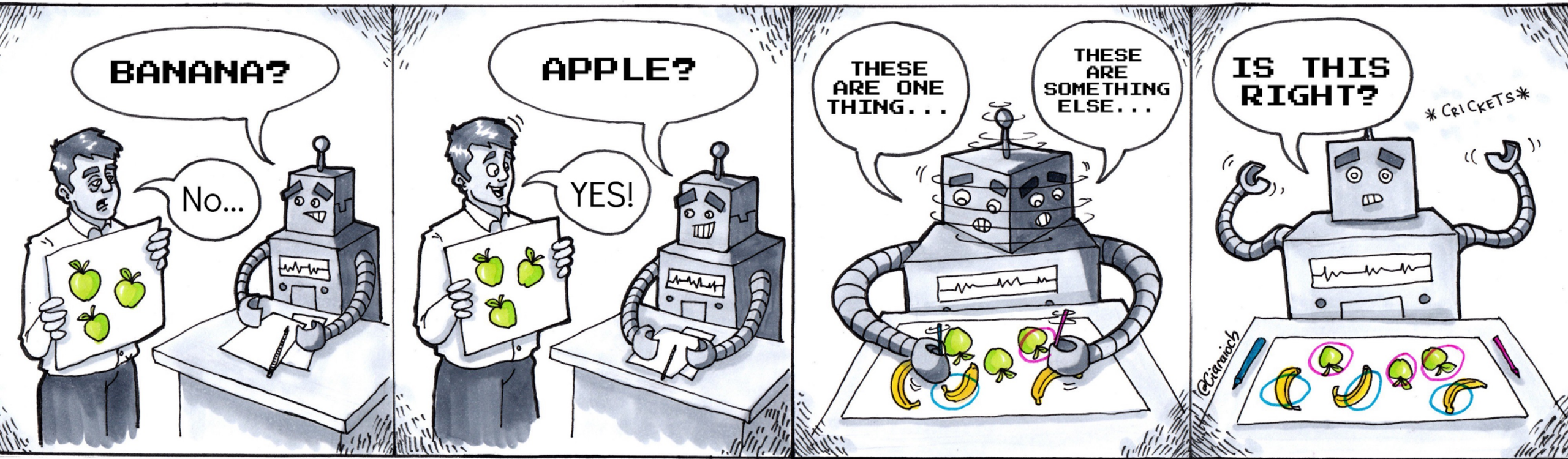
CSC 480/580 Principles of Machine Learning

Unsupervised learning

Jason Pacheco

Department of Computer Science

THE UNIVERSITY OF ARIZONA

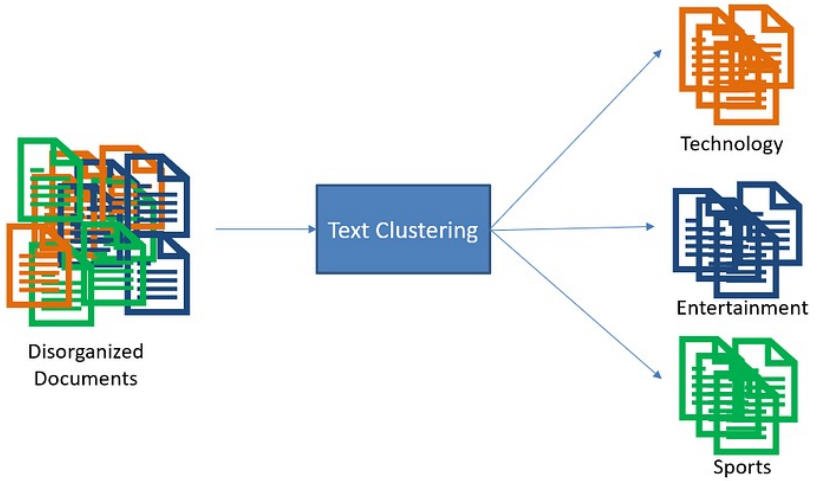


Supervised Learning

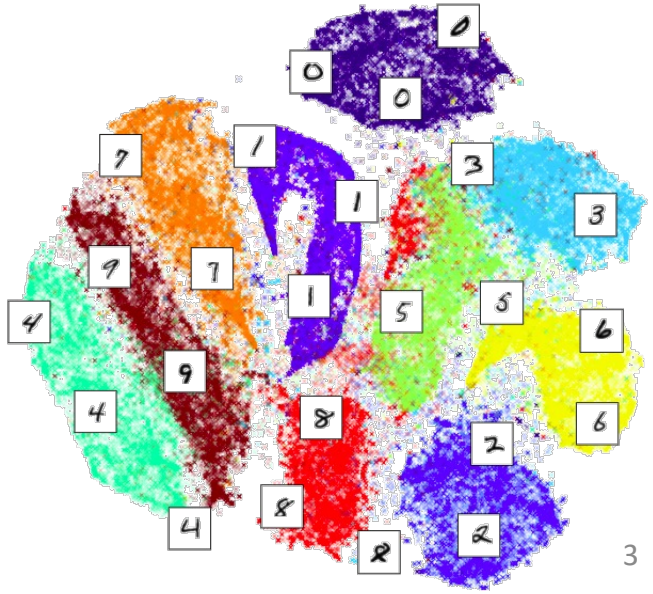
Unsupervised Learning

What is unsupervised learning?

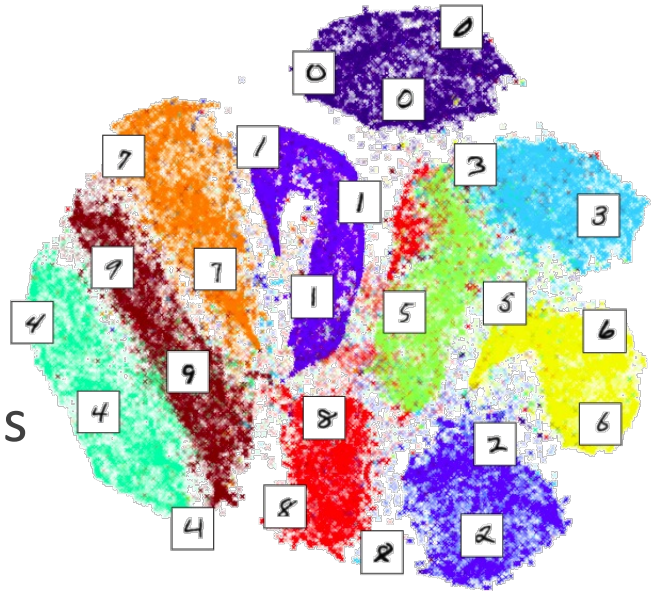
- Uncovering structures in unlabeled data
- What can we expect to learn?
  - **Clustering**: obtain partition of the data that are well-separated.
    - can be viewed as a preliminary classification without predefined class labels.



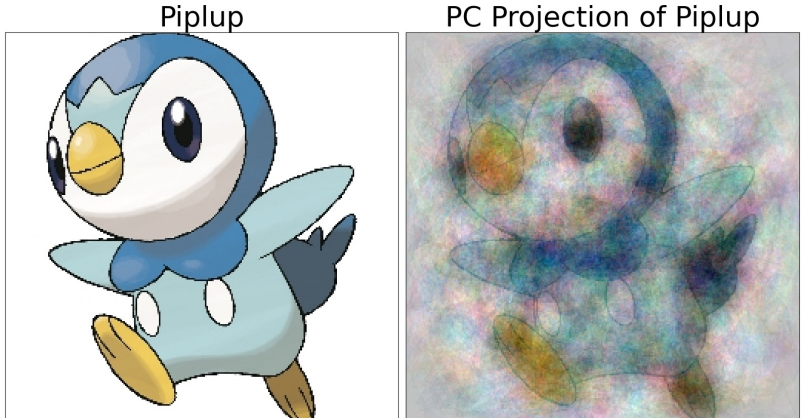
  - **Component analysis**: extract common components that compose data points.
    - e.g., Given a set of 28x28 handwritten digit images, find two “important” directions in the data and project data onto these two directions



Unsupervised learning

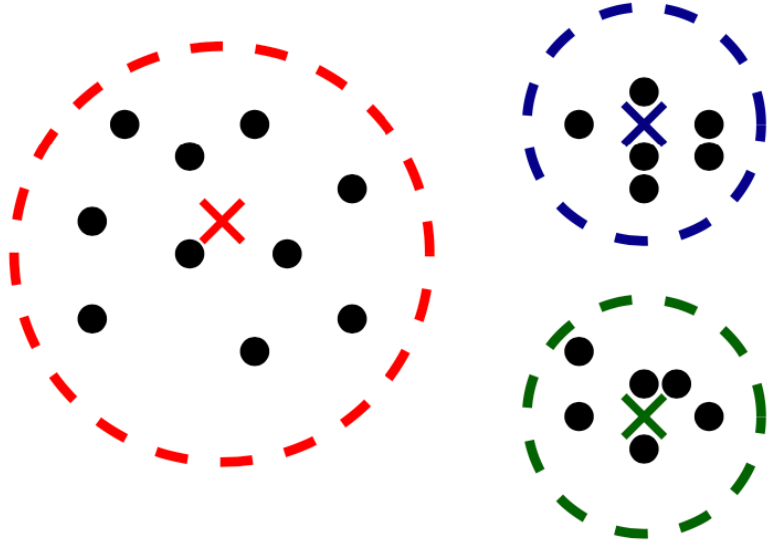


- Usage
  - Exploratory data analysis
    - e.g., do useful visualization using principal component analysis
    - what are the patterns we can get even without labels?
  - Data preprocessing & data compression
    - My ML algorithm has a running time O(D), D = #features
    - Wouldn't it be nice if I can reduce D from 10000 to say, 10?

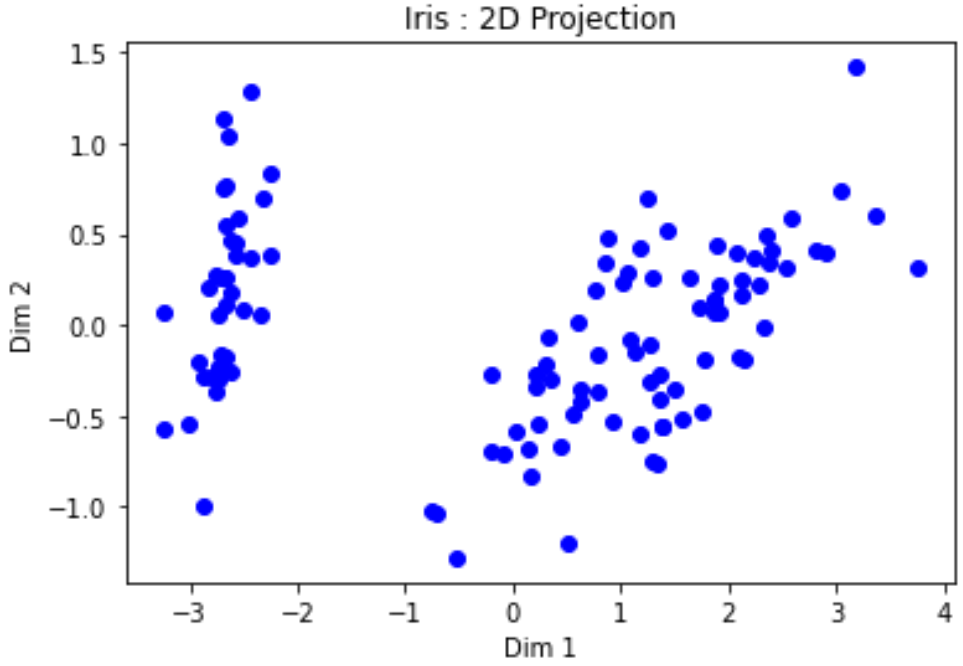


Outline

- Clustering
  - K-means clustering (we've seen it before)



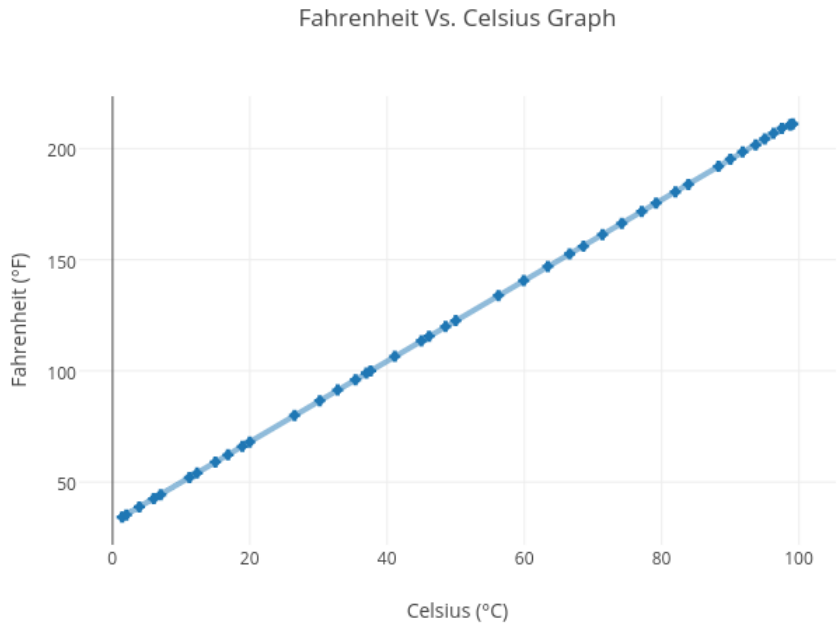
- **Dimensionality reduction**
  - **Principal Component Analysis (PCA)**



Motivation

…data may have strongly dependent features…

Fahrenheit	Celsius
3.1	-16.1
100.5	38.1
27.3	-2.6
18.1	-7.7
18.9	-7.3
21.7	-5.7
…	…

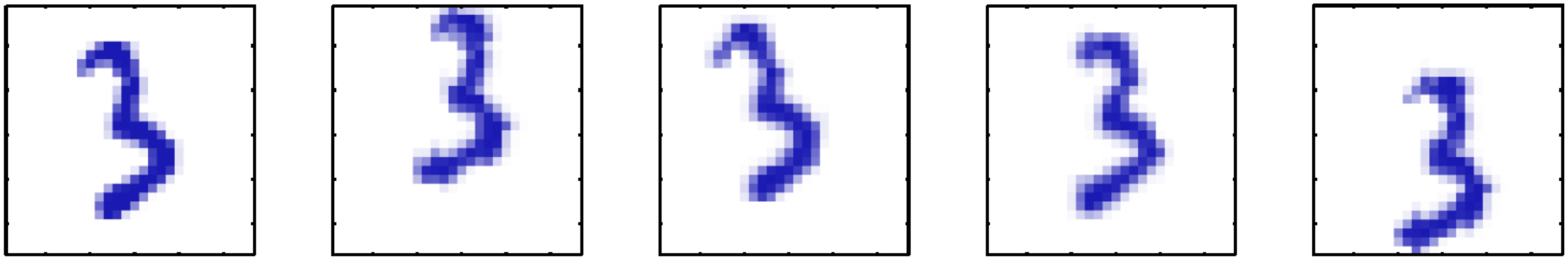


Linear Function

$$F = 1.8C + 32$$

Motivation

or data are high-dimensional but have low *intrinsic dimension*…



Example A dataset consisting of a hand-drawn 3 at random locations and rotations in a 100x100 pixel image.

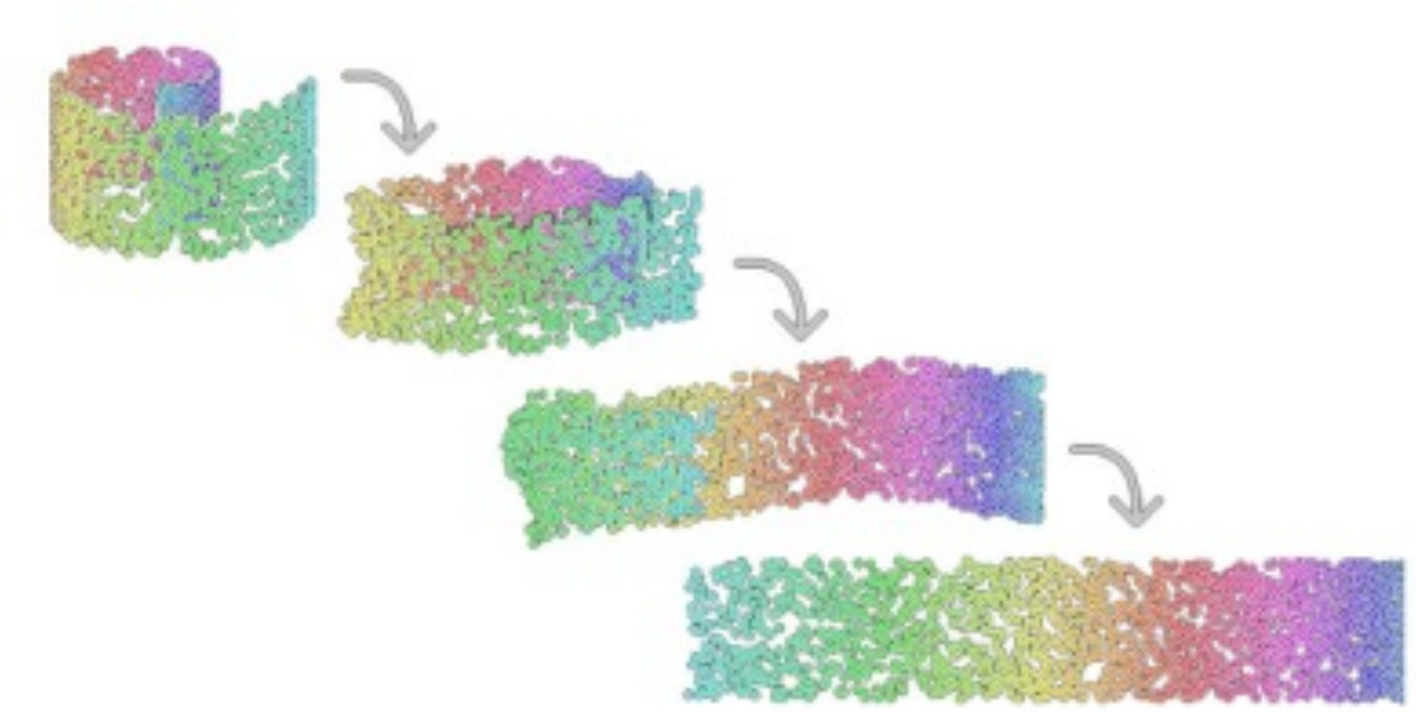
Data Dimension 100 x 100 = 10,000

Intrinsic Dimension 3 (X-position, Y-position, Rotation)

[Source: Bishop, C.]

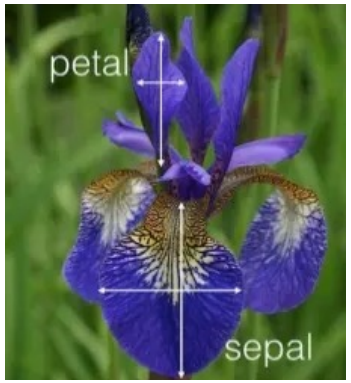
Motivation

data are high-dimensional but have low *intrinsic dimension*…

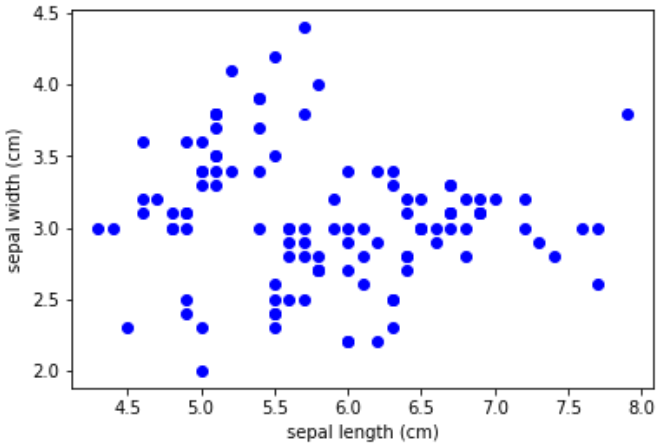


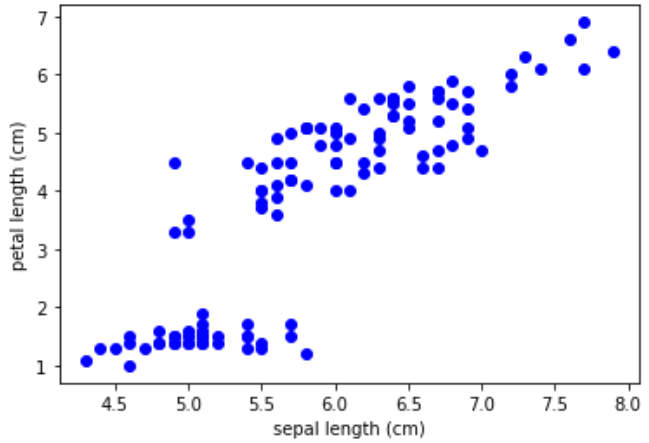
…in all cases, finding lower-dimensional representation is useful

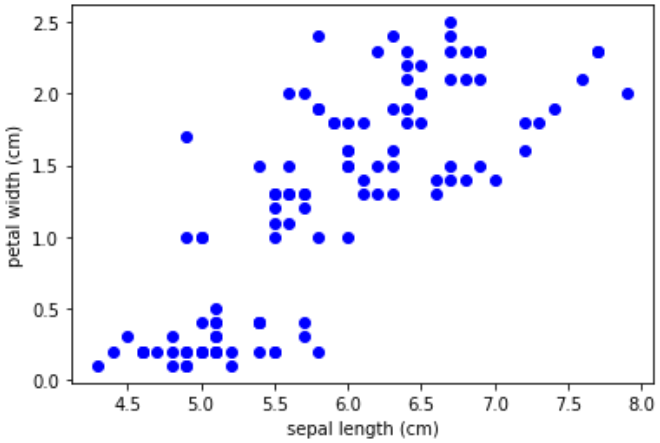
Example: Iris Dataset

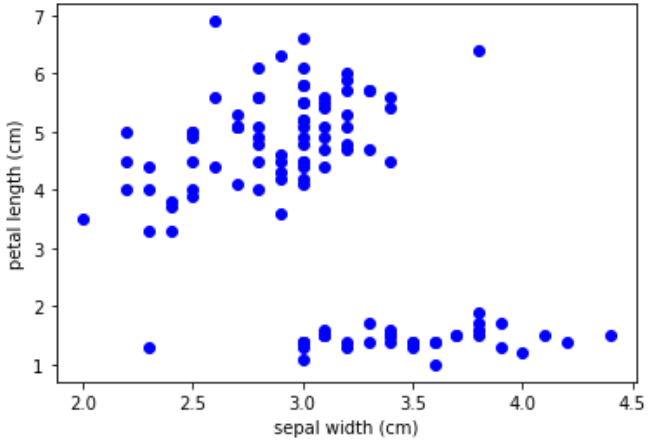


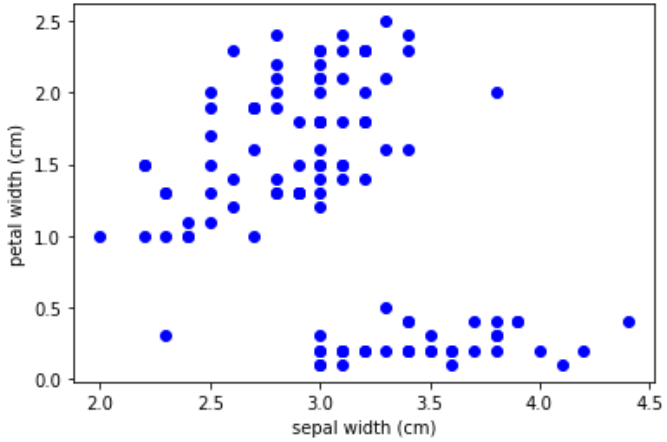
Recall that the Iris dataset has 4 features:
sepal length / width, petal length / width…

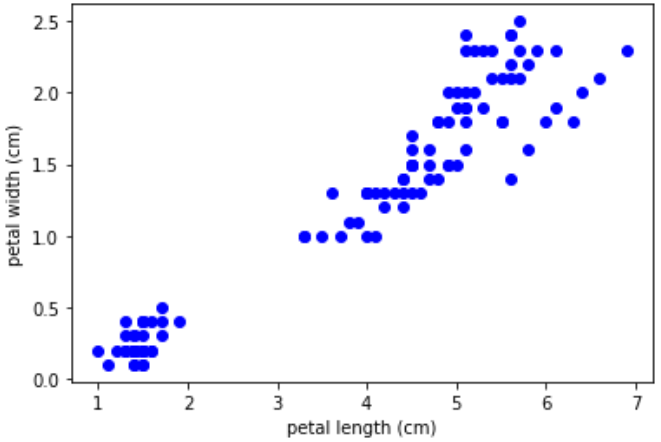




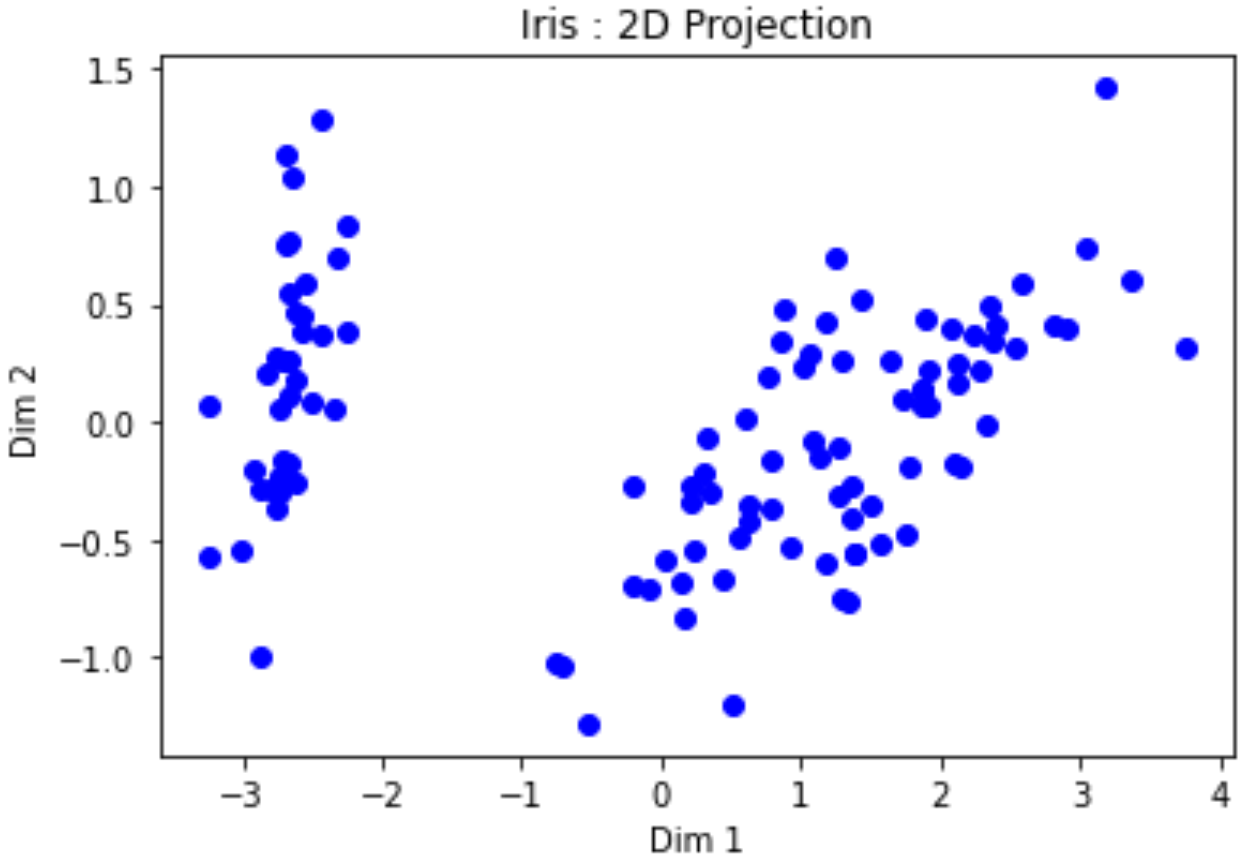








Example : Iris Dataset



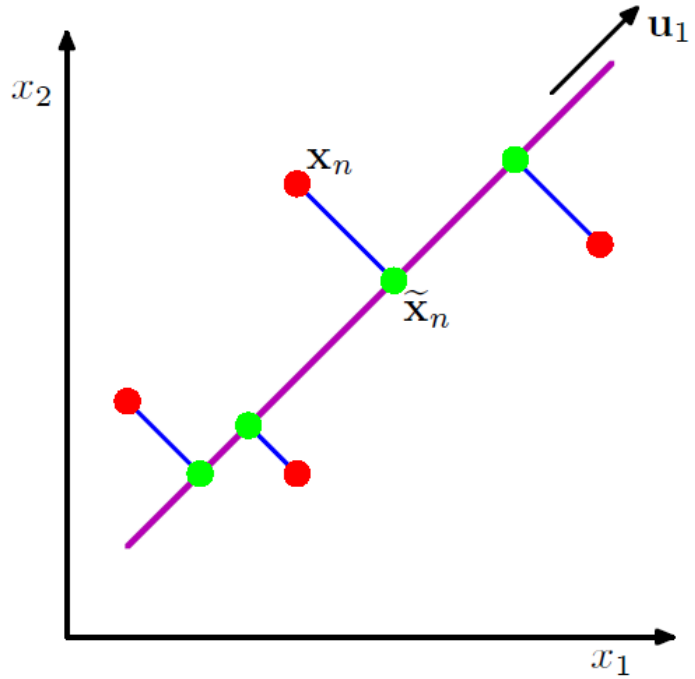
Data still cluster in a two-dimensional subspace

We can represent data in 2D to reduce complexity, visualize results, etc.

Linear Dimensionality Reduction



Project data onto a line or plane…

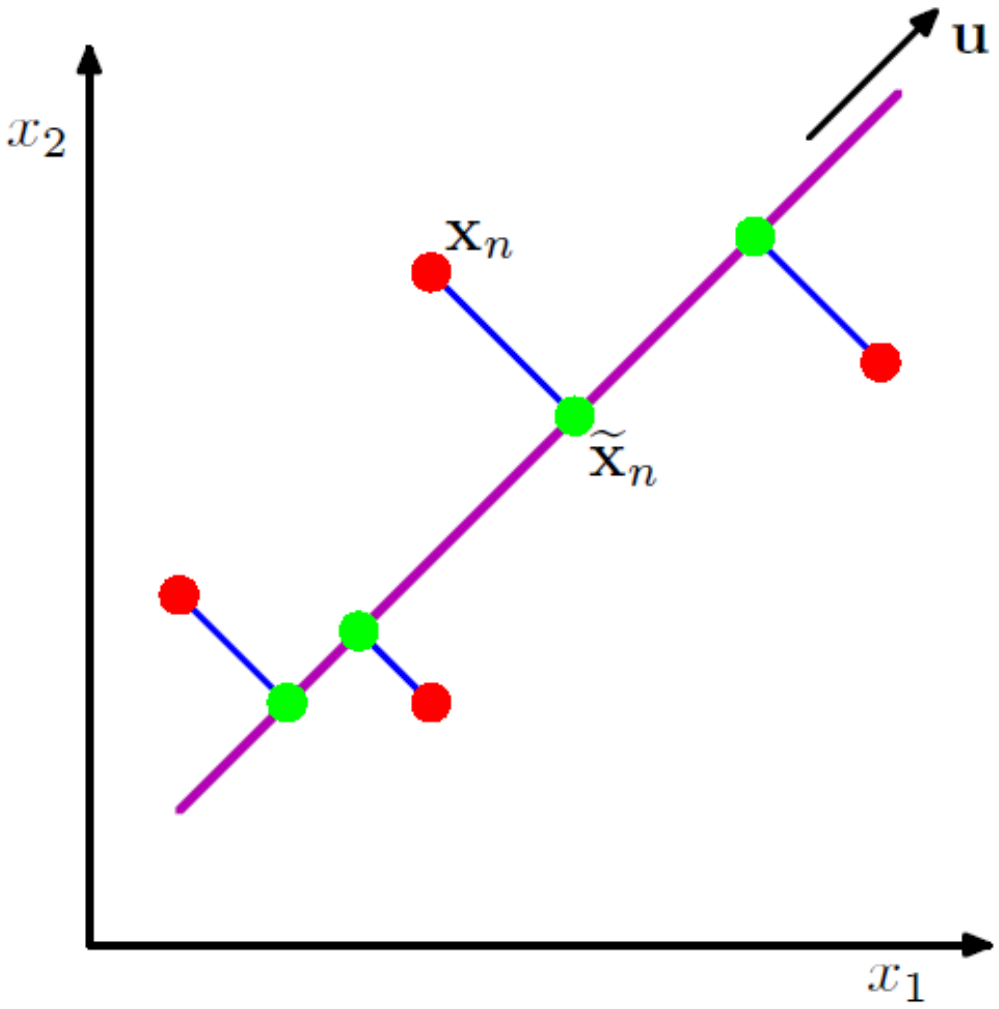


…one of the simplest dimensionality reduction approaches

First, let's review some linear algebra…

[Source: Manfred Warmuth; Bishop, C.]

Linear Dimensionality Reduction



Projecting data onto a vector u is a simple inner product,

$$\widetilde{x}_n = u^T x_n$$

We call u the *linear subspace*

Question Why would dimensionality reduction be better than feature selection (e.g. choose 1-D features X1 or X2)?

[Source: Bishop, C.]

Linear Dimensionality Reduction

Projecting data onto a vector u is a simple inner product,

$$\widetilde{x}_n = u^T x_n$$

We call u the *linear subspace*

Projecting data onto a vector u is a simple inner product,

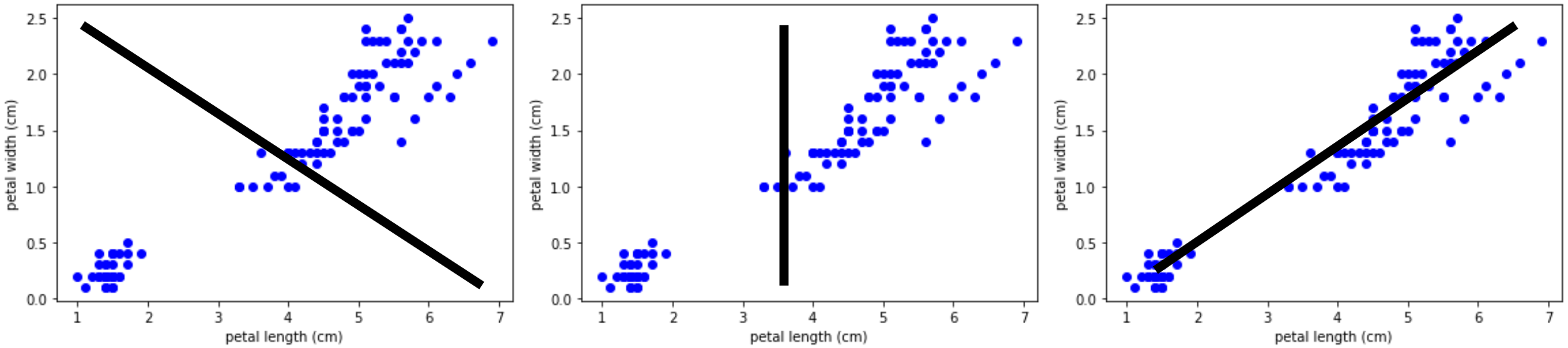
Answer No features are discarded (uses all the data),

$$\widetilde{x}_n = u_1 x_{n1} + u_2 x_{n2}$$

[Source: Bishop, C.]

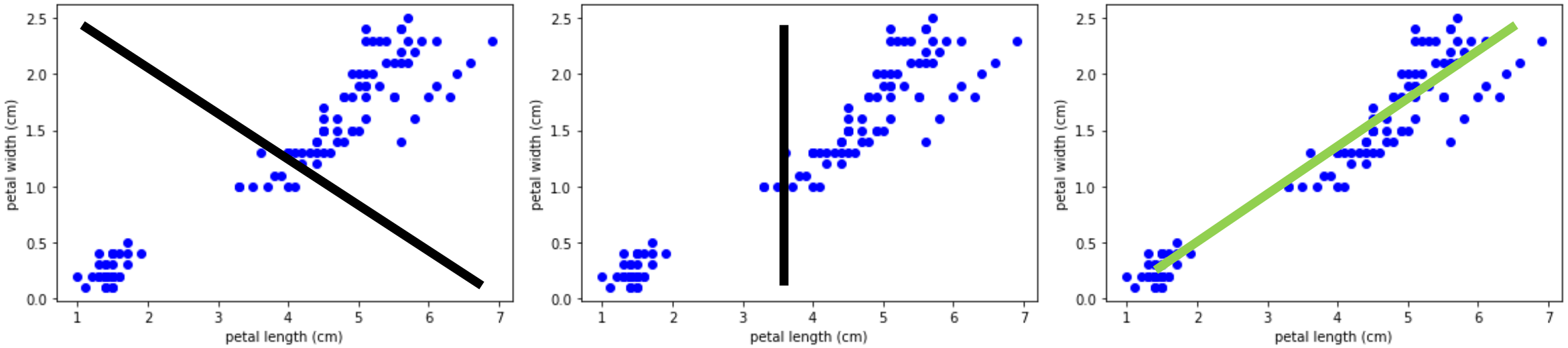
Linear Dimensionality Reduction

Which choice of subspace is best? And why?



Linear Dimensionality Reduction

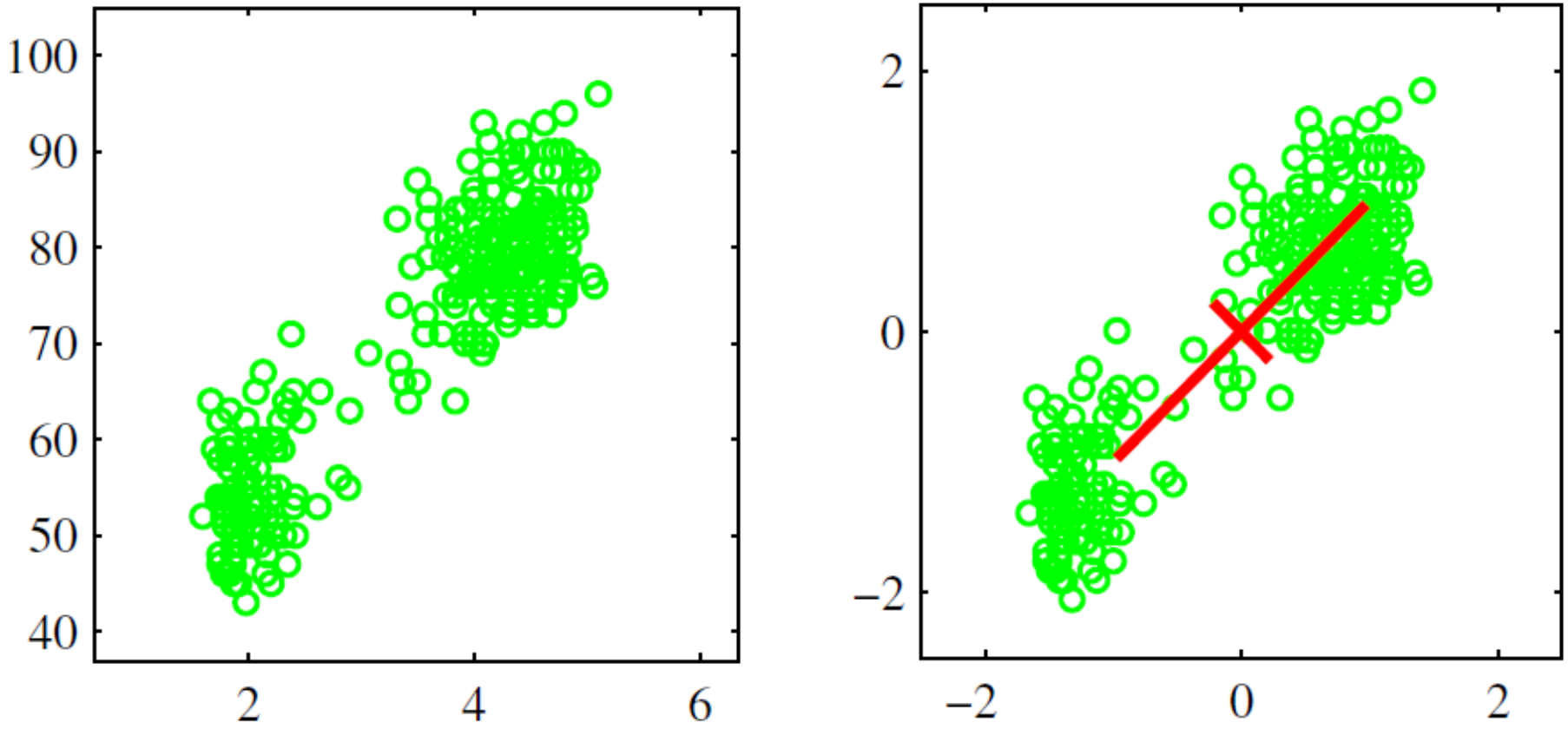
Which choice of subspace is best? And why?



Idea Choose the subspace that *captures the most variation* in the original data

Principal Component Analysis (PCA)

Identify directions of *maximum variation* as subspaces…



…we call each direction a *principal component*

[Source: Bishop, C.]

Principal Component Analysis (PCA)

First, center the data by subtracting the sample mean,

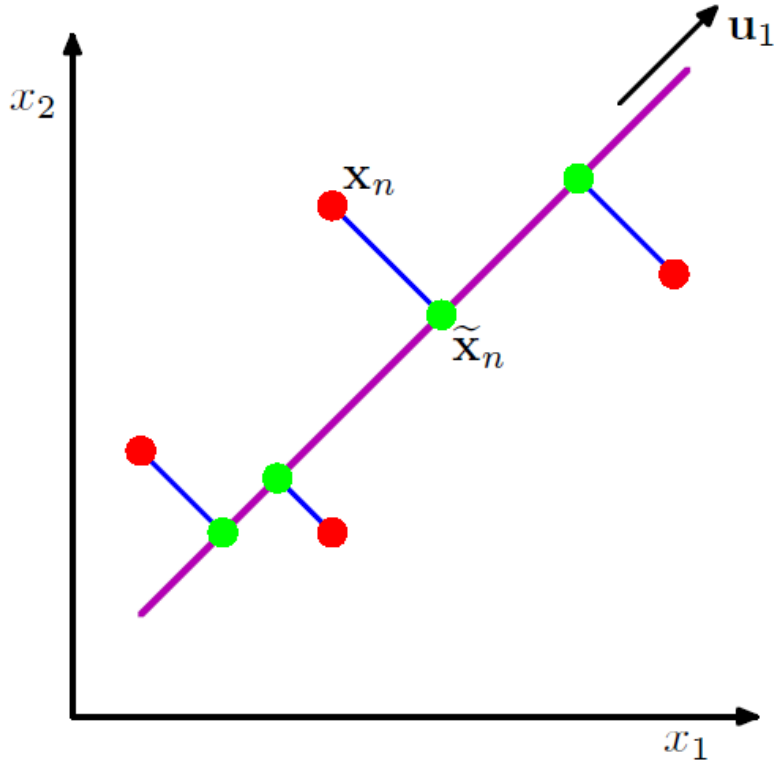
$$\bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$$

Variance of projected data,

$$\frac{1}{N}\sum_{n=1}^{N}\left(u^T x_n - u^T \bar{x}\right)^2$$

Projection of n^{th} data point

Projection of mean



Maximum Variance Formulation

A little algebra…

$$\frac{1}{N}\sum_{n=1}^{N}\left(u^T x_n - u^T \bar{x}\right)^2 = \frac{1}{N}\sum_{n=1}^{N}\left\{u^T(x_n - \bar{x})\right\}^2$$

Pull out u

Quadratic form

$$= \frac{1}{N}\sum_{n=1}^{N} u^T(x_n - \bar{x})(x_n - \bar{x})^T u$$

Define: $S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^T$

What is this? Data covariance matrix

Then: $\frac{1}{N}\sum_{n=1}^{N}\left(u^T x_n - u^T \bar{x}\right)^2 = u^T S u$ ←

This is what we will optimize over u

Maximum Variance Formulation

Find u so that projected variance is maximal…

$$\max_u \; u^T S u$$

Don't want to *cheat* with large magnitude u, so we add penalty,

$$\max_u \; u^T S u - \lambda u^T u$$

Set the derivative (gradient) to zero and solve…

$$Su - \lambda u = 0$$

$$Su = \lambda u$$

What does this equation mean?

u* is an *eigenvector* with *eigenvalue λ

Math recap: Linear Transformations

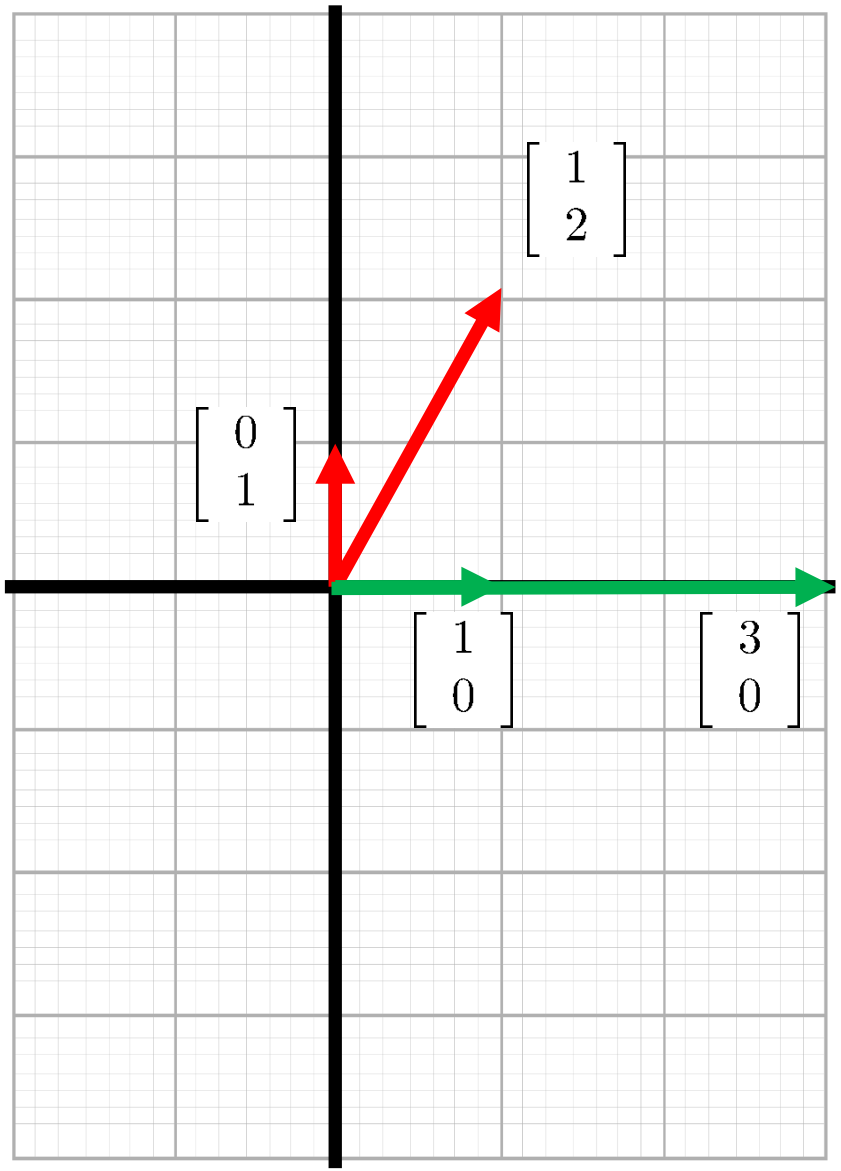
Consider the matrix: $\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$

Let's multiply it with some vectors…

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \cdot 0 + 1 \cdot 1 \\ 0 \cdot 0 + 2 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \cdot 1 + 1 \cdot 0 \\ 0 \cdot 1 + 2 \cdot 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

- Matrix transforms vectors from one basis to another
- Columns are transformation of standard basis



Eigenvalue & eigenvector

Observe that the X-axis vector just gets "stretched out",

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

Factoring out the 3 we have,

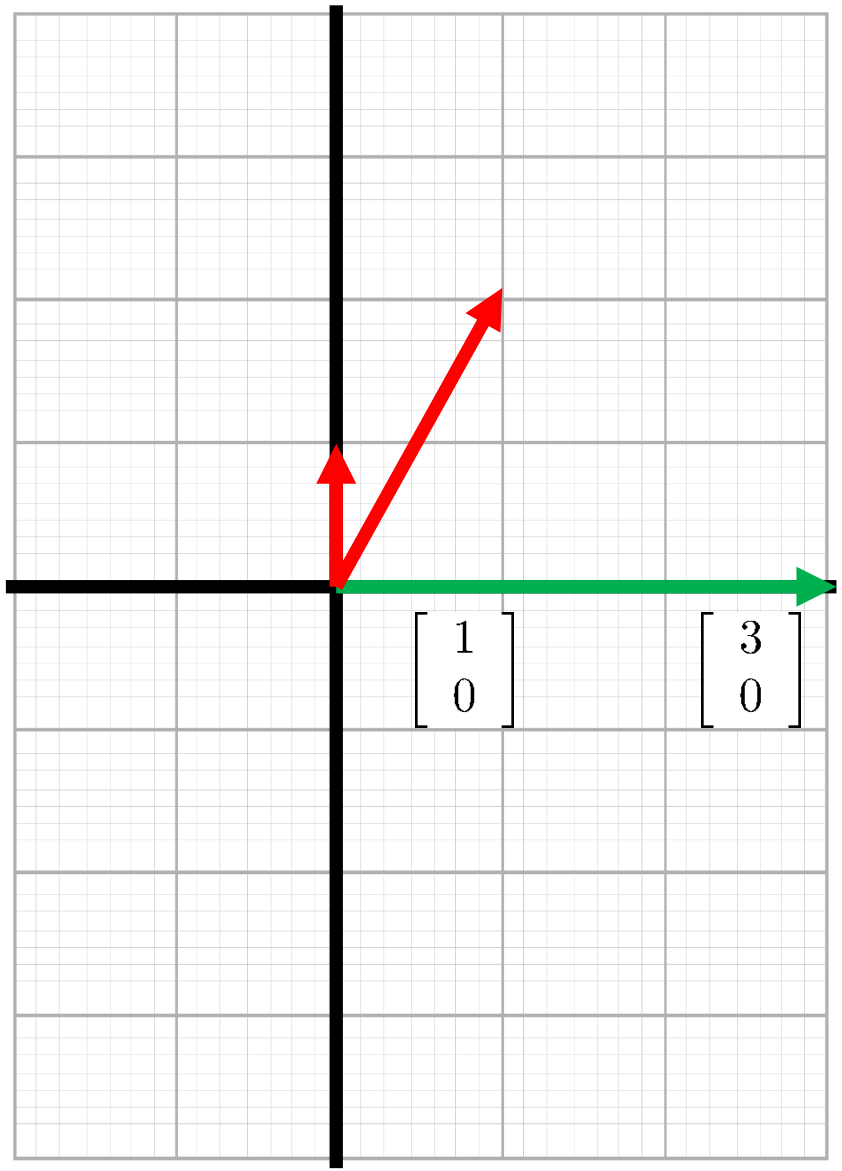
$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$\mathbf{S}$ $\mathbf{u}$ λ $\mathbf{u}$

Define some variables and we have the equation,

$$Su = \lambda u$$

So $(1,0)^T$ is an *eigenvector* of S with *eigenvalue* 3



Eigenvalue & eigenvector

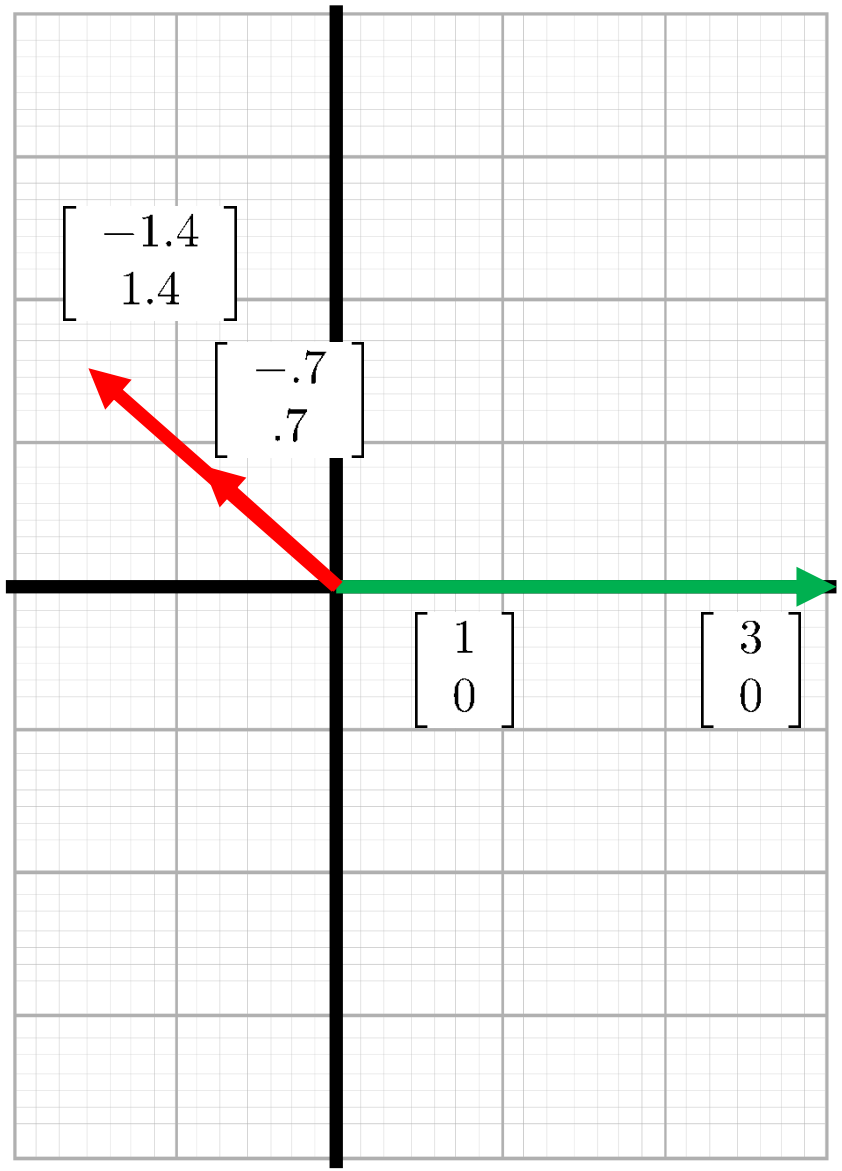
Transformation has one other eigenvector,

$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} -0.7 \\ 0.7 \end{bmatrix} = \begin{bmatrix} -1.4 \\ 1.4 \end{bmatrix} = 2 \begin{bmatrix} -0.7 \\ 0.7 \end{bmatrix}$$

- Complete eigen-representation of S

Eigenvectors $\begin{bmatrix} 1 & -0.7 \\ 0 & 0.7 \end{bmatrix}$ Eigenvalues $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$

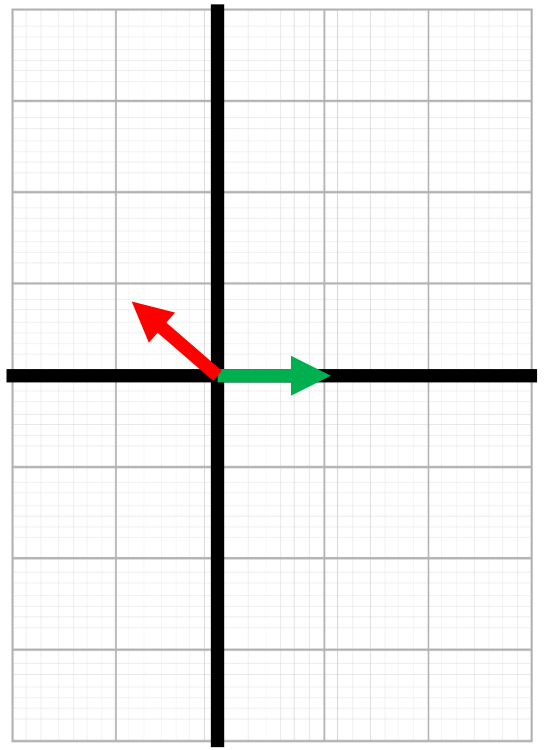
- Eigenvectors of linear transformation S are only stretched / shrunk / flipped
- Eigenvalues tell how much they are stretched / shrunk / flipped



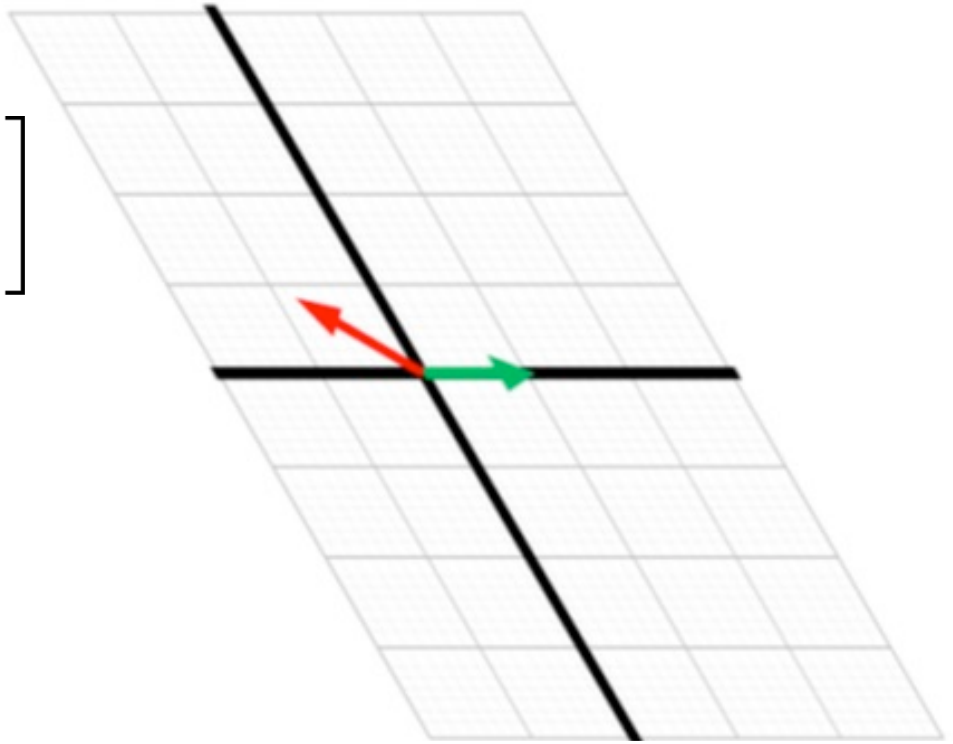
Eigenvalue & eigenvector

Eigenvectors $\begin{bmatrix} 1 & -0.7 \\ 0 & 0.7 \end{bmatrix}$ Eigenvalues $\begin{bmatrix} 3 \\ 2 \end{bmatrix}$

Eigenvalue & eigenvector highlight what a linear transformation does by identifying directions that are *not* altered



$$S = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$$

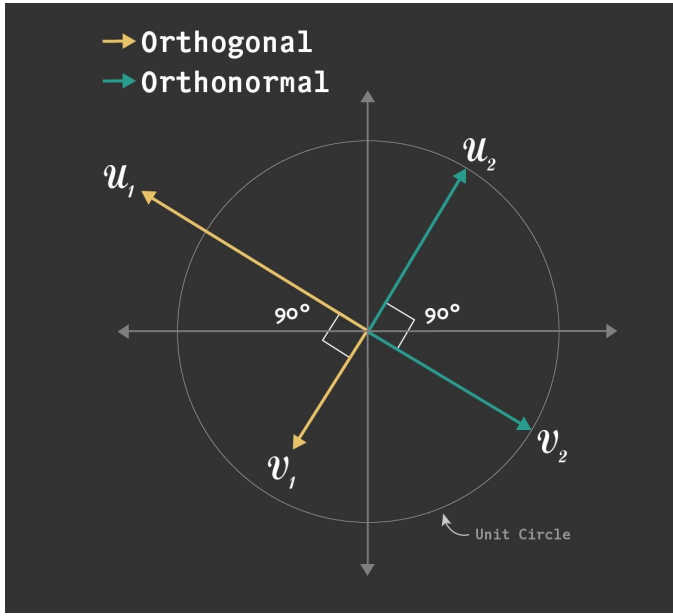


Eigenvalue & eigenvector

Eigenvectors / values of a matrix solve the equation

$$Su = \lambda u$$

- Matrix S may have *multiple* eigenvectors / eigenvalues that solve the above equation
- **Fact**: For any covariance matrix S over D-dimensional data, it has D orthonormal eigenvectors, each of which is associated with a nonnegative eigenvalue
- Orthonormal vectors $v_1, \ldots, v_D$: each $||v_i|| = 1$ and $v_i^T v_j = 0$

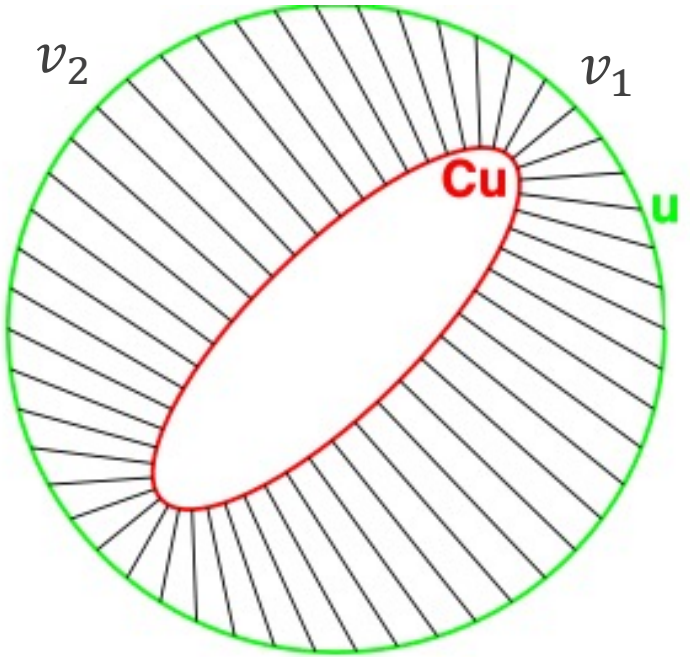


From now on, we assume that all eigenvectors are unit vectors

Eigenvectors and Ellipses

How does this connect to PCA?

Take all points on a unit circle and apply the linear transformation C



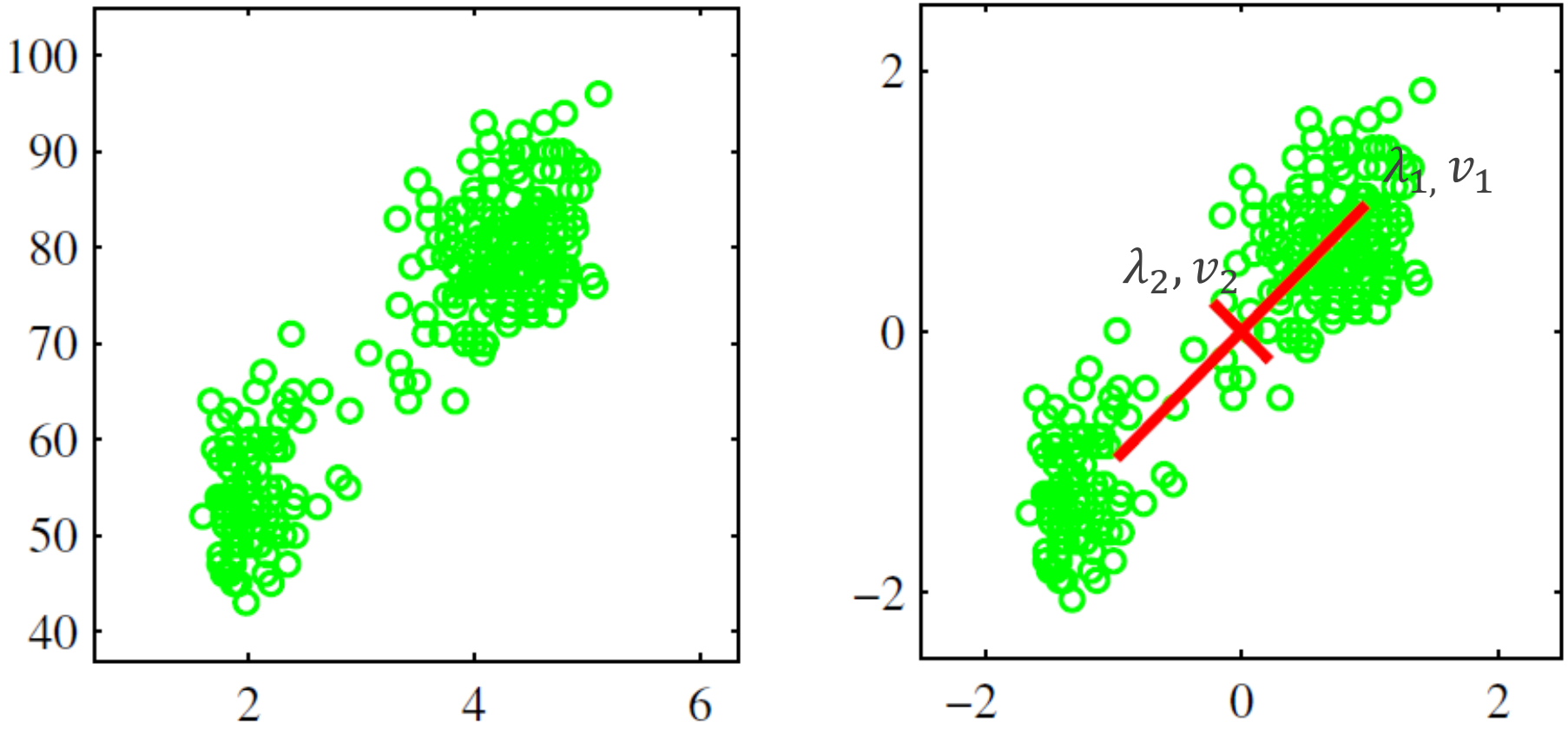
If C is a *covariance matrix,* then points will be transformed into an ellipse and…

- Eigenvectors are axes of ellipse
- Eigenvalues are length of each axes
- Sort eigenvalues to get major / minor / etc. axes
- In the context of PCA, eigenvectors = **principal components**

https://mwarmuth.bitbucket.io/pubs/C77talk.pdf

Principal Component Analysis (PCA)

Sort the covariance matrix's eigenvectors by their eigenvalues in descending order…

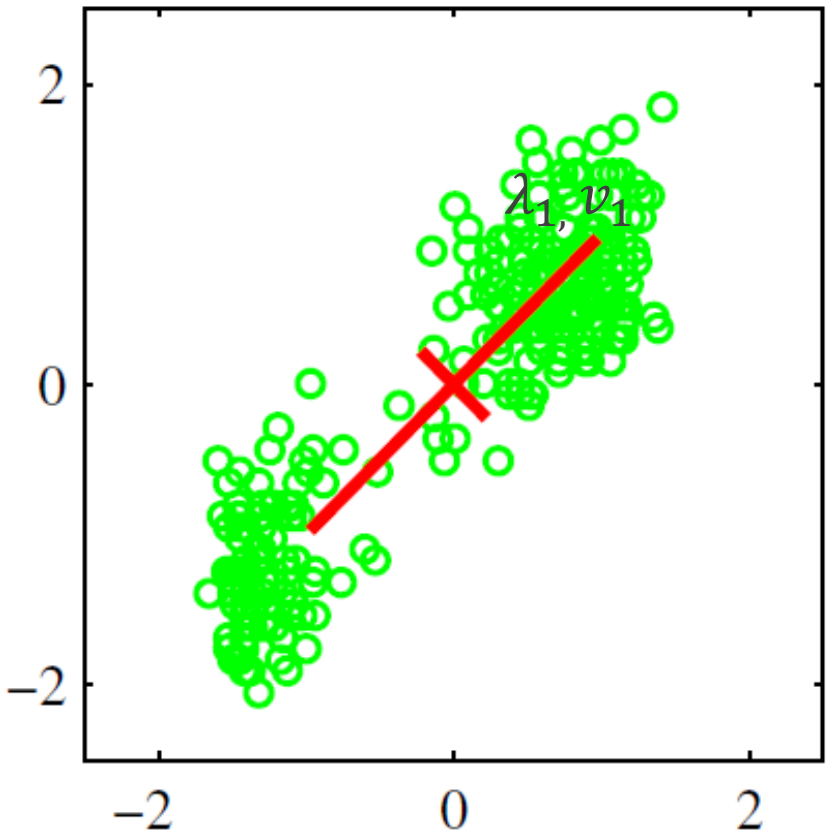


…amount of variance in each principal component decreases with eigenvalue

[Source: Bishop, C.]

Principal Component Analysis (PCA)

How much variance is captured by just the first principal component (i.e. eigenvector with largest eigenvalue)?



Let v_1 be the first principal component, then variance of first PC is,

$$\frac{1}{N}\sum_n \{v_1^T(x_n - \bar{x})\}^2 = v_1^T S v_1 = \lambda_1$$

Why? $= \boldsymbol{\lambda_1 v_1^T v_1 = \lambda_1}$

How much in the second PC?

$$\frac{1}{N}\sum_n \{v_2^T(x_n - \bar{x})\}^2 = v_2^T S v_2 = \lambda_2$$

[Source: Bishop, C.]

PCA: summary

- Step 1: find the data covariance matrix

$$S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^T$$

- Step 2: find S's D (eigenvector, eigenvalue) pairs

$$v_1, \lambda_1, \quad v_2, \lambda_2, \quad \ldots., v_D, \lambda_D$$

v_i's are orthonormal
λ_i's are in decreasing order

(can be done by calling library function scipy.sparse.linalg.eigs)

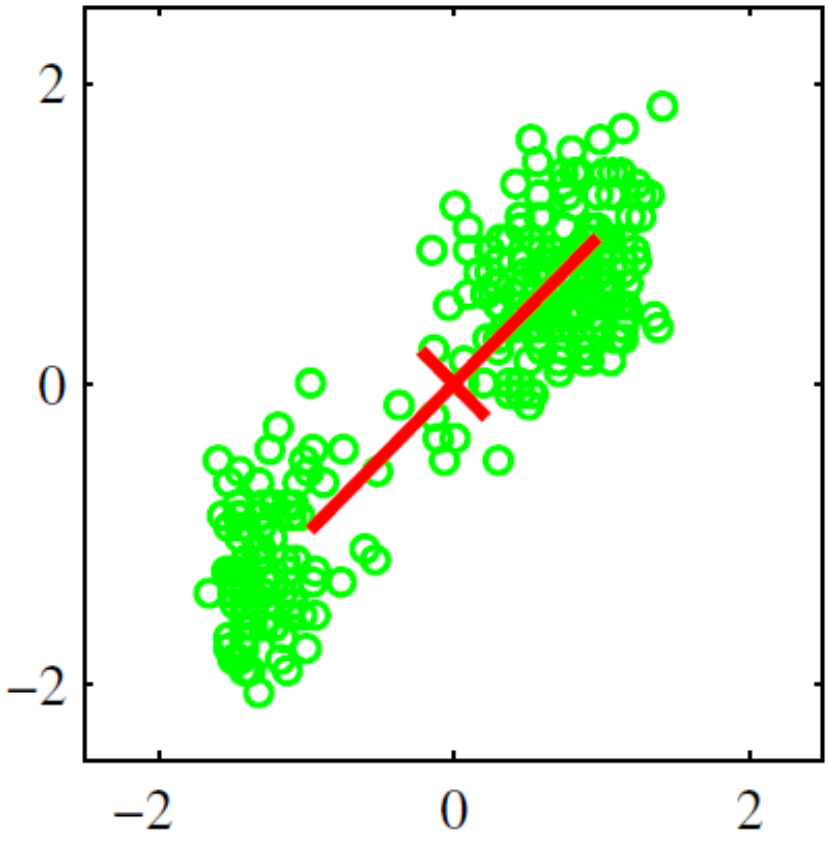
- Step 3: keep the top M eigenvectors $v_1, .., v_M$

- Step 4 (Optional): project original data to M dimensions by mapping every x to

$$(v_1^T x, .., v_M^T x)$$

Explained Variance

How much variance is captured in top M < D principal components?



$$\frac{1}{N}\sum_{m=1}^{M}\sum_{n}\left\{u_m^T(x_n-\bar{x})\right\}^2=\sum_{m=1}^{M}\lambda_m$$

We call this the *explained variance* of the first M principal components

Divide by total variance to find percentage of the total variance explained by the subspace

[Source: Bishop, C.]

sklearn.decomposition.PCA

Parameters

n_components : *int, float or 'mle', default=None*

Number of components to keep. if n_components is not set all components are kept:

copy : *bool, default=True*

If False, data passed to fit are overwritten and running fit(X).transform(X) will not yield the expected results, use fit_transform(X) instead.

whiten : *bool, default=False*

When True (False by default) the `components_` vectors are multiplied by the square root of n_samples and then divided by the singular values to ensure uncorrelated outputs with unit component-wise variances.

sklearn.decomposition.PCA

Attributes

components_ : *ndarray of shape (n_components, n_features)*

Principal axes in feature space, representing the directions of maximum variance in the data. Equivalently, the right singular vectors of the centered input data, parallel to its eigenvectors. The components are sorted by `explained_variance_`.

explained_variance_ : *ndarray of shape (n_components,)*

The amount of variance explained by each of the selected components. The variance estimation uses

explained_variance_ratio_ : *ndarray of shape (n_components,)*

Percentage of variance explained by each of the selected components.

If `n_components` is not set then all components are stored and the sum of the ratios is equal to 1.0.

singular_values_ : *ndarray of shape (n_components,)*

The singular values corresponding to each of the selected components. The singular values are equal to the 2-norms of the `n_components` variables in the lower-dimensional space.

Caution

Careful with the following parameter,

copy : *bool, default=True*

If False, data passed to fit are overwritten and running fit(X).transform(X) will not yield the expected results, use fit_transform(X) instead.

Wrong

```
pca = PCA(n_components=2, copy=False).fit(X)
X_pca = pca.transform(X)
```

X already modified

Right

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

Why would you prefer one over the other?

Right

```
X_pca = PCA(n_components=2, copy=False).fit_transform(X)
```

The first approach keeps pca object which allows to interpret PCs

Example : PCA on Iris Data

Load Iris data without labels,

```
iris = datasets.load_iris(as_frame=True)
X = iris.data
```

Find PCA with 2 principal components,

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

How much variance did we capture?

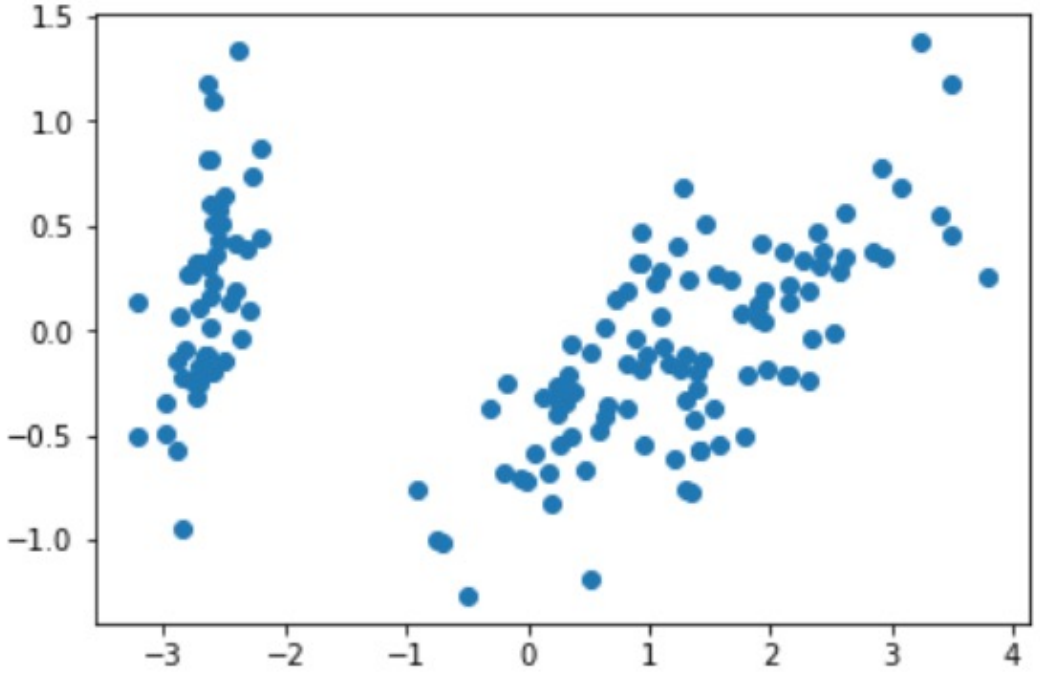
```
expvar = pca.explained_variance_ratio_
print('% Variance in 1st PC: ', expvar[0])
print('% Variance in 2nd PC: ', expvar[1])
print('Total explained variance: ', sum(expvar))
```

```
% Variance in 1st PC:  0.9246187232017271
% Variance in 2nd PC:  0.05306648311706780 4
Total explained variance:  0.977685206318795
```

Example : PCA on Iris Data

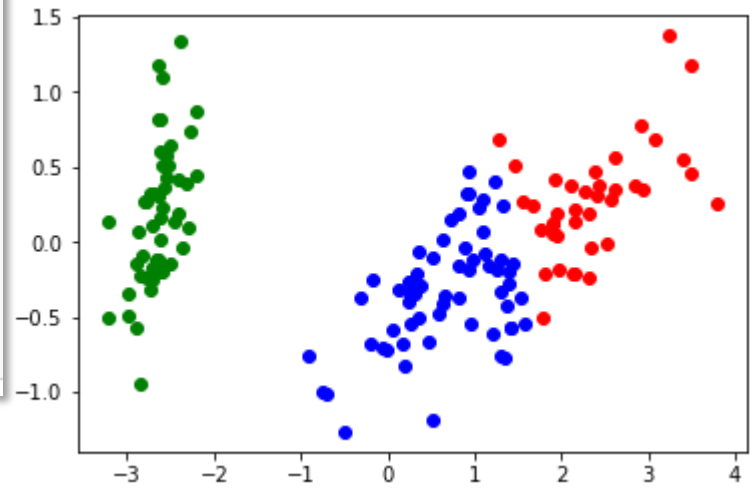
Project data to 2-D subspace and visualize,

```
plt.scatter(X_pca[:,0], X_pca[:,1])
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```



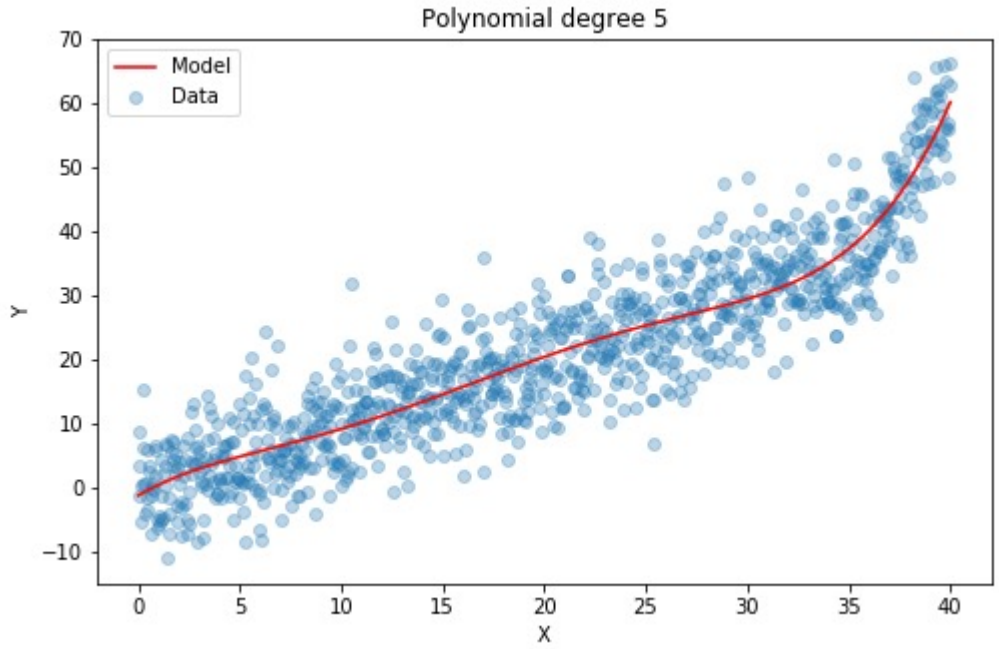
Do K-means clustering in 2-D subspace,

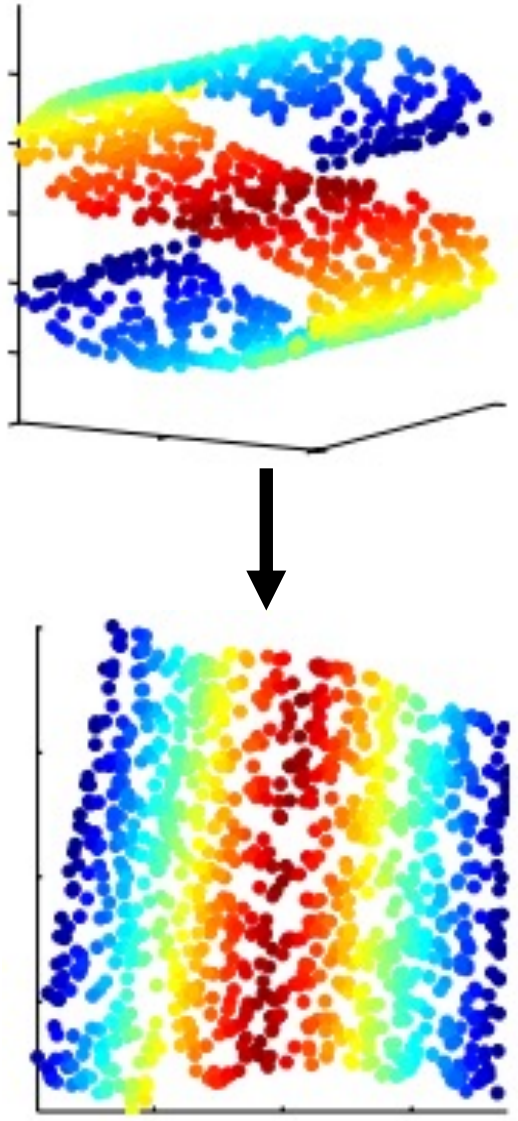
```
kmeans = KMeans(n_clusters=3).fit(X_pca)
labels = kmeans.labels_
fig, ax = plt.subplots()
ax.scatter(X_pca[:,0][labels == 0], X_pca[:,1][labels == 0], c='r')
ax.scatter(X_pca[:,0][labels == 1], X_pca[:,1][labels == 1], c='g')
ax.scatter(X_pca[:,0][labels == 2], X_pca[:,1][labels == 2], c='b')
plt.show()
```



Nonlinear Dimensionality Reduction

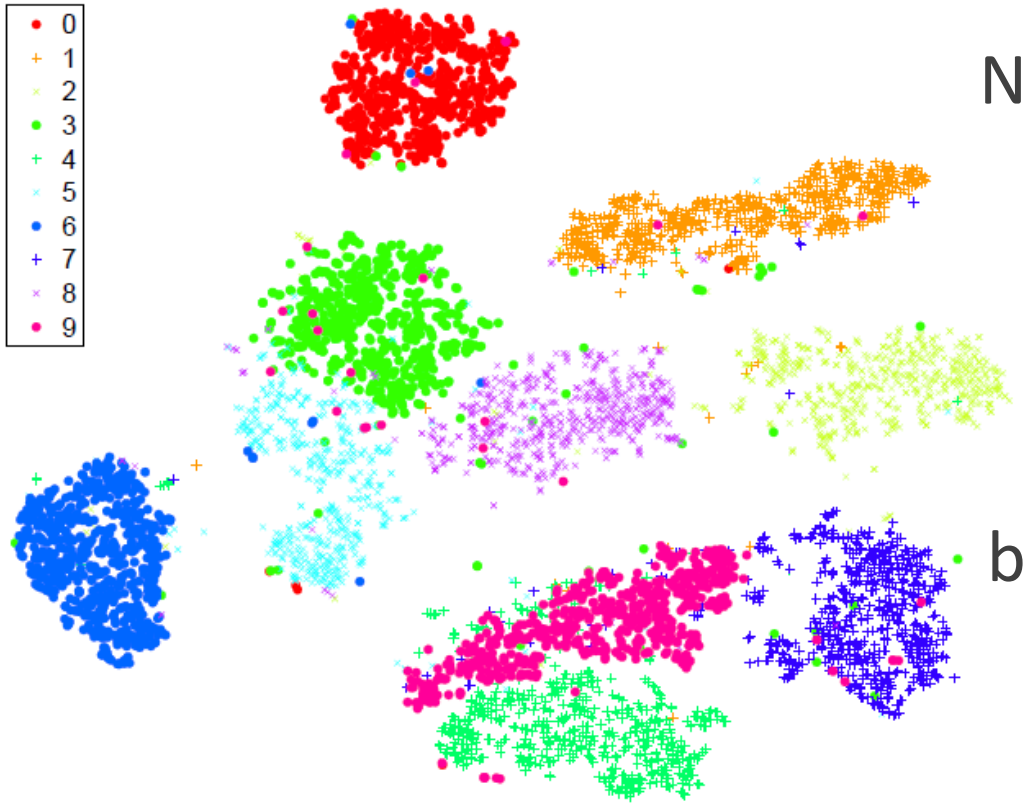
For general data, linear dimensionality reduction is not sufficient…





Many methods exist for nonlinear dimensionality reduction

t-SNE



Nonlinear reduction can (potentially) amplify clustering properties

t-Distributed Stochastic Neighbor Embedding (t-SNE) Models similarity between data as a Student's-t distribution in high / low dimensions and optimizes reduction to preserve similarity

Visualization shows MNIST digits projected to 2D

sklearn.manifold.TSNE

Parameters

n_components : *int, default=2*
Dimension of the embedded space.

perplexity : *float, default=30.0*
The perplexity is related to the number of nearest neighbors that is used in other manifold learning algorithms. Larger datasets usually require a larger perplexity. Consider selecting a value between 5 and 50. Different values can result in significantly different results.

learning_rate : *float or 'auto', default=200.0*
The learning rate for t-SNE is usually in the range [10.0, 1000.0].

Attributes

embedding_ : *array-like of shape (n_samples, n_components)*
Stores the embedding vectors.

Example : t-SNE on Iris Dataset

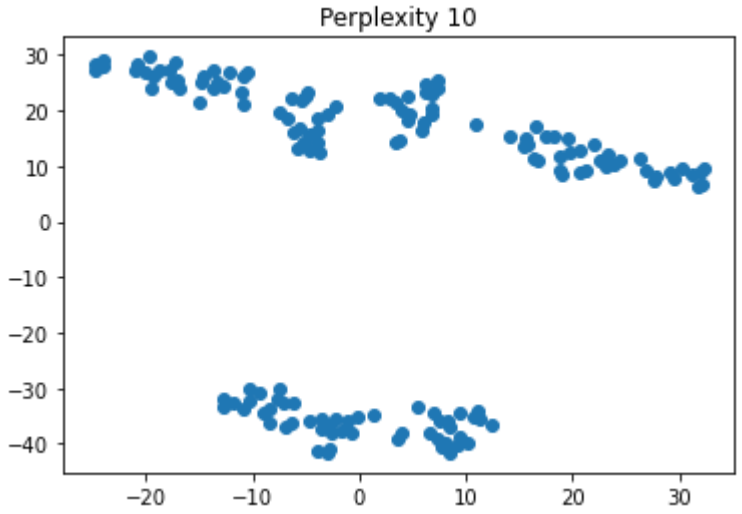
t-SNE can work surprisingly well…

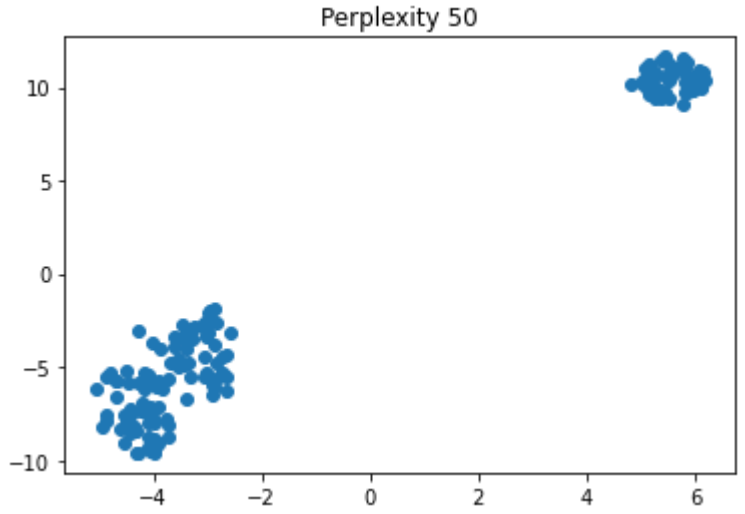
```
from sklearn.manifold import TSNE
perplexity = [20, 50, 100]
for perp in perplexity:
    tsne = TSNE(n_components=2, perplexity=perp)
    X_tsne = tsne.fit_transform(X)
    fig, ax = plt.subplots()
    ax.scatter(X_tsne[:,0], X_tsne[:,1])
    ax.set_title('Perplexity %i' % perp)
plt.show()
```

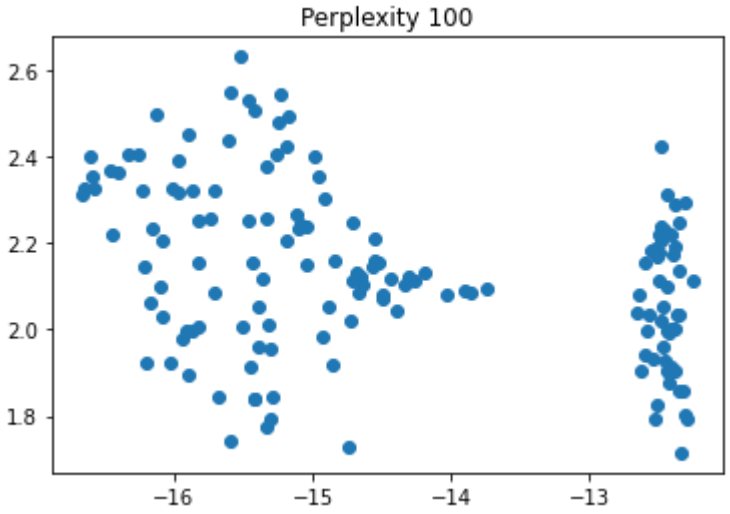
One advantage of PCA is that it has no parameters that need tuning (aside from number of PCs)

PCA is also much easier to interpret

…but can be a bit fussy about parameters and unreliable







Autoencoders

- Suppose you have a number in {0,1,2,3,4,5,6,7}
- What would be a compact representation that incurs small memory cost?

- Q: how many bits do we need?

Early Observations

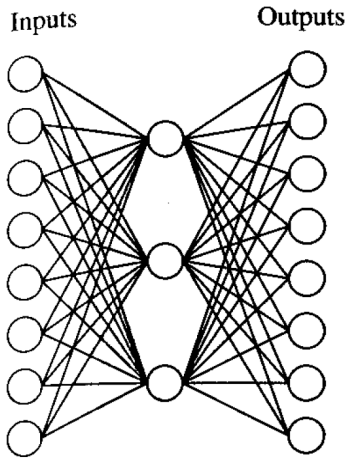
Train a pair of (encoder network E, decoder network D) such that:

- *We force E to reduce dimension*

- D(E(x)) recovers x

- imposing squared loss on all the output units & backpropagation.

Q: What do the hidden values (codes) E(x) look like?



Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

FIGURE 4.7
Learned Hidden Layer Representation. This $8 \times 3 \times 8$ network was trained to learn the identity function, using the eight training examples shown. After 5000 training epochs, the three hidden unit values encode the eight distinct inputs using the encoding shown on the right. Notice if the encoded values are rounded to zero or one, the result is the standard binary encoding for eight distinct values.

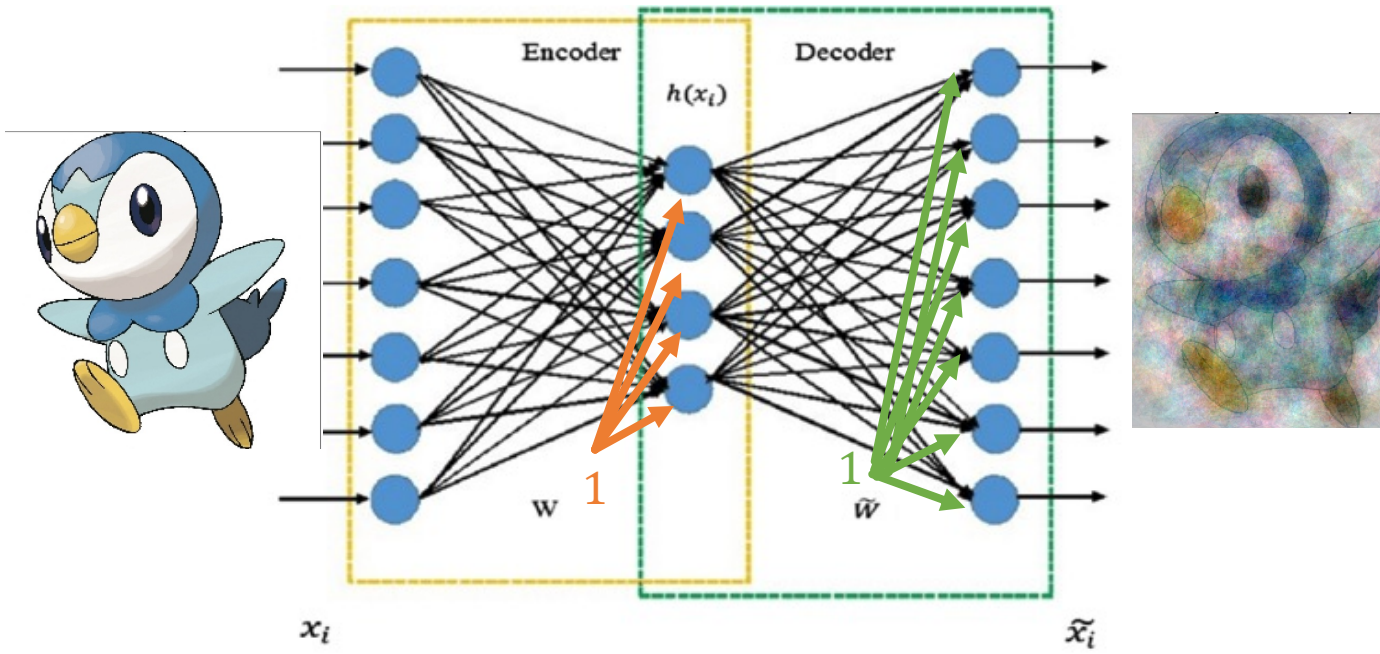
p107, Tom Mitchell, “Machine Learning”

PCA as a linear autoencoder

- The PCA can be represented as an autoencoder

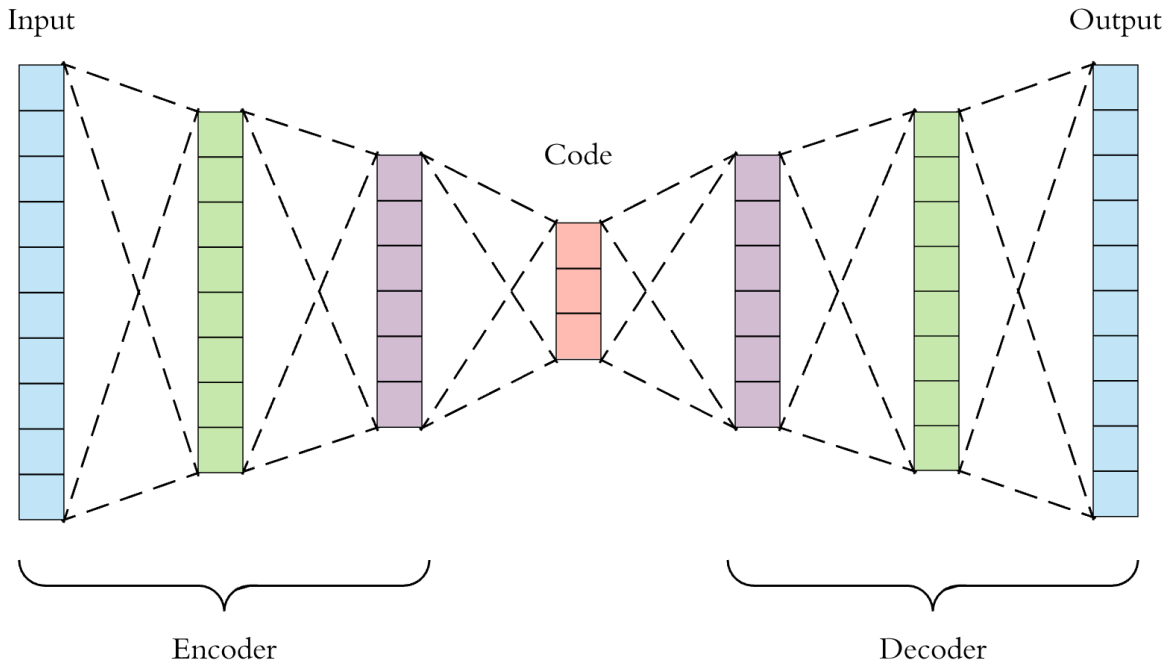
with k units in the hidden layer, constant bias added in each layer):

- Encoder: $h = \begin{pmatrix} - & v_1 & - \\ & \cdots & \\ - & v_k & - \end{pmatrix} \cdot x + \begin{pmatrix} -v_1^\top \mu \\ \cdots \\ -v_k^\top \mu \end{pmatrix}$
- Decoder: $\tilde{x} = \begin{pmatrix} | & & | \\ v_1 & \cdots & v_k \\ | & & | \end{pmatrix} \cdot h + \begin{pmatrix} | \\ \mu \\ | \end{pmatrix}$



Training autoencoders

- Given:
  - data $x_1, \dots, x_n \in \mathbb{R}^d$,
  - Embedding dimension k ($k \ll d$)



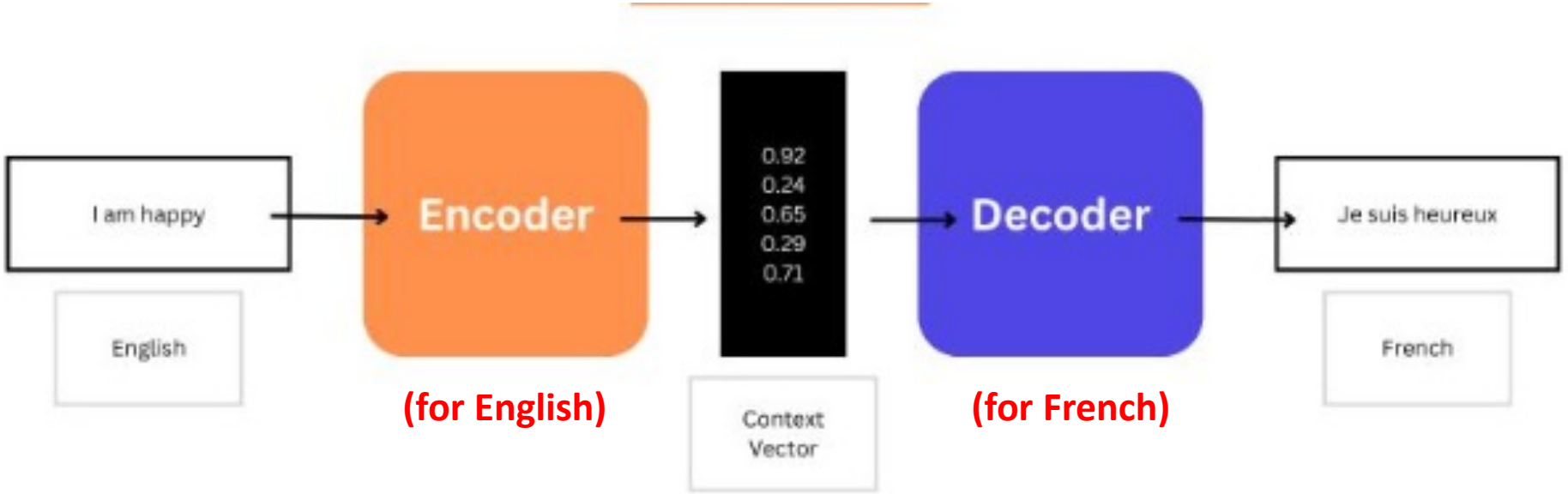
- Goal: obtain
  - Encoder network $f_\theta: \mathbb{R}^d \to \mathbb{R}^k$
  - Decoder network $g_\phi: \mathbb{R}^k \to \mathbb{R}^d$
  - Such that for every i, $x_i \approx g_\phi(f_\theta(x_i))$

- Most commonly used formulation (can be straightforwardly trained by gradient-based methods):

$$\text{minimize}_{\theta,\phi} \sum_{i=1}^{n} \underbrace{\left\| x_i - g_\phi\left(f_\theta(x_i)\right) \right\|^2}_{\text{Reconstruction error}}$$

The Encoder-decoder architecture

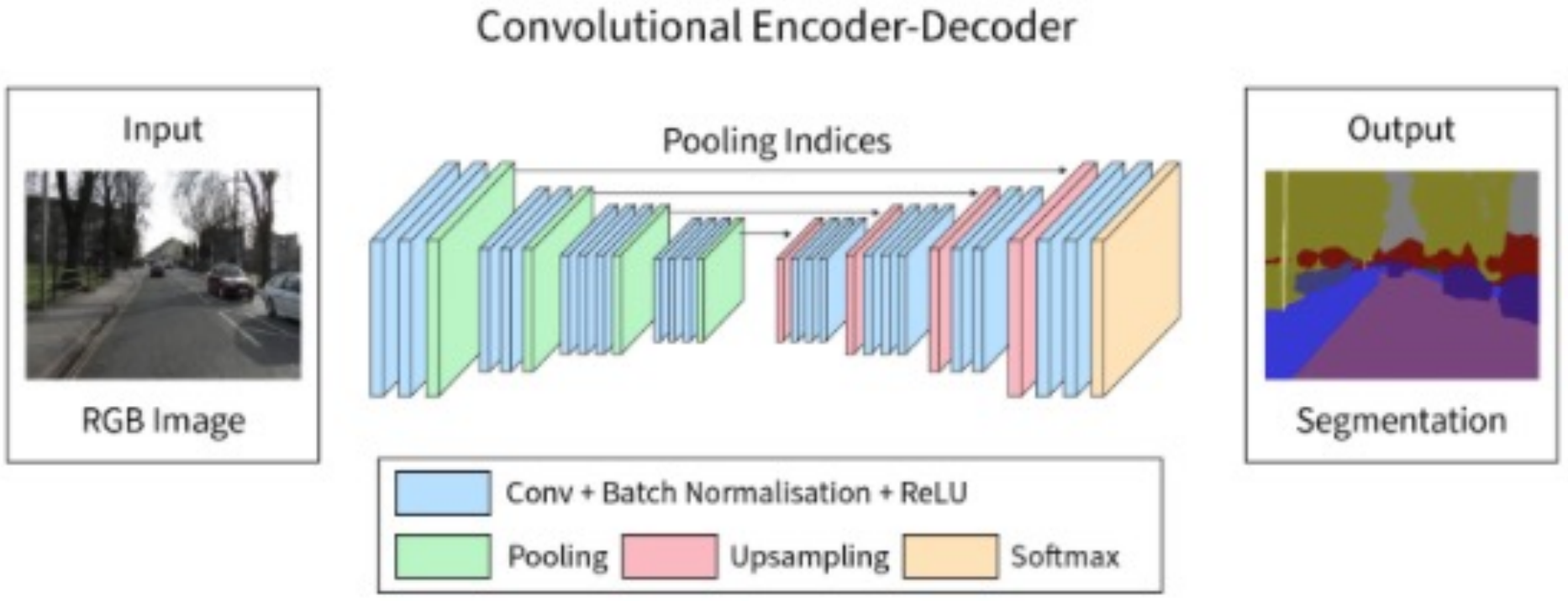
- Encoder-decoder are now standard in deep learning systems
- Encoder: “compress” input text / image to some low dimensional, latent semantic representation



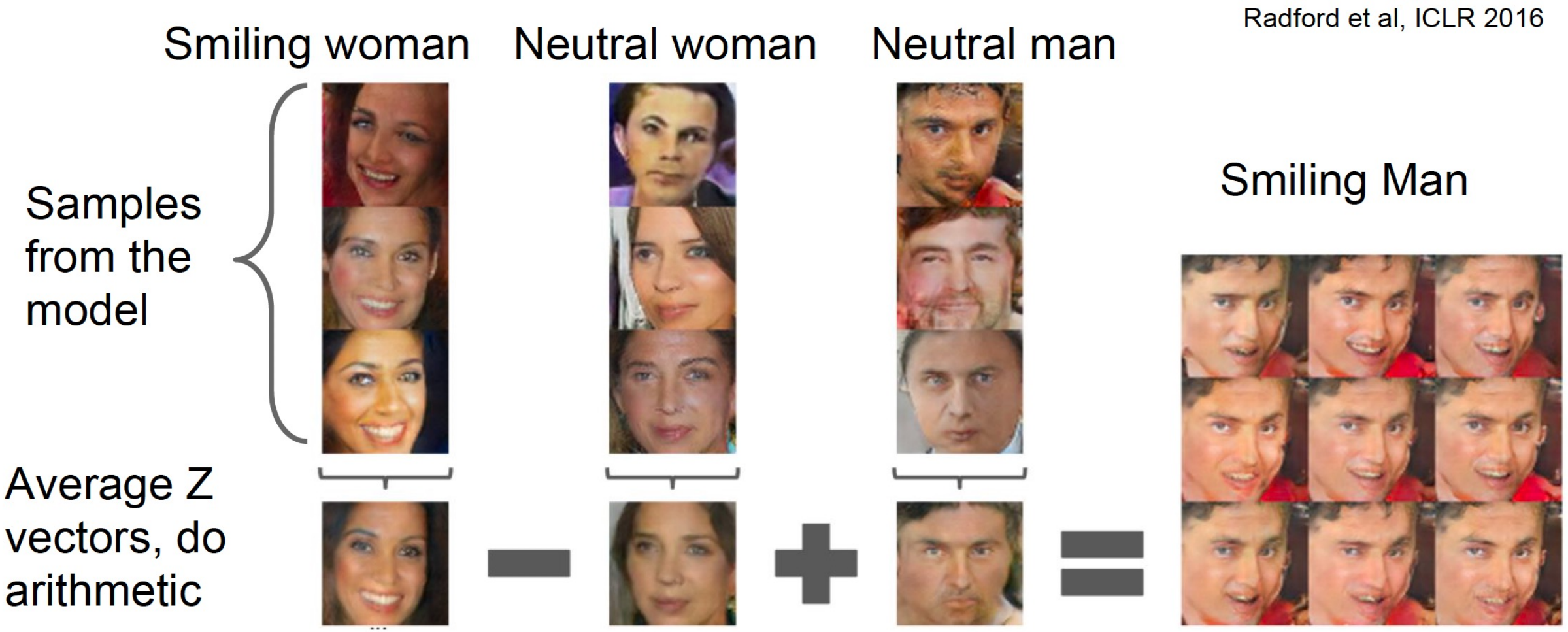
- Decoder: “decompress” the latent representation to output text / image

The Encoder-decoder architecture

- Can also be used for image segmentation, style transfer, etc



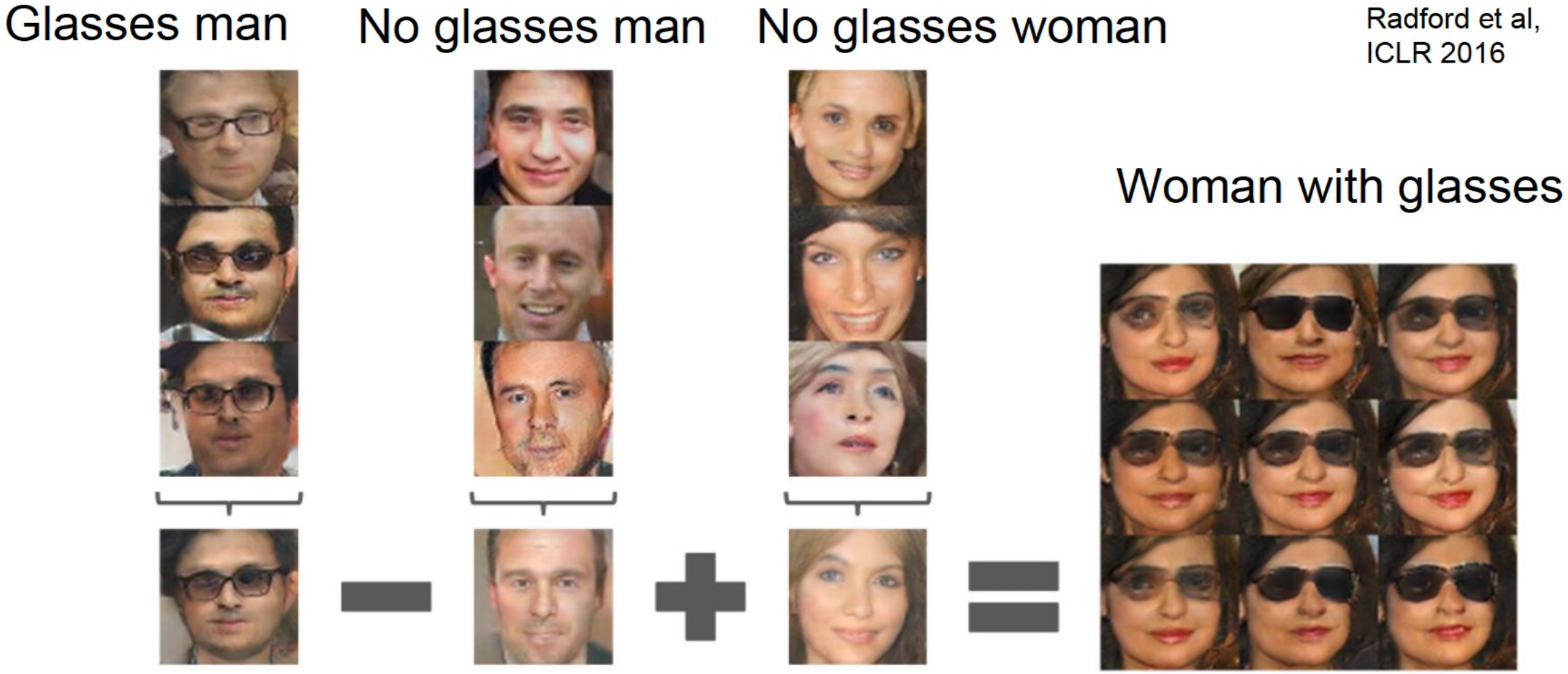
Learned ‘code’ is interpretable



- This means that there are ‘directions’ in the latent code z that have particular meanings!

(slide from Stanford CS231n)

Learned ‘code’ is interpretable



Component Analysis: closing remarks

- Nonlinear methods in Scikit-Learn categorized under “manifold learning” in the *manifold* sub-package,
  - Isomap, Locally Linear Embedding, Spectral Embedding, Multidimensional scaling, and of course t-SNE

- For multiple data sources, consider cross-decomposition
  - Canonical Correlation Analysis (CCA)
  - Learns same embedding for both spaces
  - Under *cross_decomposition* sub-package

Machine Learning: the frontier

- How can we improve foundation models?
  - Multimodal models: text + image + audio
  - How can we reduce hallucination of LLMs?
  - Can models reliably reason? See this talk
- Reinforcement learning (welcome to take CSC 496 next Fall ☺)
  - How can we build robots that can conduct tasks reliably and harmlessly? E.g. lightwheel.ai
  - How can LLMs learn from (human) verification signals?
  - How can agents find the *right* data the learn from, not just more data? Active exploration
- How can machine learning accelerate scientific discovery? E.g. Autonomous Lab
  - Can LLMs find new causal relationship?

The world of machine learning welcomes your involvement!

