



Computer
Science

CSC380: Principles of Data Science

Dimensionality Reduction

Prof. Jason Pacheco

TA: Enfa Rose George

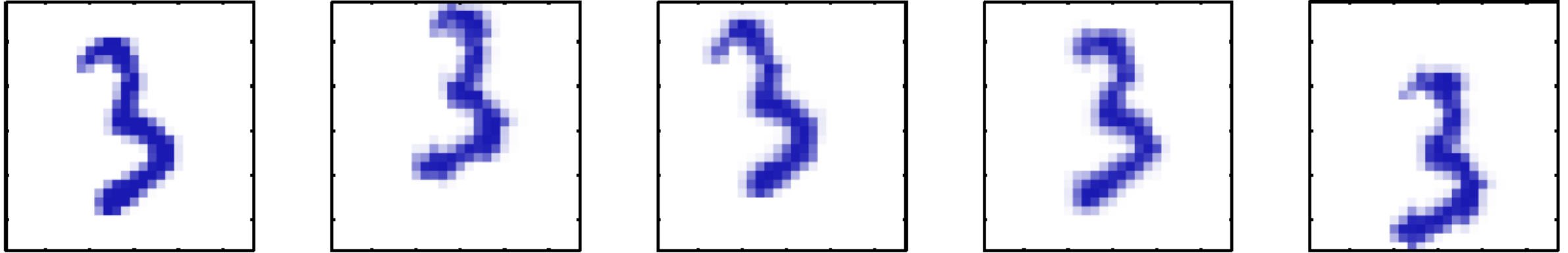
TA: Saiful Islam Salim

Administrative Items

- HW9 Due next Tue (12/7)
- Final Exam
 - “Take Home”
 - Will go out next Thurs (12/9)
 - Due following Wed (12/15) our official Final day
 - It will not take that full time (I know you have other finals)

Motivation

Data often have a lot of redundant information...



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

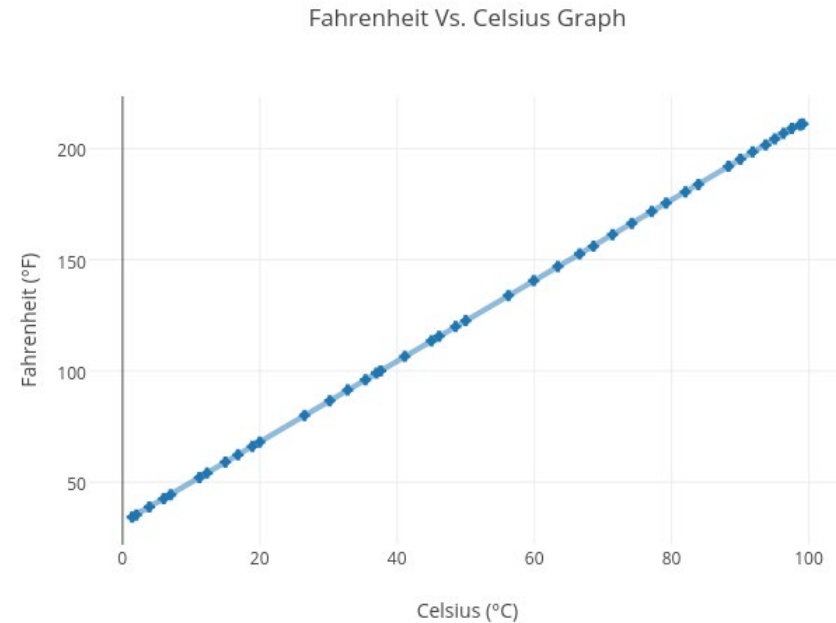
Data Dimension $100 \times 100 = 10,000$

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

Motivation

...or data 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
...	...

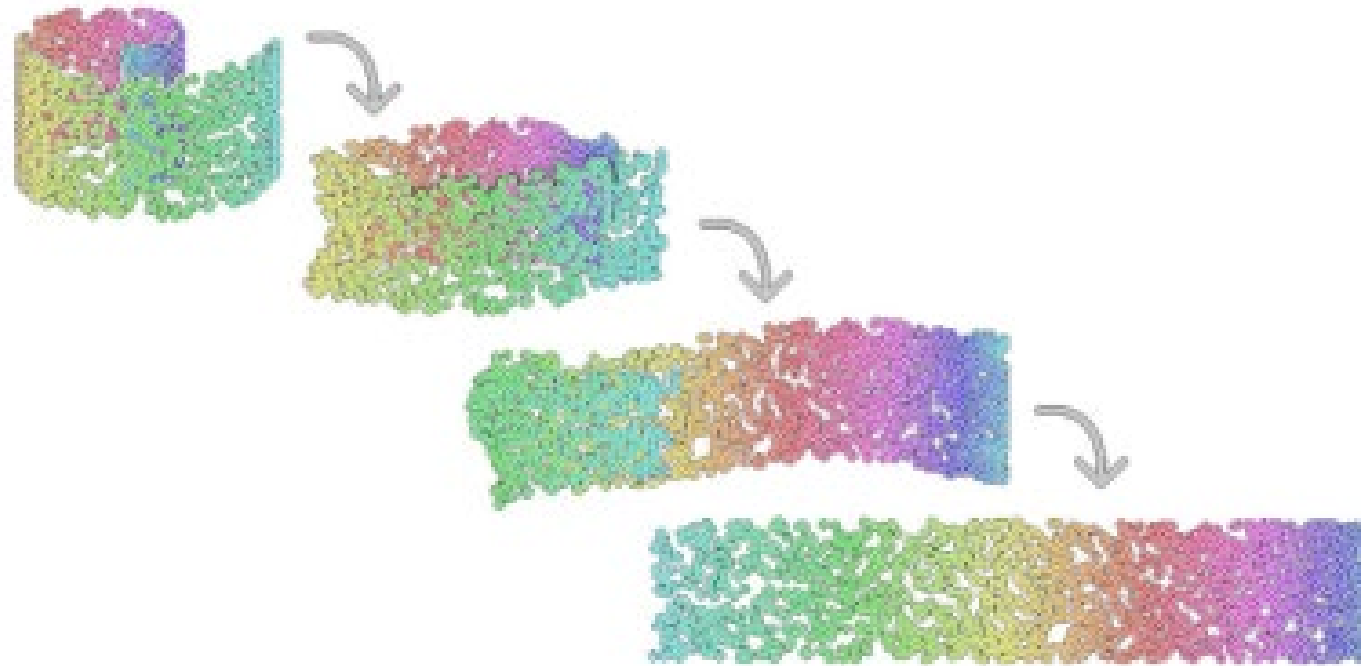


Linear Function

$$F = 1.8C + 32$$

Motivation

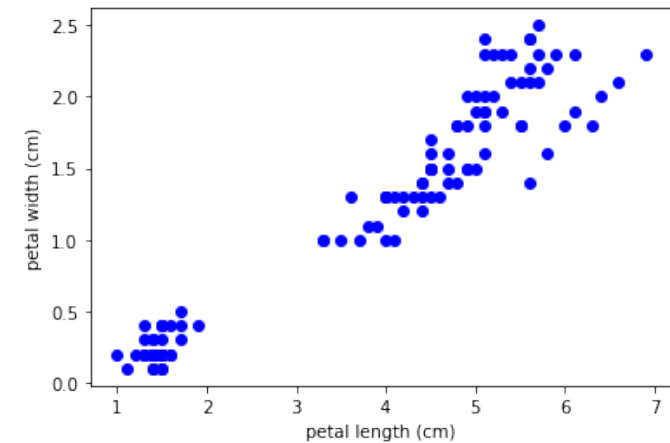
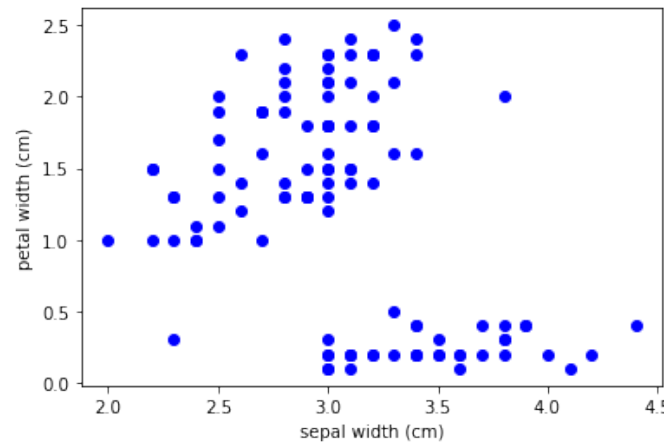
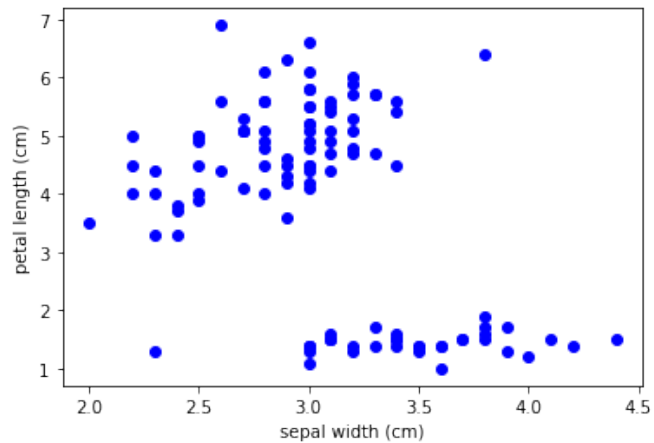
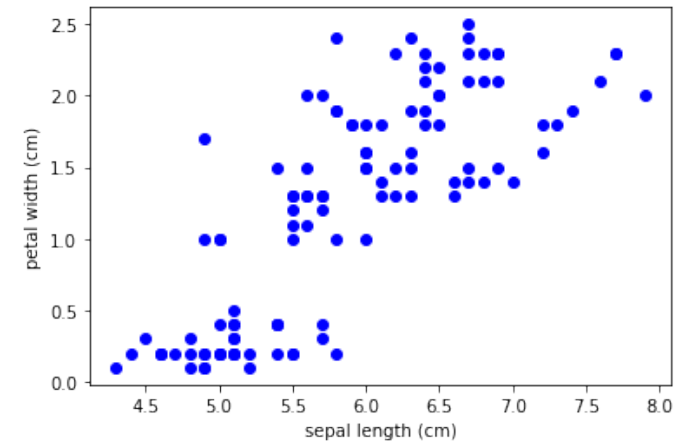
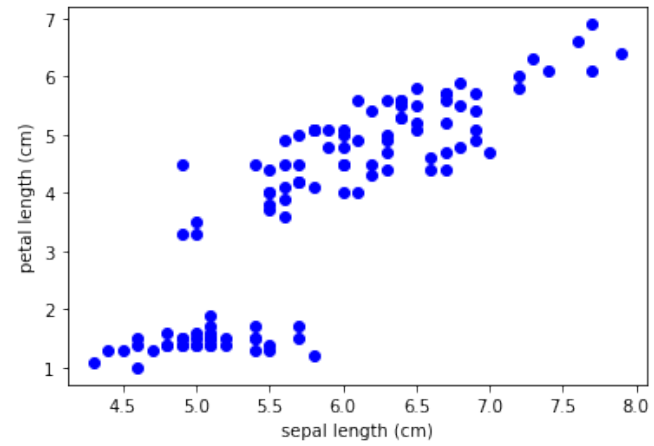
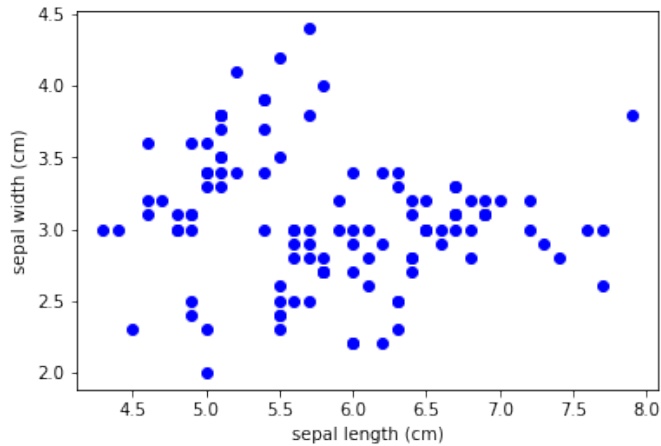
...or data are high-dimensional and hard to visualize...



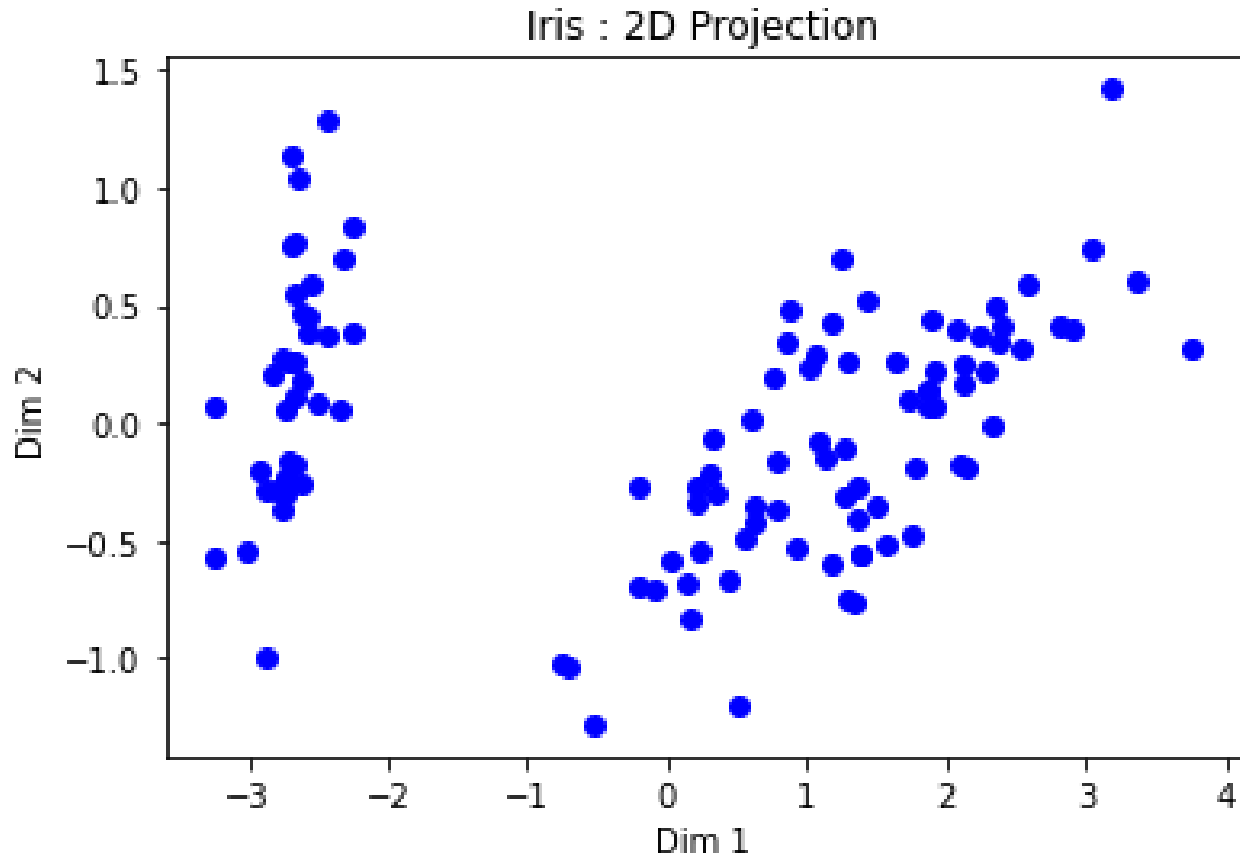
...in all cases finding lower *intrinsic dimension* is useful

Example : Iris Dataset

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



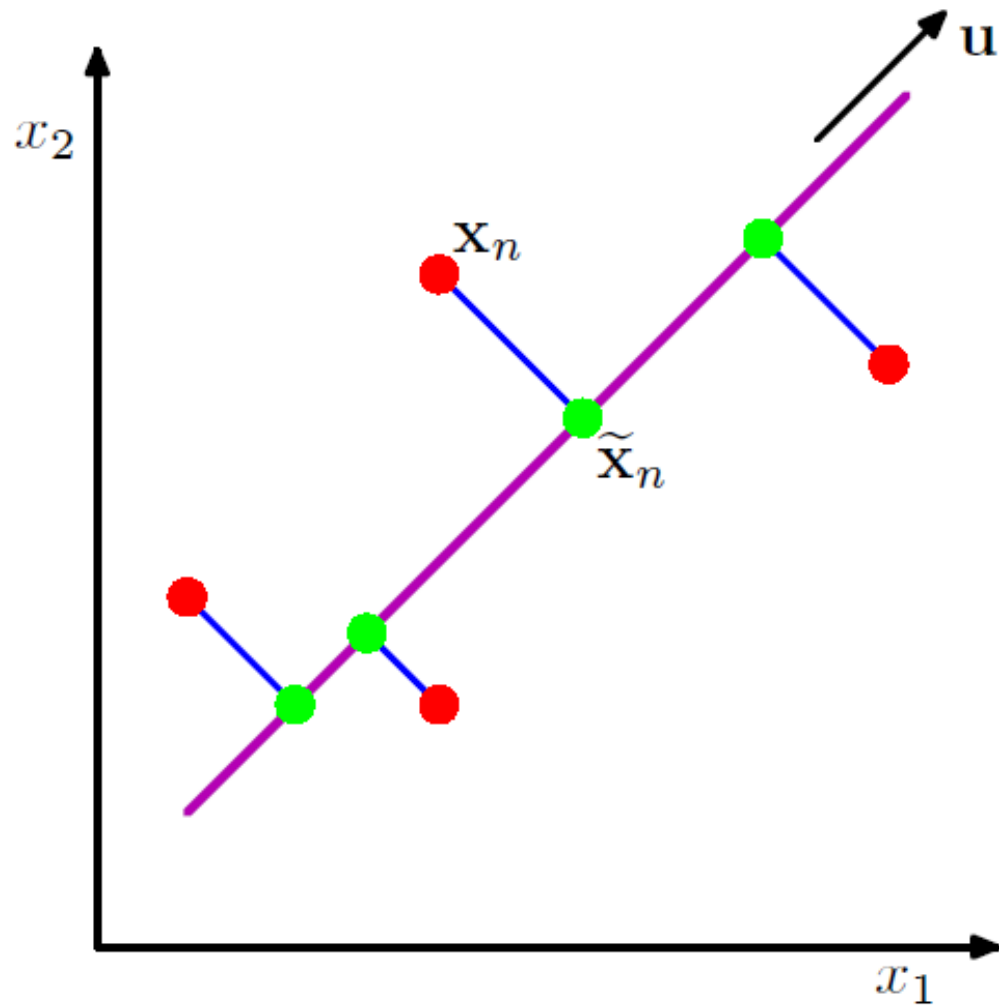
Example : Iris Dataset



Data still cluster in a two-dimensional subspace

We can fit model 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...

Inner Products

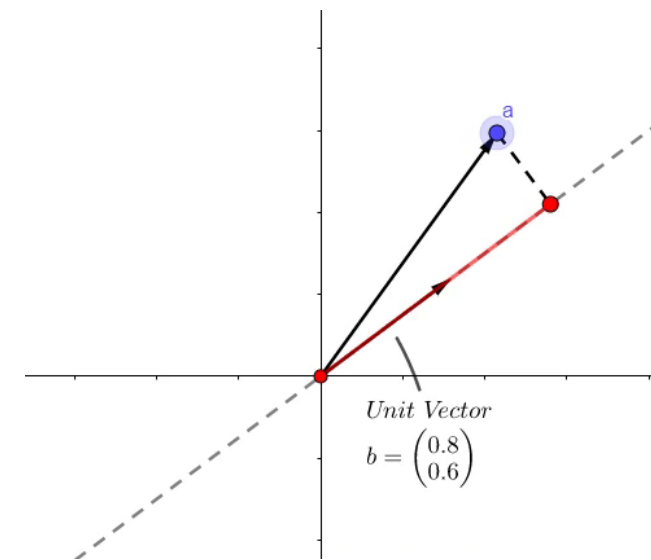
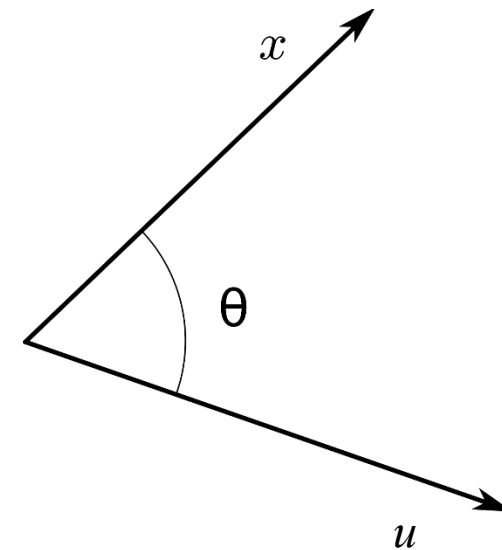
Recall the definition of an *inner product*:

$$\begin{aligned} u^T x &= u_1 x_1 + u_2 x_2 + \dots + u_D x_D \\ &= \sum_{d=1}^D u_d x_d \end{aligned}$$

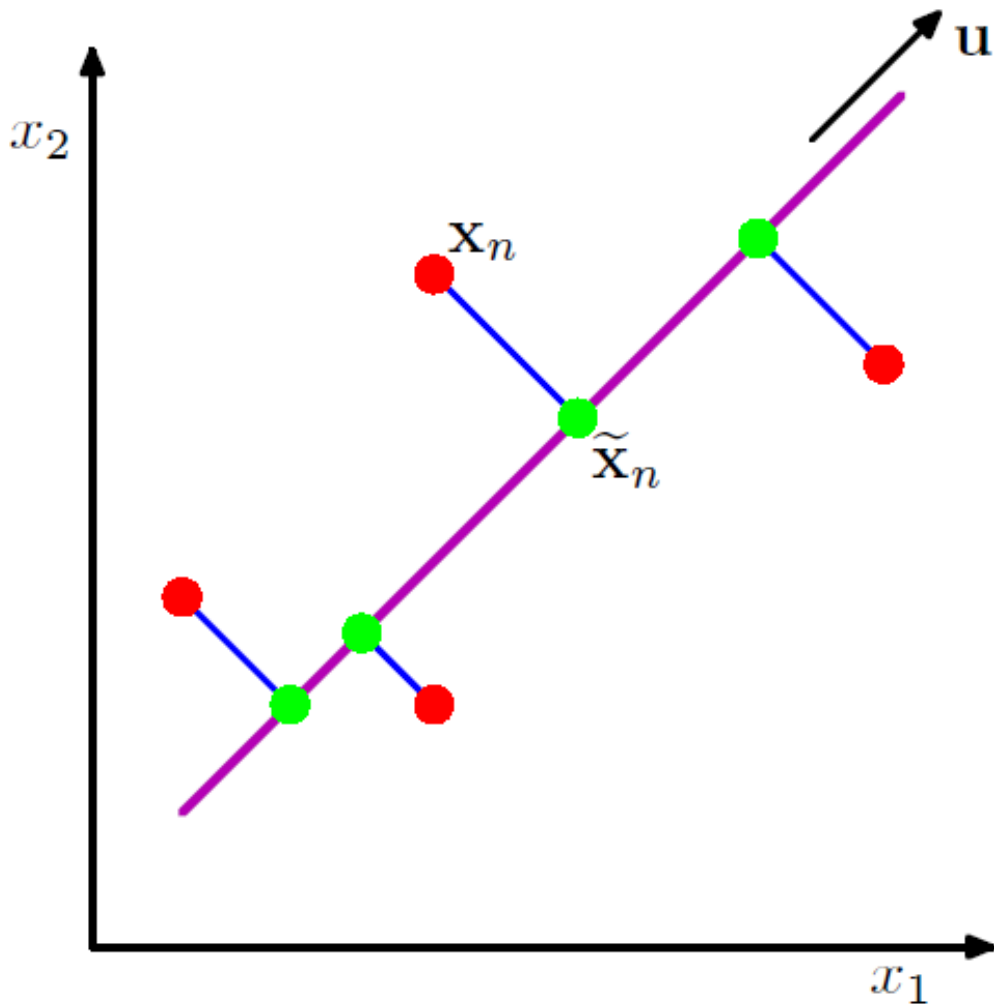
Equivalently, projection of one vector onto another,

$$u^T x = |u| |x| \cos \theta \quad \text{where} \quad |x| = \sqrt{\sum_d x_d^2}$$

Vector Norm



Linear Dimensionality Reduction



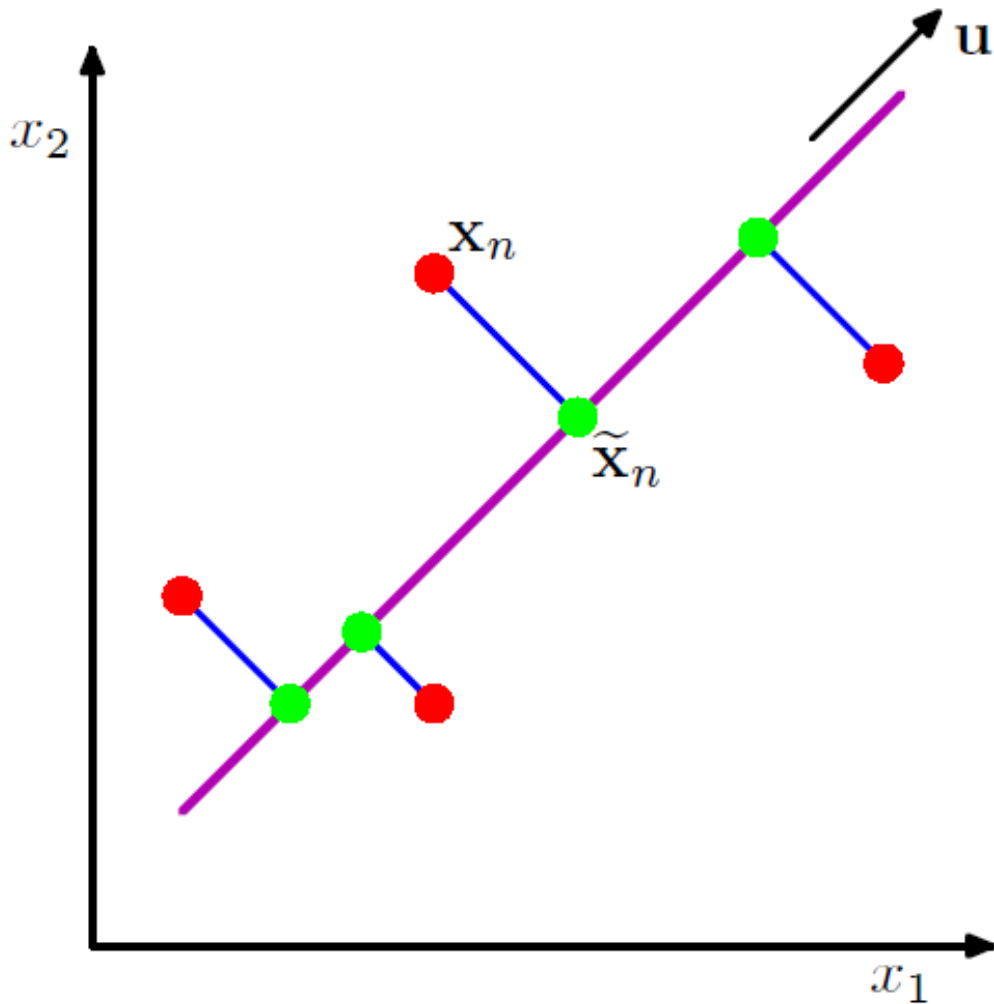
Projecting data onto a vector is a simple inner product,

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

*We call u the **linear subspace***

Question Why would dimensionality reduction be better than feature selection (e.g. choose 1-D features X_1 or X_2)?

Linear Dimensionality Reduction



Projecting data onto a vector is a simple inner product,

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

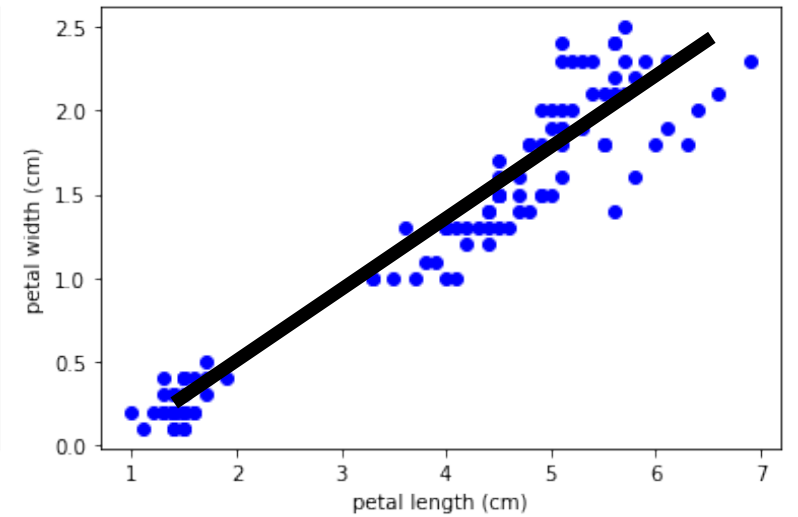
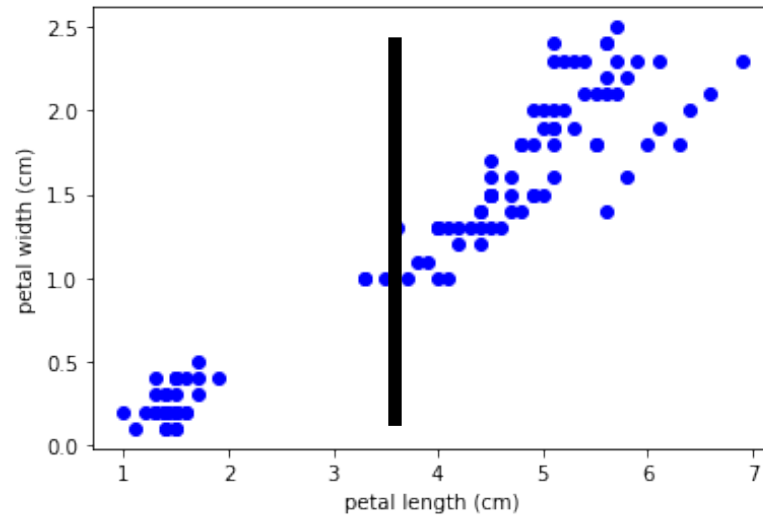
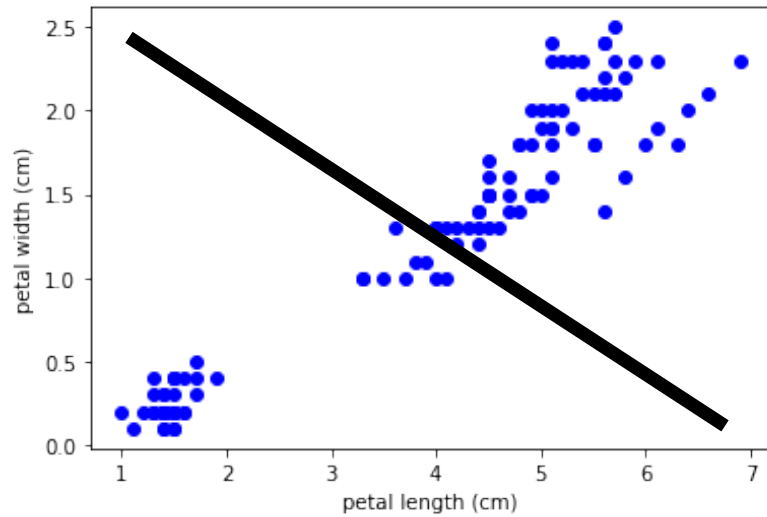
*We call u the **linear subspace***

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

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

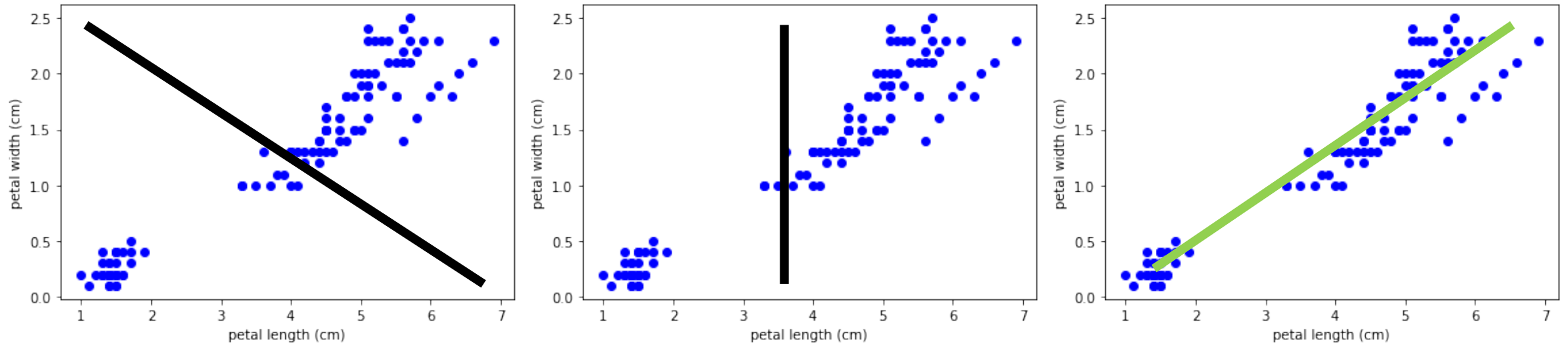
Linear Dimensionality Reduction

Which choice of subspace is best? And why?



Linear Dimensionality Reduction

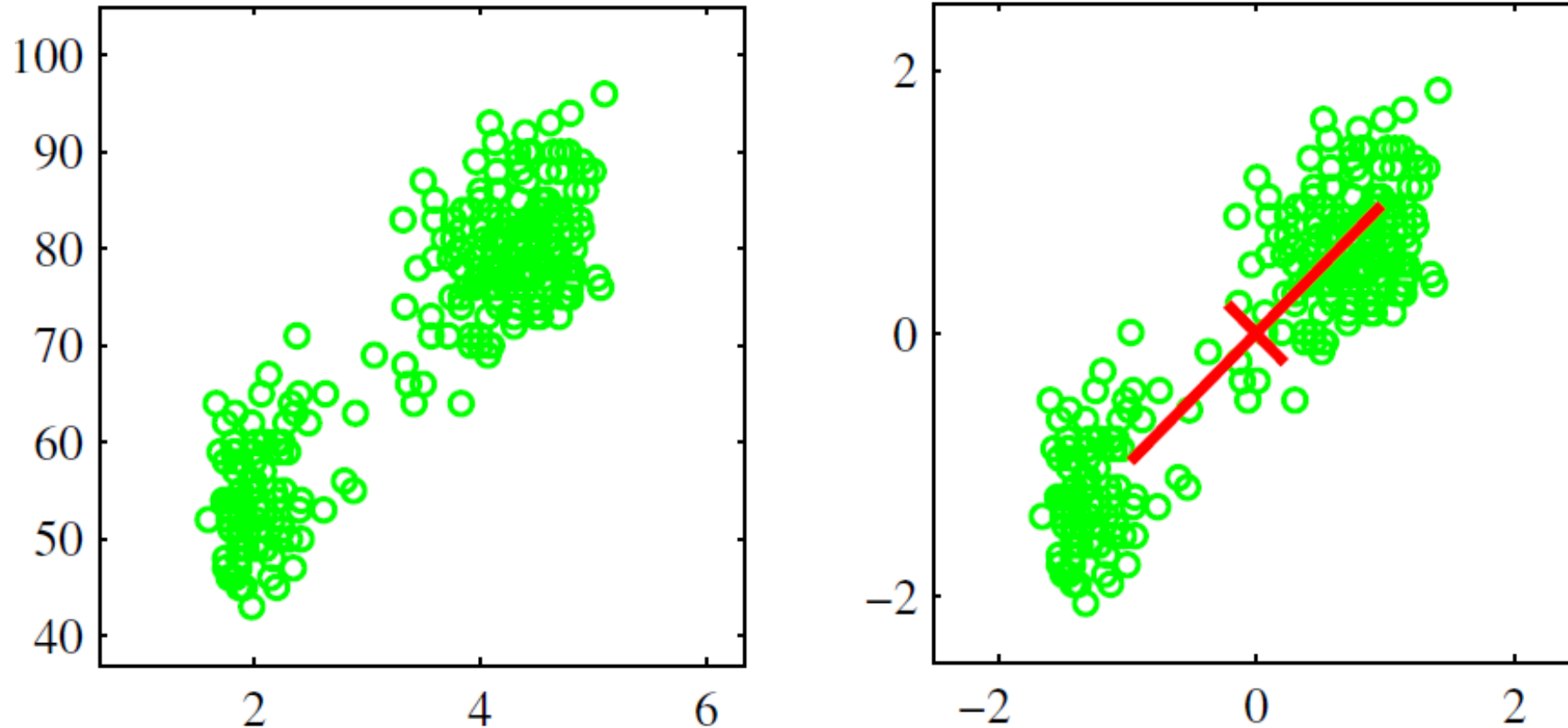
Which choice of subspace is best? And why?



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*

Principal Component Analysis (PCA)

First, center the data by subtracting the sample mean,

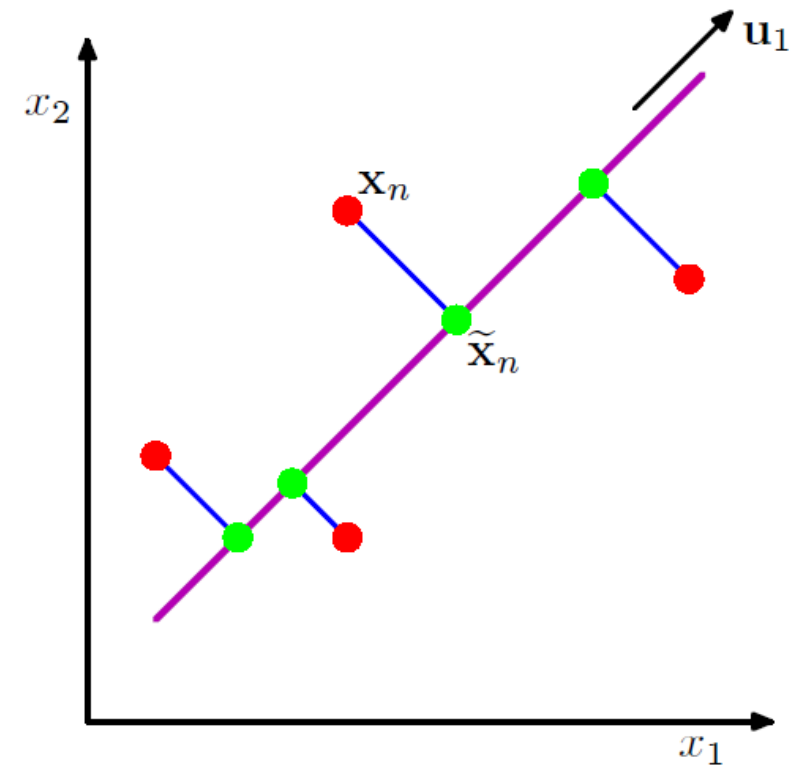
$$\bar{x} = \frac{1}{N} \sum_{n=1}^N x_n$$

Variance of projected subspace,

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

Projection of
nth data point

Projection of
mean




Minimum Variance Formulation

A little algebra...

$$\frac{1}{N} \sum_{n=1}^N (u^T x_n - u^T \bar{x})^2 = \frac{1}{N} \sum_{n=1}^N \{u^T (x_n - \bar{x})\}^2 \quad \text{Pull out } u$$

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

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

Then: $\frac{1}{N} \sum_{n=1}^N (u^T x_n - u^T \bar{x})^2 = u^T S u$  **This is what we will optimize over u**

Minimum 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...

$$S u - \lambda u = 0$$

$$S u = \lambda u$$

For those that have taken linear algebra: What equation is this?

u is an *eigenvector* with *eigenvalue* λ

Recap of Concepts

- Learning a reduced *intrinsic dimension* is useful for a bunch of reasons
- The easiest approach is to find a *linear subspace*
- PCA defines the linear subspace as that which maximizes variance of the projected data

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

- The set of subspaces are defined by the *eigenvectors*,

$$S u = \lambda u$$

But what is an eigenvector?

Linear Transformations

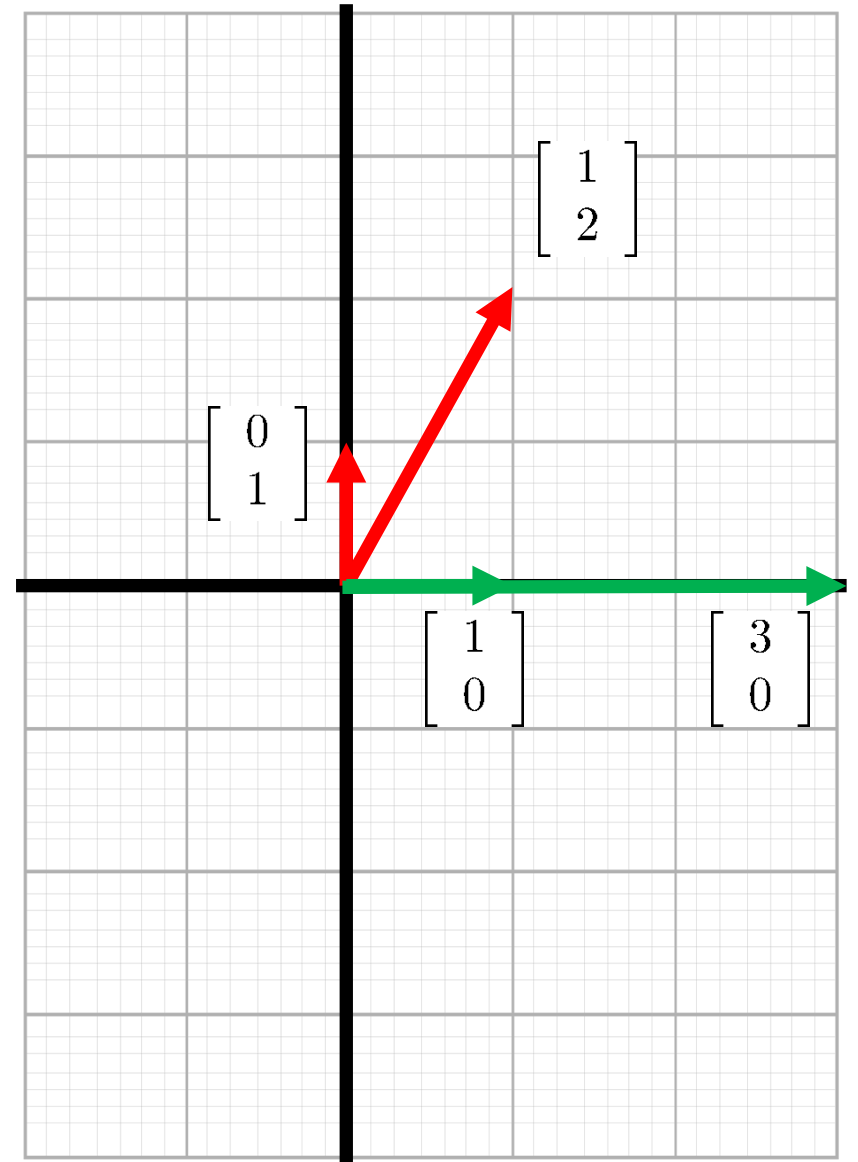
Consider the matrix: $\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix}$

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



Eigenstuff

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,

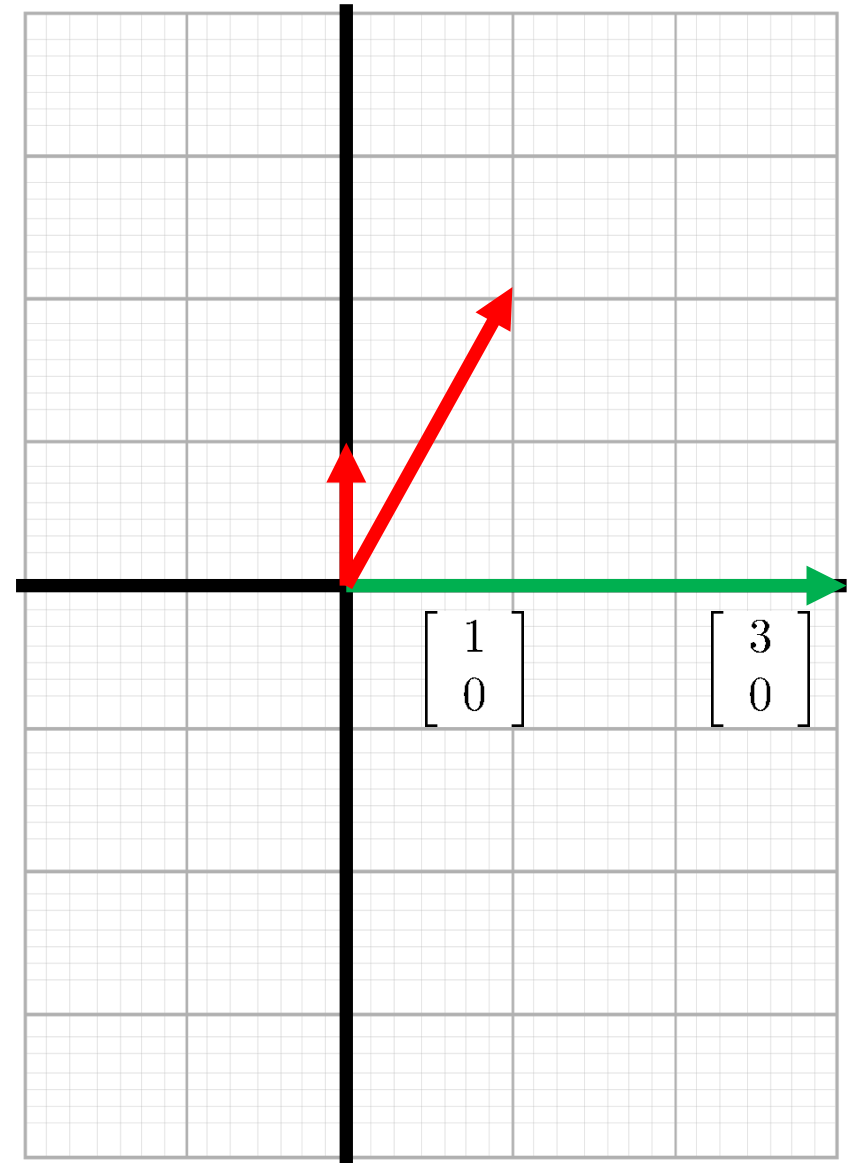
$$\begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

\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



Eigenstuff

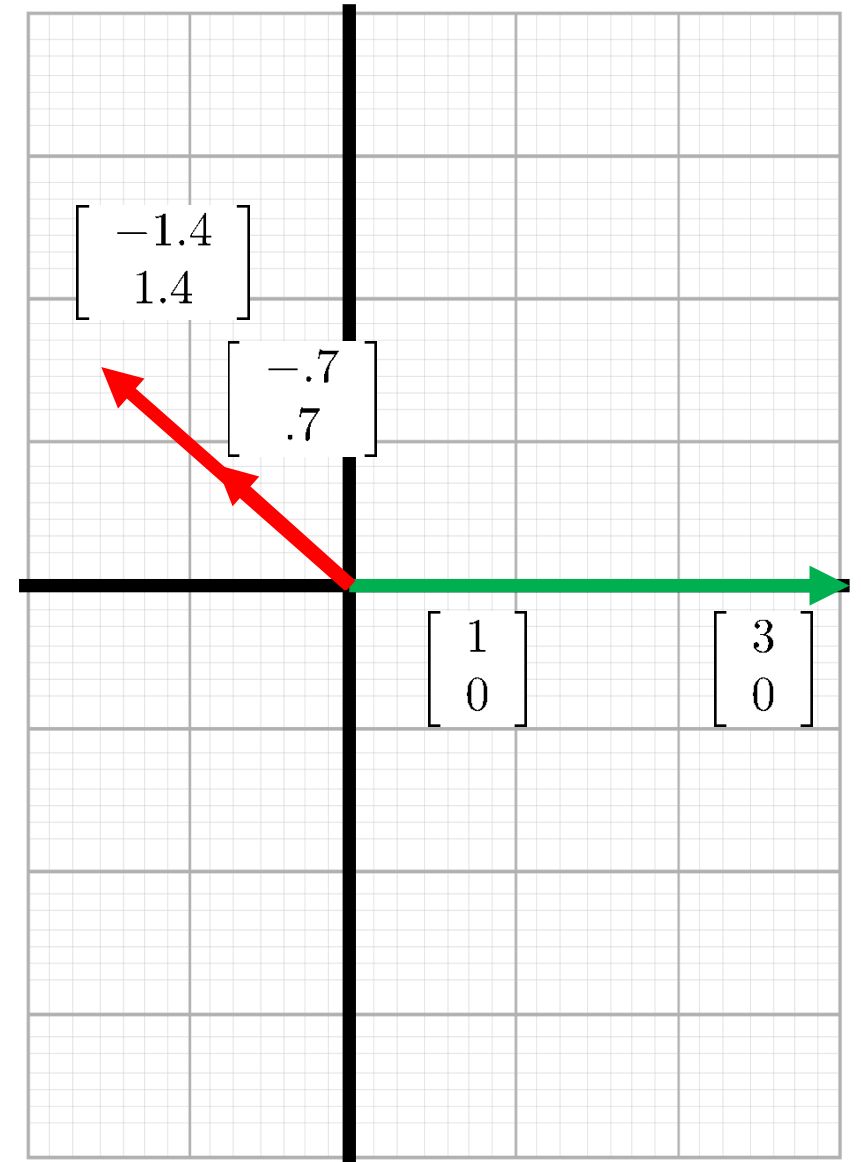
Transformation has one other eigenvector,

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

- Complete eigendecomposition of S

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

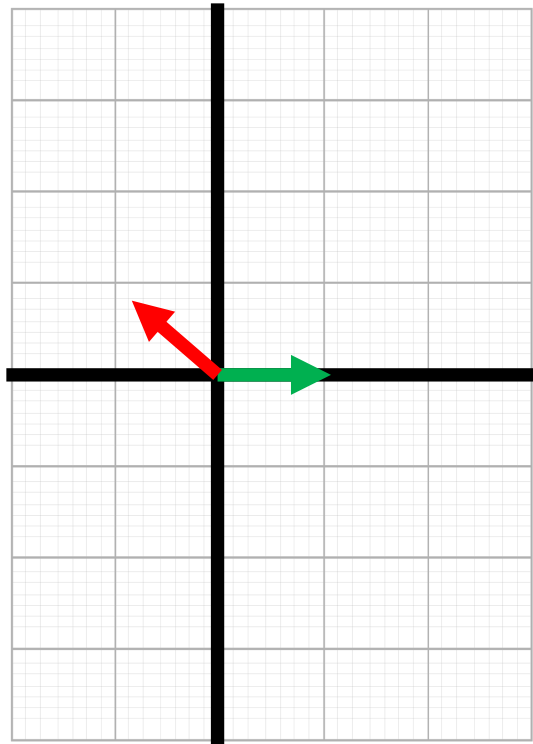
- Eigenvectors of linear transformation S are only stretched / shrunk / flipped
- Eigenvalues tell how much they are stretched / shrunk / flipped



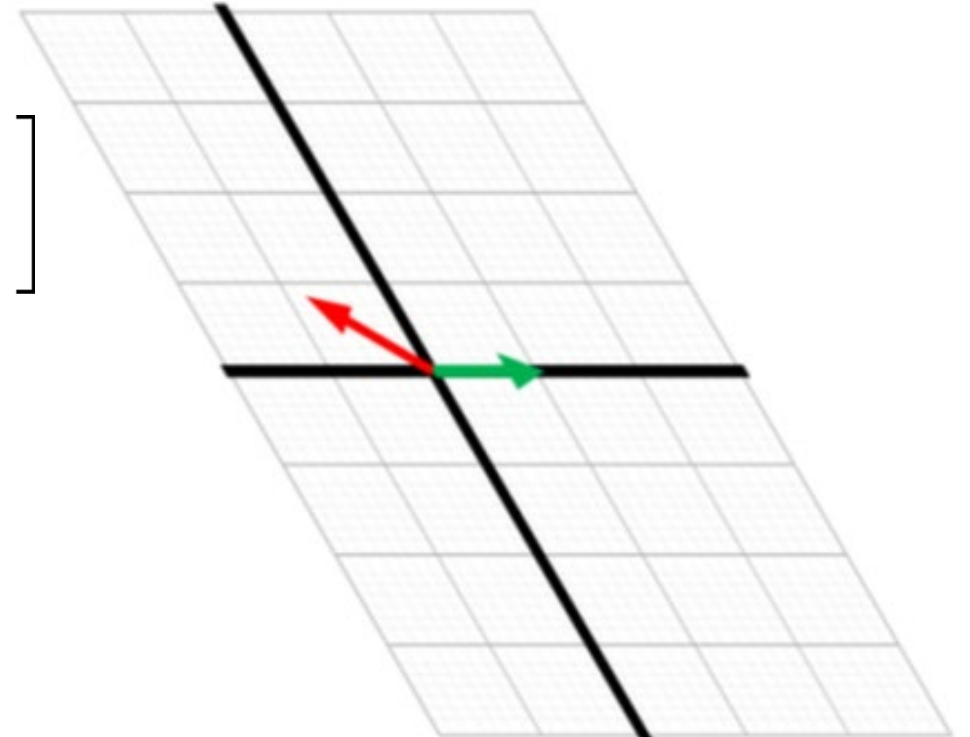
Eigenstuff

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

Eigendecomposition highlights what a linear transformation does by identifying directions that are *not* altered



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



Eigenstuff

Eigenvectors / values of a matrix solve the equation

$$Su = \lambda u$$

- Matrix S may have *multiple* eigenvectors / values that solve the above equation
- For D -dimensional u can find all vectors in $O(D^3)$ time
- PCA finds $M < D$ vectors with largest eigenvalues
- Can find $M < D$ sorted eigenvectors in $O(MD^2)$ time
- Note that D can be large!

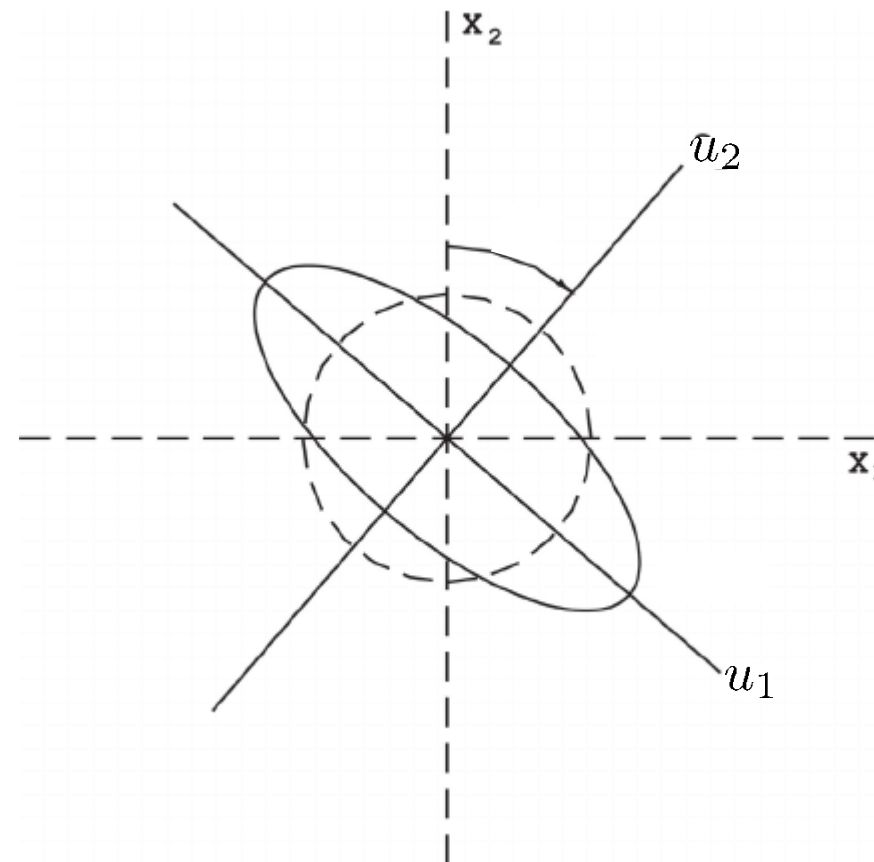
Eigenvectors and Ellipses

How does this connect to PCA?

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

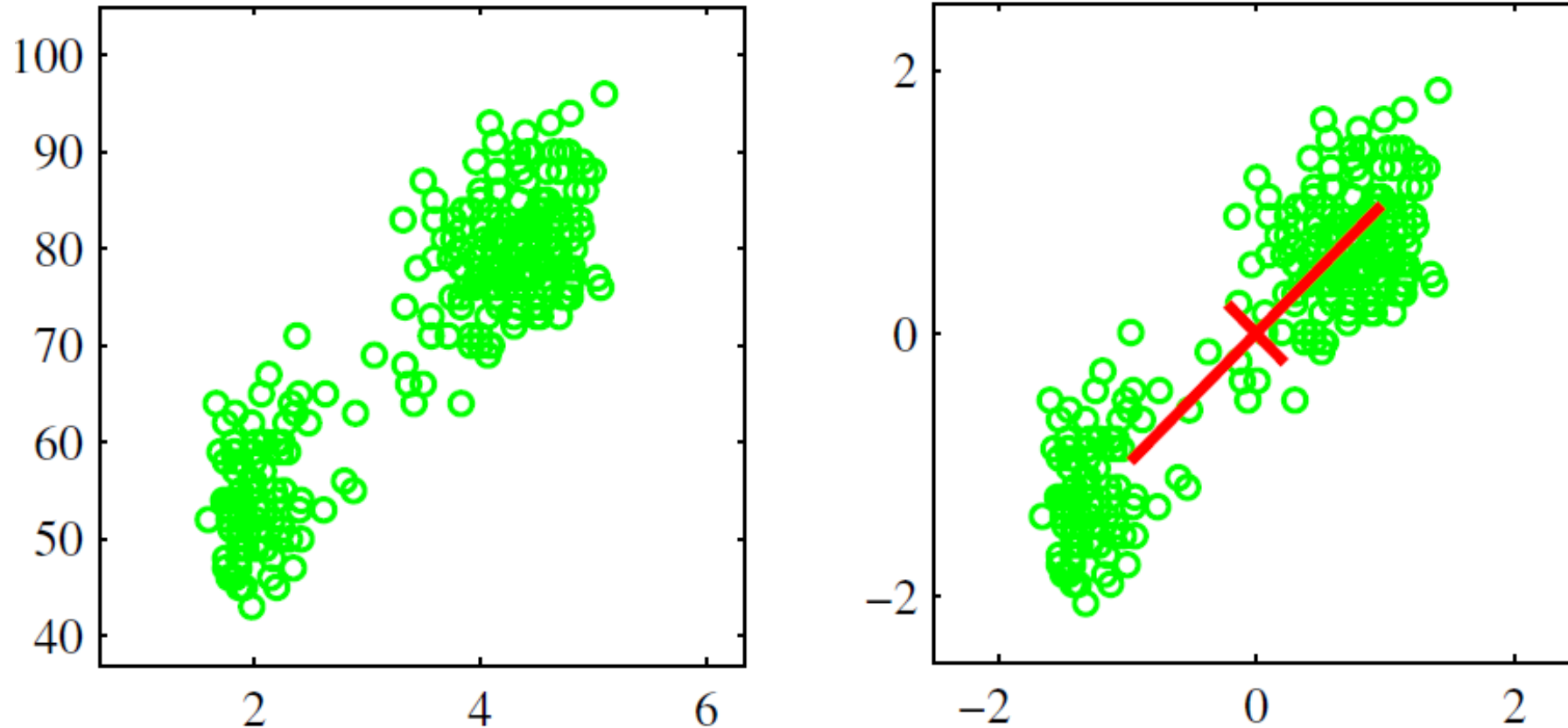
If S is a *covariance matrix*, then points will be transformed into an ellipse and...

- 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**



Principal Component Analysis (PCA)

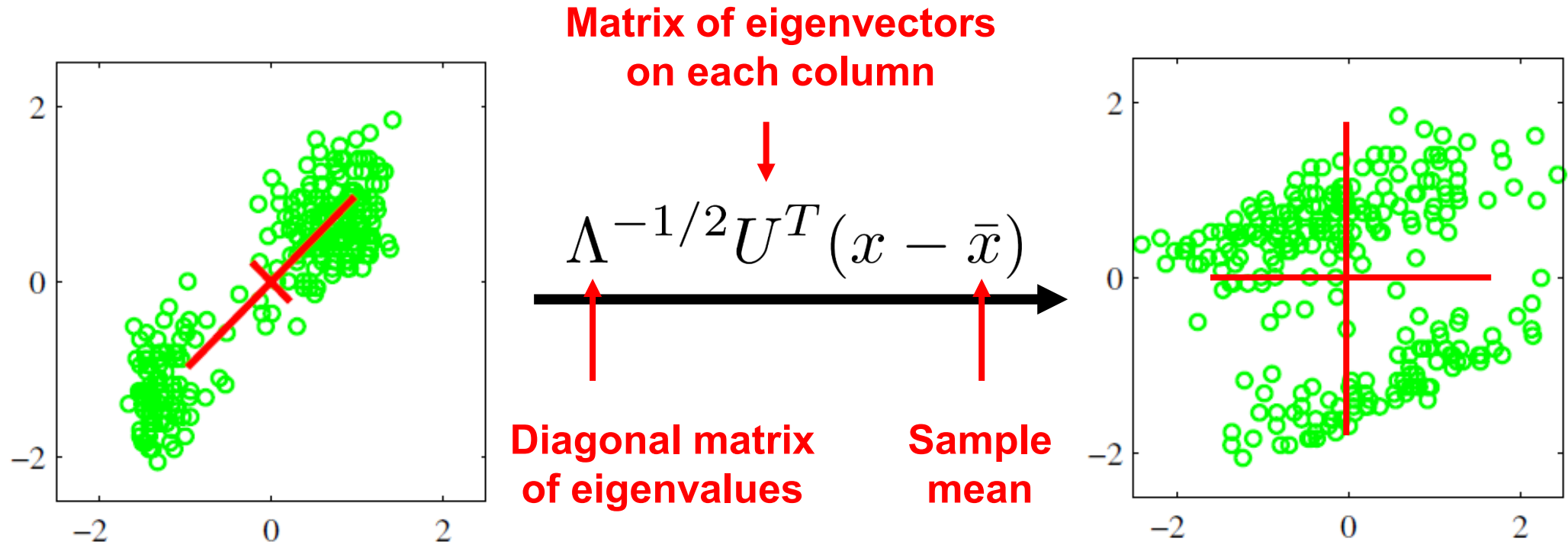
Sort eigenvectors by their eigenvalues...



...amount of variance in each principal component decreases with eigenvalue

Data “Whitening”

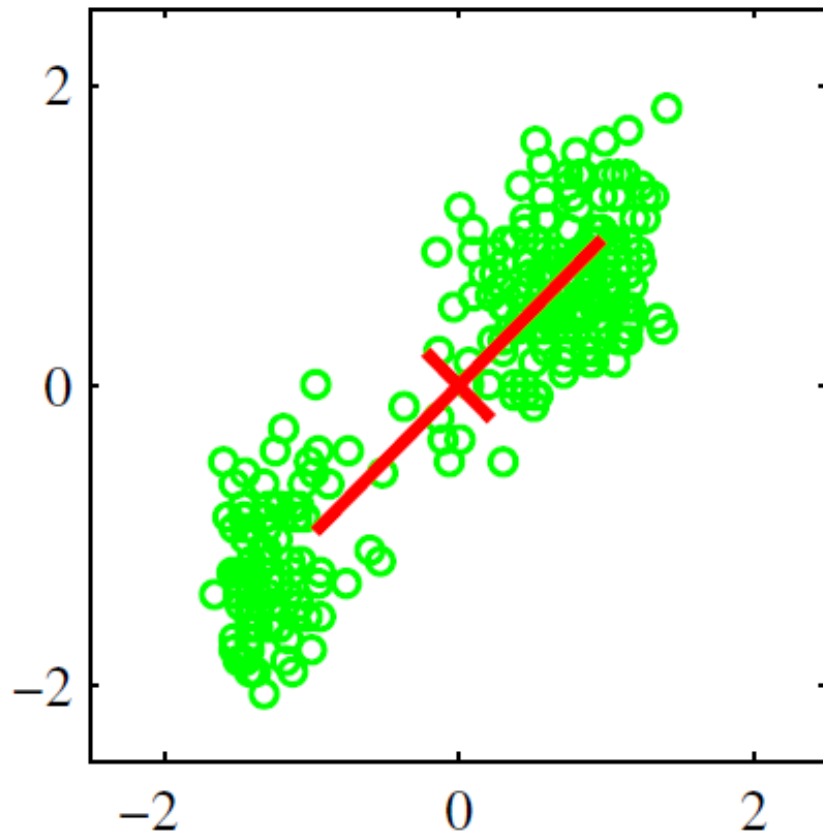
Multiplying data by eigenvectors transforms data so they are zero-mean and uncorrelated



Data whitening can be an important preprocessing step for many data science applications (even if we don't care about dimensionality reduction)

Principal Component Analysis (PCA)

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



Let u_1 be the first principal component, then variance of first PC is,

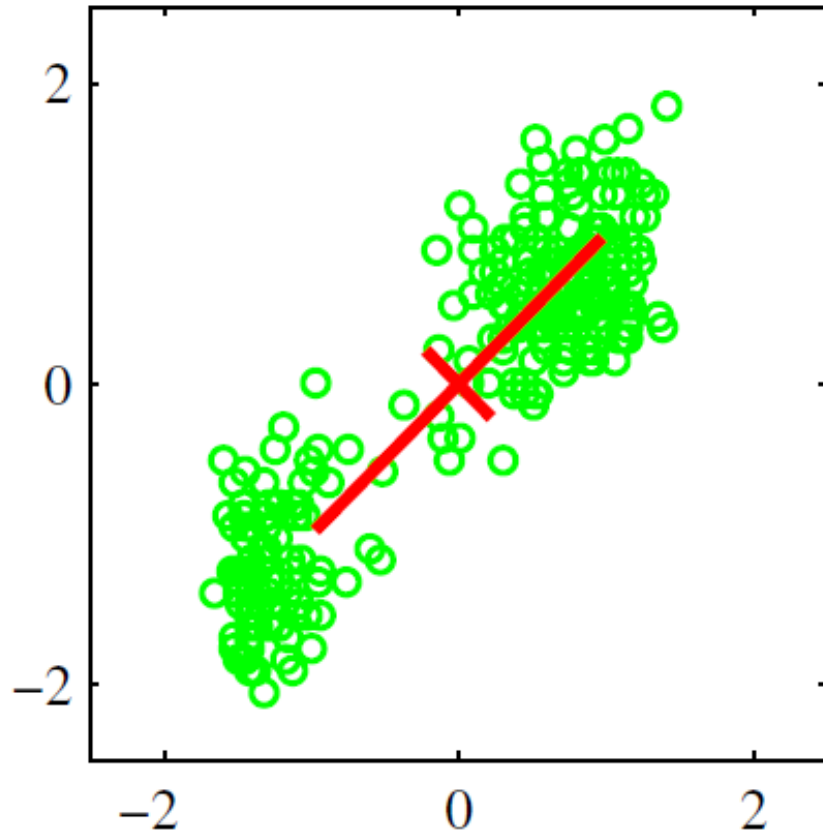
$$\frac{1}{N} \sum_n \{u_1^T (x_n - \bar{x})\}^2$$

How much in the second PC?

$$\frac{1}{N} \sum_n \{u_2^T (x_n - \bar{x})\}^2$$

Explained Variance

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



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

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

EM for PCA

We can derive an *expectation maximization* (EM) algorithm for PCA...but why would we do this if PCA is closed-form?

For N data points of D -dimensions

- Computing the first $M < D$ principal components takes $O(MD^2)$
- Evaluating the covariance needs $O(ND^2)$ time
- Most expensive step in EM is $O(NDM)$ time
- If D large and $M \ll D$ then $O(NDM) \ll O(ND^2)$

Unlike in GMM, EM always finds the exact solution for PCA

Concept Recap

Eigenvectors

- For a general linear transform – identify directions that are only stretched / shrunk / flipped
- For a covariance matrix – identify axes of the ellipse that describes covariance

PCA

- Learns linear subspace as $M < D$ principal components corresponding to M eigenvectors with largest eigenvalue
- Can be used to *whiten* (standardize, de-correlate) data
- Explained variance of M principal components easily calculated as percent of total explained variance in whitened data

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.

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)
```

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?

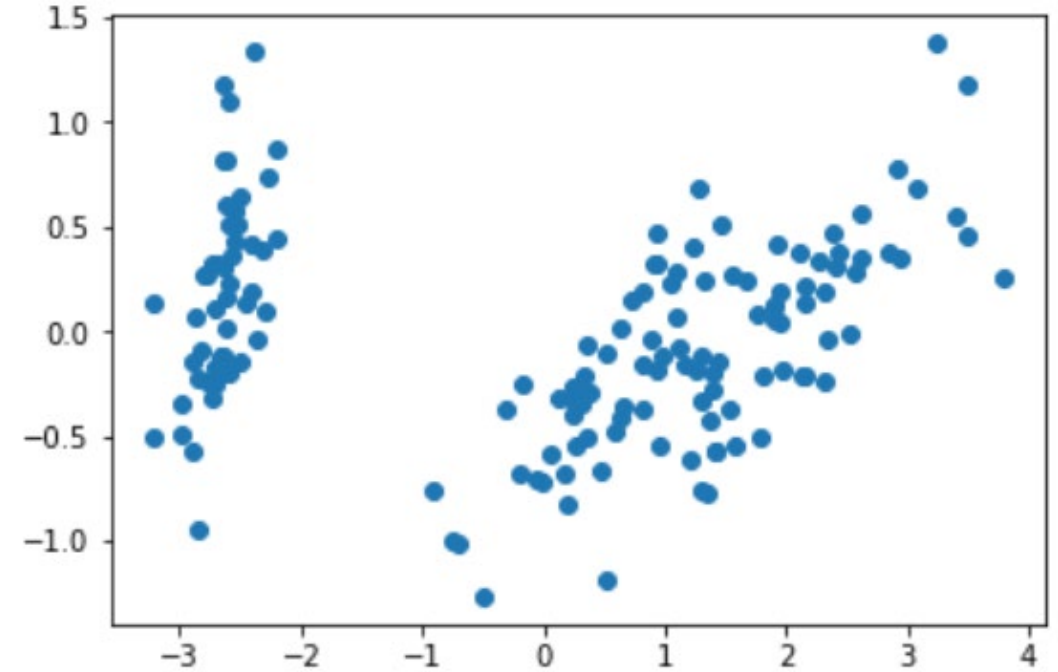
```
expvar = pca.explained_variance_ratio_
print('% Variance in 1st PC: ', expvar[0])
print('% Variance in 2nd PC: ', expvar[1])
print('Total explained variance: ', sum(expvar))
```

```
% Variance in 1st PC: 0.9246187232017271
% Variance in 2nd PC: 0.053066483117067804
Total explained variance: 0.977685206318795
```

Example : PCA on Iris Data

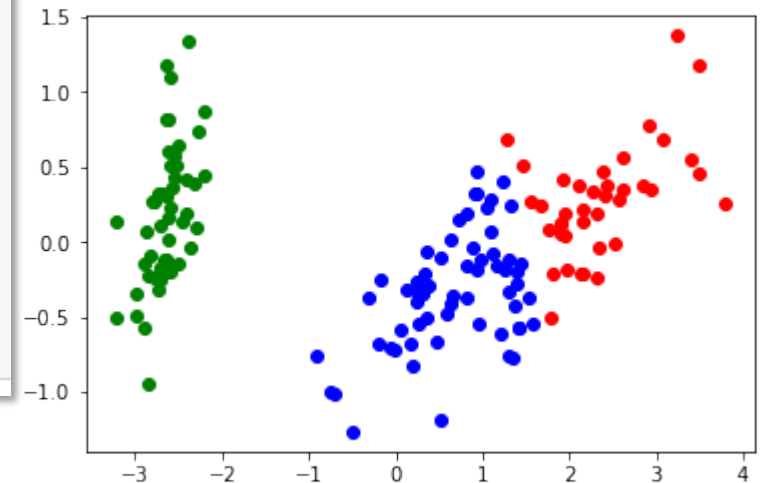
View data in 2-D subspace,

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



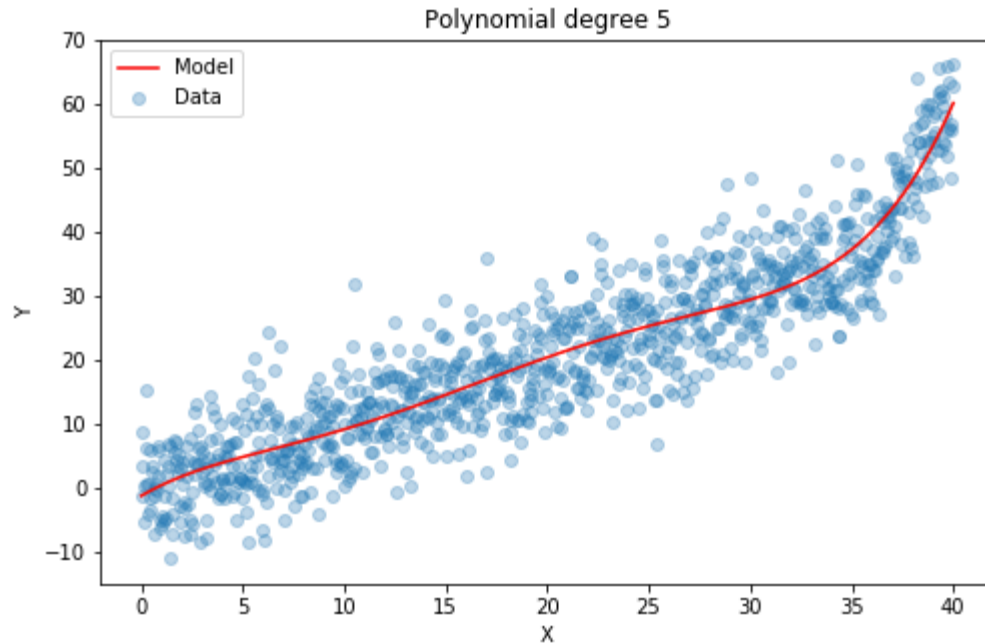
Do K-means clustering in 2-D subspace,

```
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()
```

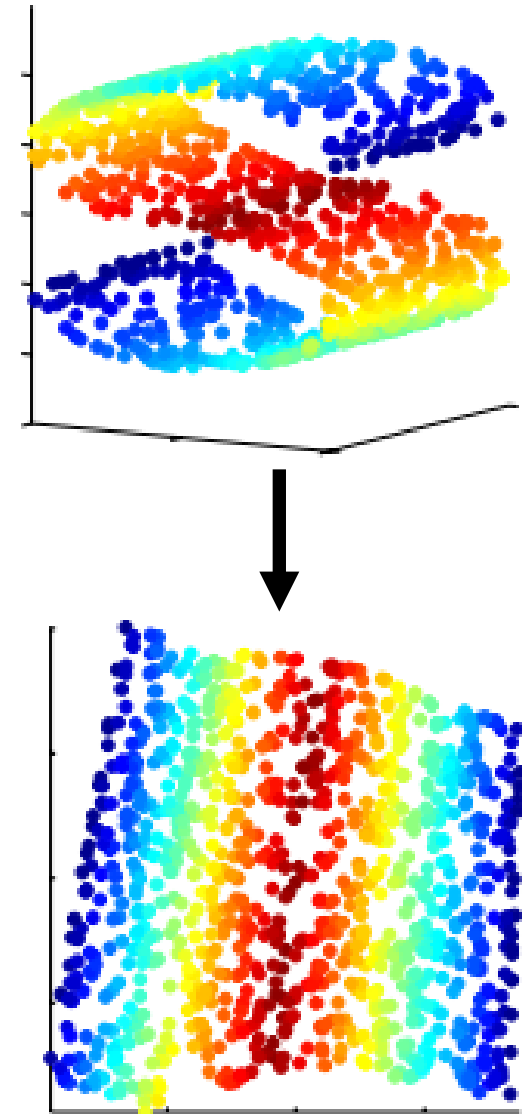


Nonlinear Dimensionality Reduction

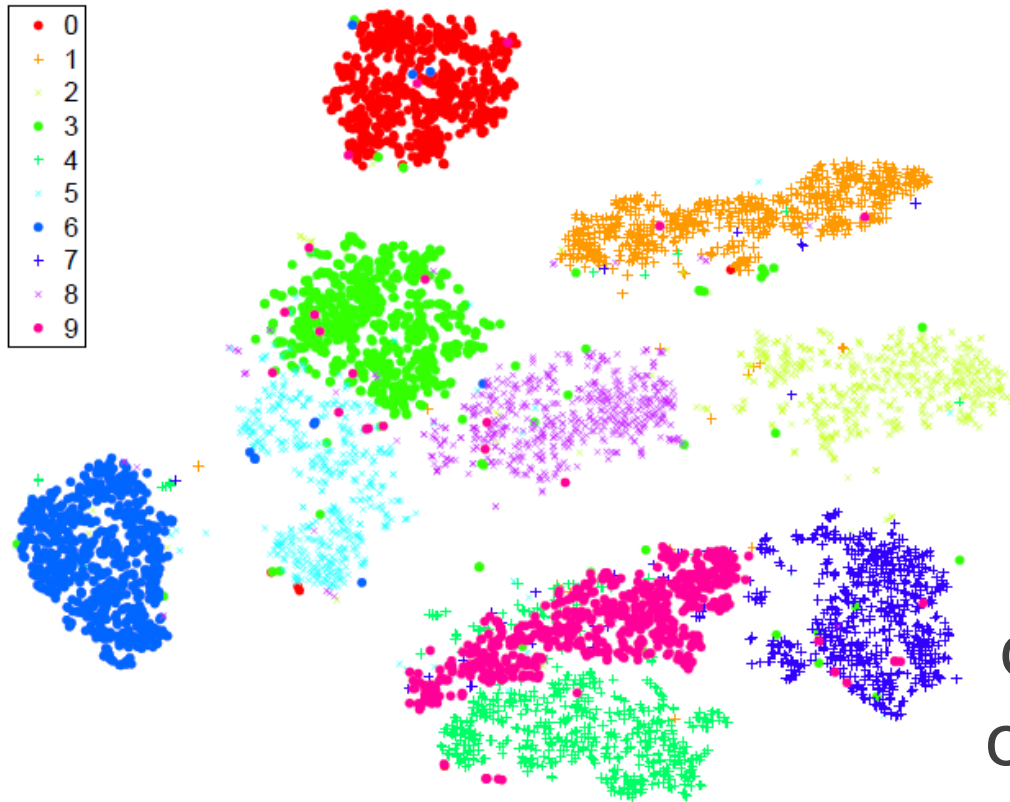
For general data, linear dimensionality reduction is not sufficient...



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 (recall from lecture on Neural Nets) projected to 2D and clustered

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

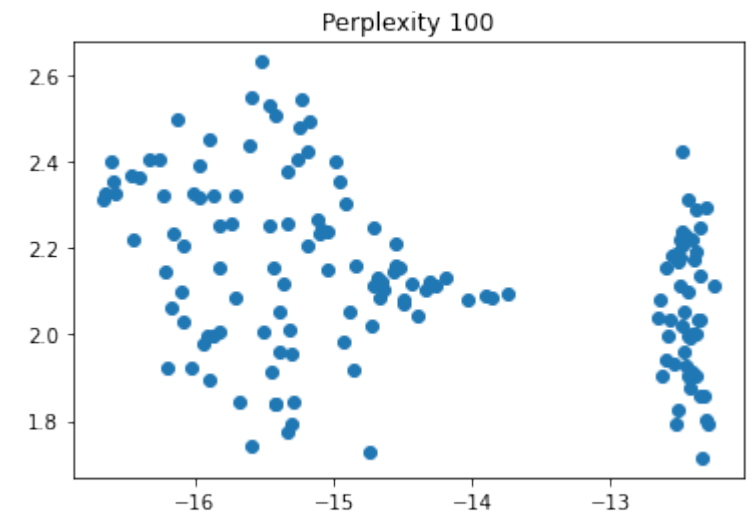
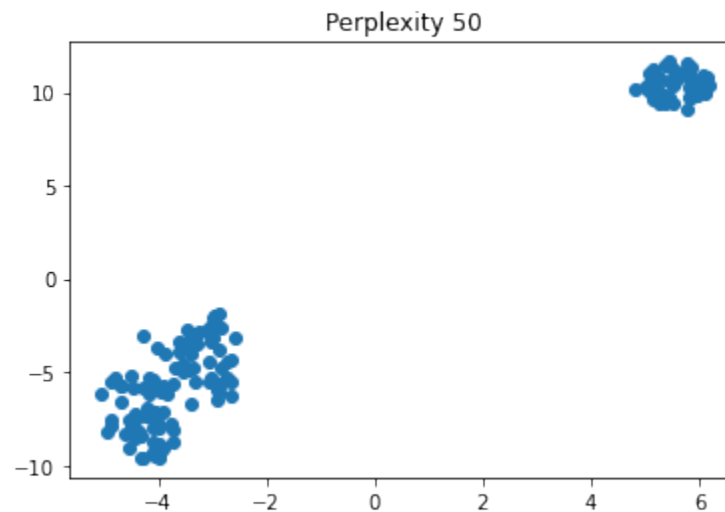
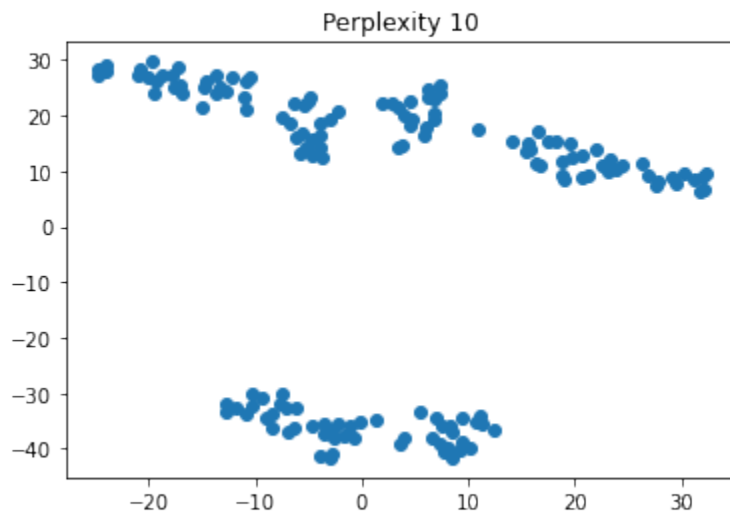
t-SNE can work surprisingly well...

```
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



Closing Comments

- 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 TSNE
- Other methods related to PCA (in *decomposition* sub-pkg):
 - Factor Analysis, Kernel PCA, Incremental PCA
- For multiple data sources, consider cross-decomposition
 - Canonical Correlation Analysis (CCA)
 - Learns same embedding for both spaces
 - Under *cross_decomposition* sub-package