dimensionality reduction viewpoint of PCA

This shows dimension reduction from 2D to 1D in a minimum reconstruction way.

```
rng = np.random.RandomState(1)
X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
plt.scatter(X[:, 0], X[:, 1])
plt.axis('equal');

pca = PCA(n_components=1)
pca.fit(X)
X_pca = pca.transform(X)
print("original shape:   ", X.shape)
print("transformed shape:", X_pca.shape)

X_new = pca.inverse_transform(X_pca)
plt.scatter(X[:, 0], X[:, 1], alpha=0.8, label='Original')
plt.scatter(X_new[:, 0], X_new[:, 1],
    alpha=0.8, label='Dimension Reduced')
plt.axis('equal');
plt.legend()
```

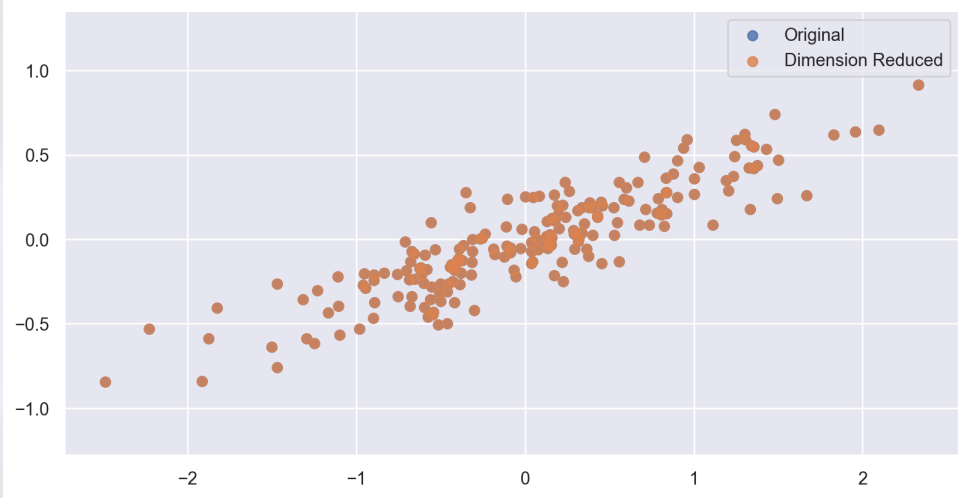
```
original shape:    (200, 2)
transformed shape: (200, 1)
```



PCA Demo: If we keep all components, then we get perfect reconstruction

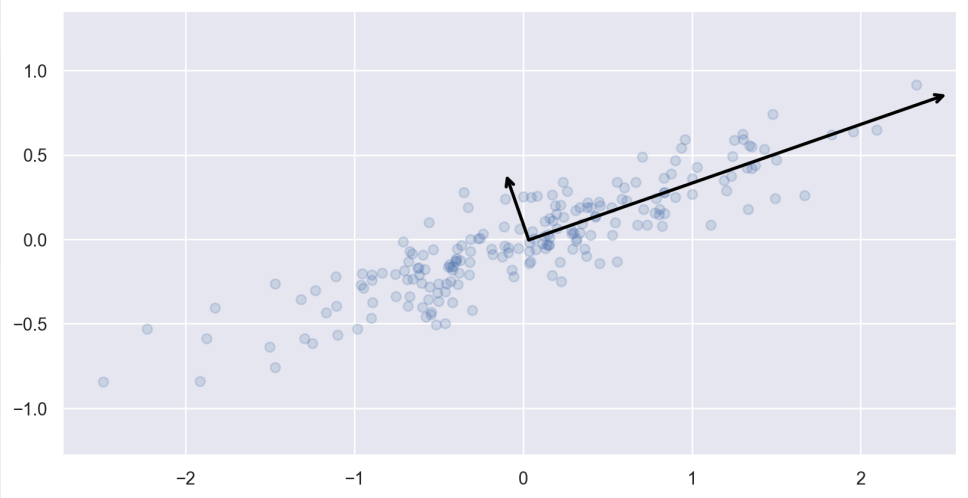
```python
# Keep both dimensions
pca = PCA(n_components=2)
pca.fit(X)
X_pca = pca.transform(X)
print("original shape:   ", X.shape)
print("transformed shape:", X_pca.shape)

X_new = pca.inverse_transform(X_pca)
plt.scatter(X[:, 0], X[:, 1], alpha=0.8, label='Original')
plt.scatter(X_new[:, 0], X_new[:, 1],
    alpha=0.8, label='Dimension Reduced')
plt.axis('equal');
plt.legend()
```

```
original shape:    (200, 2)
transformed shape: (200, 2)
```



PCA Demo: Maximum variance of projected data viewpoint of PCA

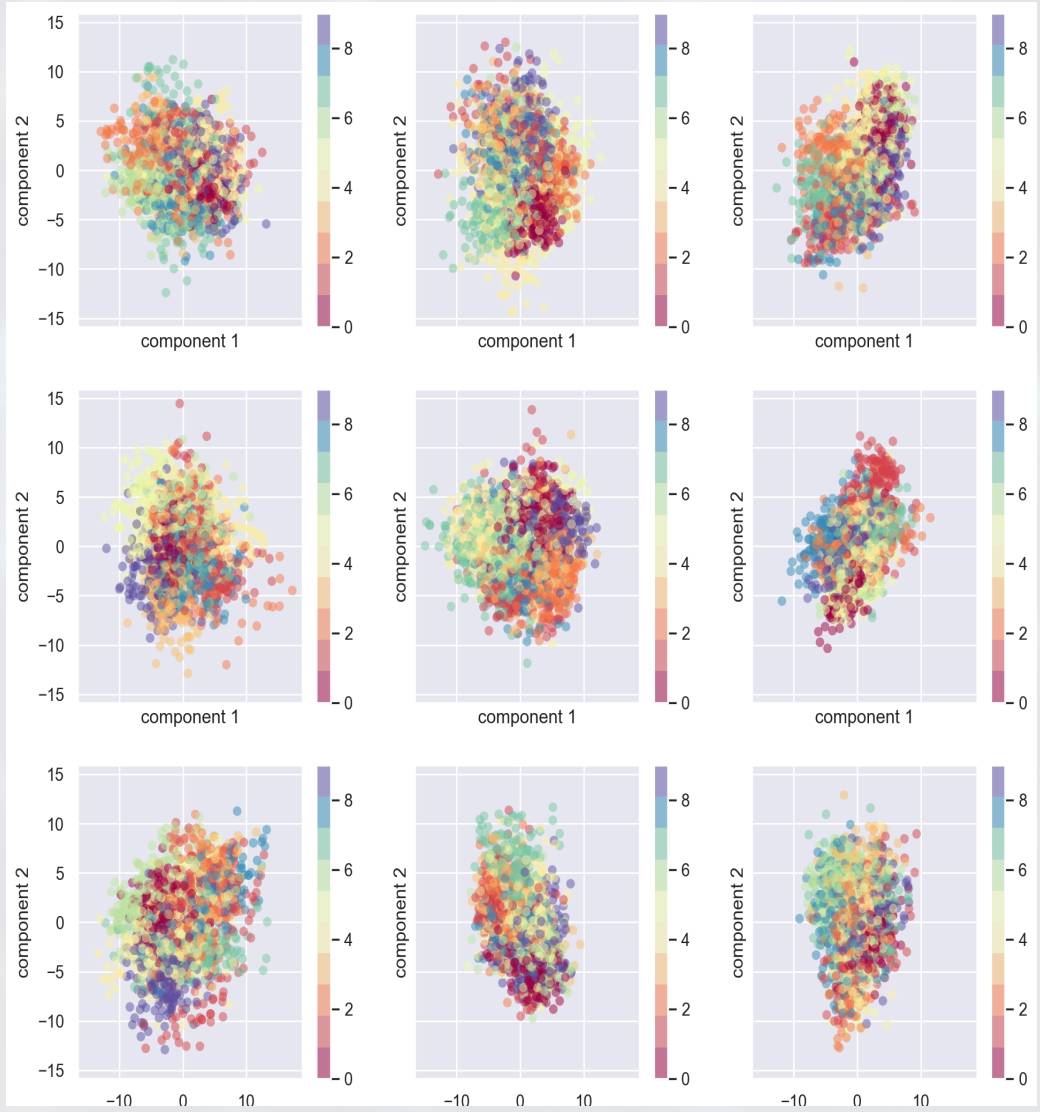
```
pca = PCA(n_components=2)
pca.fit(X)

#print(pca.components_)
#print(pca.explained_variance_)

def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0, color='black')
    ax.annotate('', v1, v0, arrowprops=arrowprops)

## plot data
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
for length, vector in zip(pca.explained_variance_, pca.compon
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal');
```



PCA Demo: Random 2D projections of hand-written digits

```python
from sklearn.datasets import load_digits
digits = load_digits()
X = digits.data
X = X - np.mean(X, axis=0)
y = digits.target
print(X.shape)

def show_projected(projected, y, ax=None):
    if ax is None:
        ax = plt.gca()
    sc = ax.scatter(projected[:, 0], projected[:, 1],
                    c=y, edgecolor='none', alpha=0.5,
                    cmap=plt.cm.get_cmap('Spectral', 10))
    ax.set_xlabel('component 1')
    ax.set_ylabel('component 2')
    plt.colorbar(sc, ax=ax)
    return sc

rng = np.random.RandomState(0)
n_rows, n_cols = 3, 3
fig, axes = plt.subplots(n_rows, n_cols, figsize=(12, 12), sh
for ax in axes.ravel():
    A = rng.randn(X.shape[1], 2)
    Q, _ = np.linalg.qr(A)
    Z = np.dot(X, Q)
    sc = show_projected(Z, y, ax=ax)
```

```
(1797, 64)
```

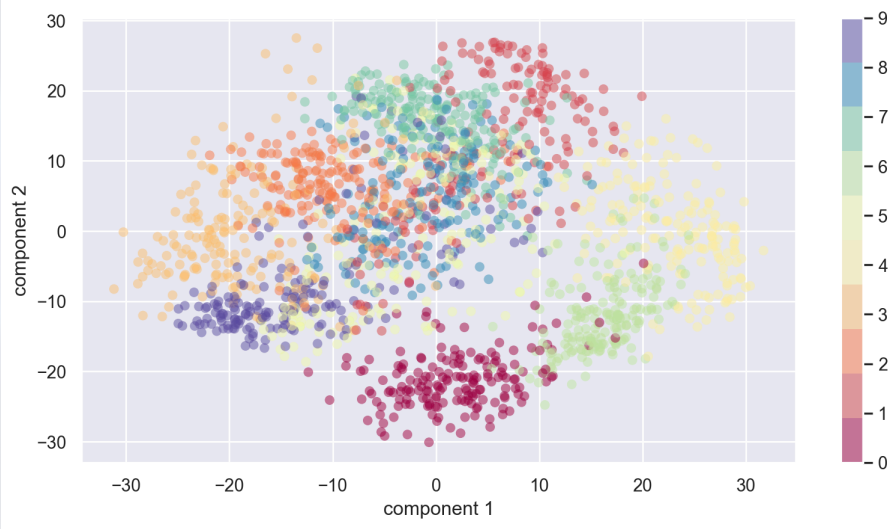


PCA Demo: Now let's use Principal Component Analysis (PCA)

Notice that the limits of the component are [-30, 30] rather than [-10, 10] as with random projections. This is because PCA finds the directions of maximum variance.

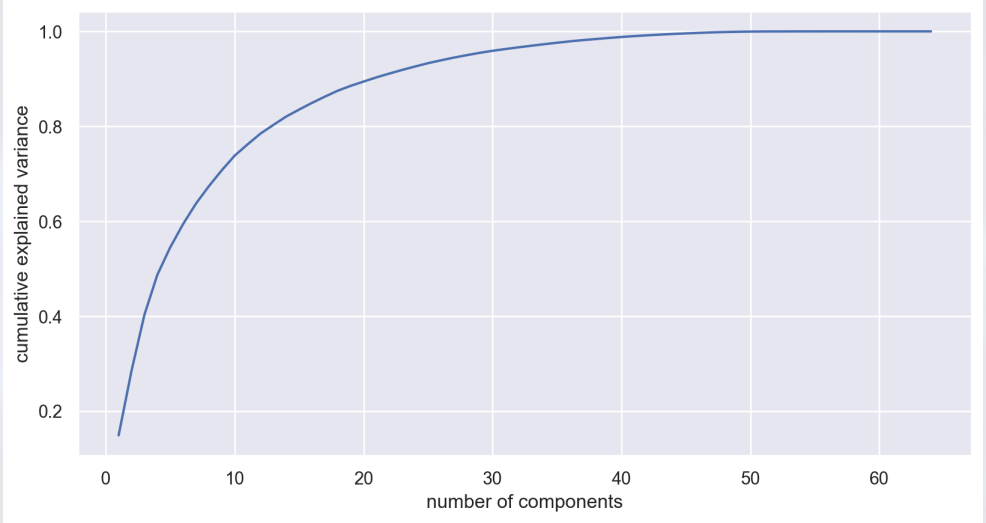
```
pca = PCA(2)  # project from 64 to 2 dimensions
projected = pca.fit_transform(digits.data)
print(digits.data.shape)
print(projected.shape)

plt.scatter(projected[:, 0], projected[:, 1],
            c=digits.target, edgecolor='none', alpha=0.5,
            cmap=plt.cm.get_cmap('Spectral', 10))
plt.xlabel('component 1')
plt.ylabel('component 2')
plt.colorbar();
```

```
(1797, 64)
(1797, 2)
```



Amount of variance explained increases as the number of components increases while the first components capture the most variance

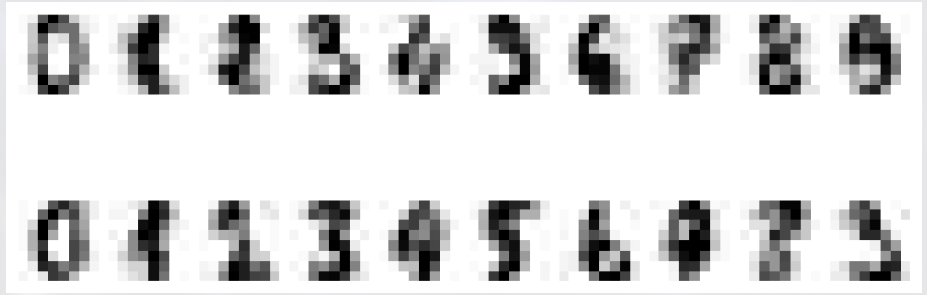
```
pca = PCA().fit(digits.data)
plt.plot(np.arange(64)+1, np.cumsum(pca.explained_variance_ra
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



PCA Example: PCA as Noise Filtering

```python
def plot_digits(data):
    fig, axes = plt.subplots(2, 10, figsize=(10, 4),
                             subplot_kw={'xticks':[], 'yticks
                             gridspec_kw=dict(hspace=0.1, wsp
    for i, ax in enumerate(axes.flat):
        ax.imshow(data[i].reshape(8, 8),
                  cmap='binary', interpolation='nearest',
                  clim=(0, 16))
plot_digits(digits.data)

np.random.seed(42)
noisy = np.random.normal(digits.data, 4)
plot_digits(noisy)

pca = PCA(12).fit(noisy)
print(noisy.shape)
scores = pca.transform(noisy)
print(scores.shape)
filtered = pca.inverse_transform(scores)
print(filtered.shape)
plot_digits(filtered)
```

```
(1797, 64)
(1797, 12)
(1797, 64)
```



PCA Example: Eigenfaces

```
from sklearn.datasets import fetch_lfw_people
faces = fetch_lfw_people(min_faces_per_person=60)
print(faces.target_names)
print(faces.images.shape)

fig, axes = plt.subplots(1, 8, figsize=(9, 4),
                         subplot_kw={'xticks':[], 'yticks':[]
                         gridspec_kw=dict(hspace=0.1, wspace=
for i in range(8):
    axes[i].imshow(faces.data[i].reshape(62, 47), cmap='binar
```

```
['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W
Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Junichiro Koizumi' 'Tony
Blair']
(1348, 62, 47)
```

PCA Example: Eigenfaces are the principal components of the faces dataset

```
pca = PCA(n_components=150, svd_solver='randomized', whiten=T
pca.fit(faces.data)

fig, axes = plt.subplots(3, 8, figsize=(9, 4),
                         subplot_kw={'xticks':[], 'yticks':[]
                         gridspec_kw=dict(hspace=0.1, wspace=
for i, ax in enumerate(axes.flat):
    ax.imshow(pca.components_[i].reshape(62, 47), cmap='bone'

## Compute the components and projected faces
scores = pca.transform(faces.data)
projected = pca.inverse_transform(scores)

## Plot the results
fig, ax = plt.subplots(2, 10, figsize=(10, 2.5),
                       subplot_kw={'xticks':[], 'yticks':[]},
                       gridspec_kw=dict(hspace=0.1, wspace=0.
for i in range(10):
    ax[0, i].imshow(faces.data[i].reshape(62, 47), cmap='bina
    ax[1, i].imshow(projected[i].reshape(62, 47), cmap='binar

ax[0, 0].set_ylabel('full-dim\ninput')
ax[1, 0].set_ylabel('150-dim\nreconstruction');
```

Summary: Linear Dimensionality Reduction (PCA)

1. **The Motivation** High-dimensional data (images, genetics, text) is computationally expensive and difficult to interpret. PCA reduces dimensions ($d \to k$) to achieve:
   - **Efficiency:** Faster training times ($1000\times$ speedups).
   - **Visualization:** Plotting complex data in 2D/3D.
   - **Denoising:** Reconstructing data from signals, ignoring noise.
2. **Two Equivalent Intuitions** PCA can be derived from two seemingly different goals that yield the **same mathematical solution**:
   - **Viewpoint A:** Minimize the *Reconstruction Error* (closest fit).
   - **Viewpoint B:** Maximize the *Projected Variance* (widest spread of data).
3. **The Solution via SVD or Eigen Decomposition** For centered data X_c, the optimal linear projection is found using **Singular Value Decomposition (SVD)**:
   - $X_c = USV^T$
   - The Principal Components (W^*) are the top k **Right Singular Vectors** ($V_{1:k}$).
   - Equivalent to the top k eigenvectors of the (scaled) covariance matrix $X_c^T X_c$.