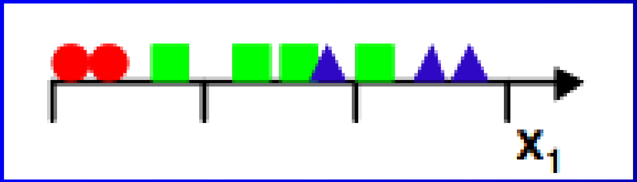
The Curse of Dimensionality

- Term coined by Bellman in 1961
- Refers to the problems associated with multivariate data analysis as the dimensionality increases
- As the number of features or dimensions within a dataset increases, the amount of data needed to effectively generalize or make accurate predictions increases exponentially.
  - Models learn from limited information

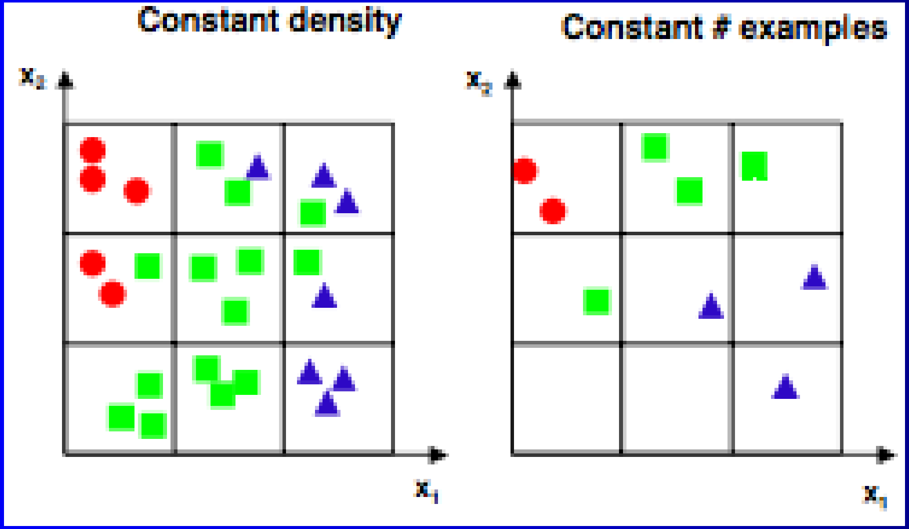
Curse of Dimensionality - Example

- Use 1 feature to divide the following data:



Too much overlap between triangle and square

- Add another dimension (feature):

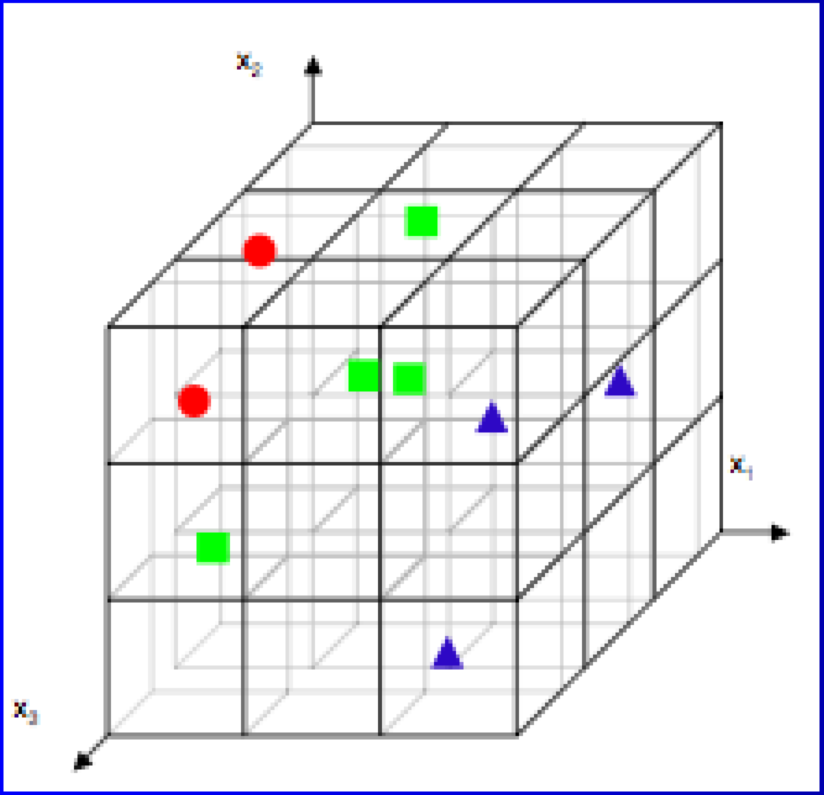


Do we maintain the number of samples?

- yes – then we have bins we can't identify what class
- no – when we increase the training samples, we see more overlap

Curse of Dimensionality - Example

- What about 3 dimensions (features):



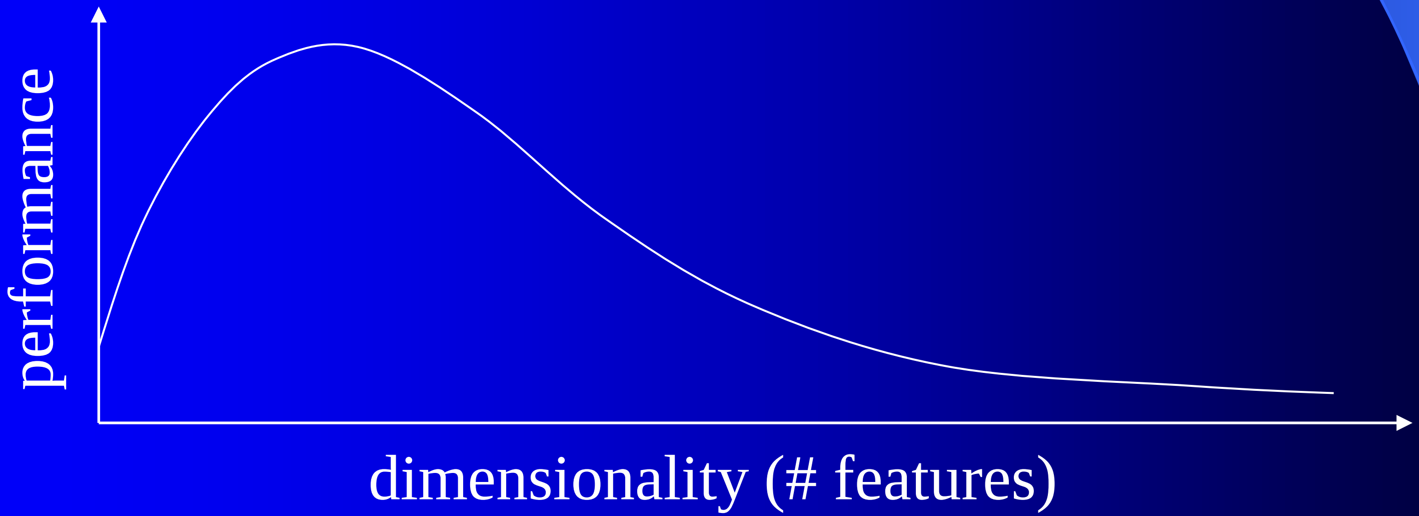
Number of bins grows $3^3 = 27$

For the same density in each bin, we would need 81 training samples

If we keep the same training set, we have a lot of empty space. How do we classify something that falls in an empty space we haven't seen?

Curse of Dimensionality

- How do we beat it?
  - Incorporating domain knowledge
  - Reducing the dimensionality
- In practice, the curse of dimensionality means that, for a given sample size, there is a maximum number of features above which the performance of a classifier will degrade rather than improve.



Feature Selection and Feature Reduction

- Given *n* original features, it is often advantageous to reduce this to a smaller set of features for actual training
  - Can improve/maintain accuracy if we can preserve the most relevant information while discarding the most irrelevant information
  - And/or can make the learning process more computationally and algorithmically manageable by working with less features
  - Curse of dimensionality requires an exponential increase in data set size in relation to the number of features to learn without overfit – thus decreasing features can be critical
- *Feature Selection* seeks a *subset* of the *n* original features which retains most of the relevant information
  - Filters, Wrappers
- *Feature Reduction* <u>*combines/fuses*</u> the *n* original features into a smaller set of newly created features which hopefully retains most of the relevant information from *all* the original features - Data fusion (e.g. LDA, PCA, t-SNE, UMAP etc.)

Feature Selection - Filters

- Given *n* original features, how do you select size of subset
  - User can preselect a size *p* (< *n*) – not usually as effective
  - Usually try to find the smallest size where adding more features does not yield improvement
- 1-Feature Accuracy Feature Selection
  - Create model with each feature independent of the others
  - Select top p features ordered by 1-feature scores
- Score subsets of features together?
  - Exponential number of subsets requires a more efficient, sub-optimal search approach
  - How to score features is independent of the ML model to be trained on and is an important research area
  - Decision Tree or other ML model pre-process

Feature Selection - Wrappers

- Optimizes for a specific learning algorithm
- The feature subset selection algorithm is a "wrapper" around the learning algorithm
  1. Pick a feature subset and pass it to learning algorithm
  2. Create training/test set based on the feature subset
  3. Train the learning algorithm with the training set
  4. Find accuracy (objective) with validation set
  5. Repeat for all feature subsets and pick the feature subset which gives the highest predictive accuracy (or other objective)
- Basic approach is simple
- Variations are based on how to select the feature subsets, since there are an exponential number of subsets

Feature Selection - Wrappers

- Exhaustive Search – Exhausting

- Forward Search – O(n^2 · learning/testing time) - Greedy
  1. Score each feature by itself and add the best feature to the initially empty set *FS* (*FS* will be our final Feature Set)
  2. Try each subset consisting of the current *FS* plus one remaining feature and add the best feature to *FS*
  3. Continue until stop getting significant improvement (over a window)
- Backward Search – O(n^2 · learning/testing time) - Greedy
  1. Score the initial complete *FS*
  2. Try each subset consisting of the current *FS* minus one feature in *FS* and drop the feature from *FS* causing least decrease in accuracy
  3. Continue until dropping any feature causes a significant decreases in accuracy

- Branch and Bound and other heuristic approaches available

PCA – Principal Components Analysis

- PCA is one of the most common feature reduction techniques
- A linear method for dimensionality reduction
- Allows us to combine much of the information contained in n features into p features where $p < n$
- PCA is *unsupervised* in that it does not consider the output class/value of an instance – There are other algorithms which do (e.g. Linear Discriminant Analysis)
- PCA works well in many cases where data features have mostly linear correlations
- Non-linear dimensionality reduction is also a successful area and can give better results for data with significant non-linear correlations between the data features

PCA Overview

- Seek new set of bases (coordinate system) which correspond to the highest variance in the data
- Transform *n*-dimensional *normalized* data to a new *n*-dimensional basis
  - The new dimension with the most variance is the first principal component
  - The next is the second principal component, etc.
  - Note z_1 combines/fuses significant information from both x_1 and x_2

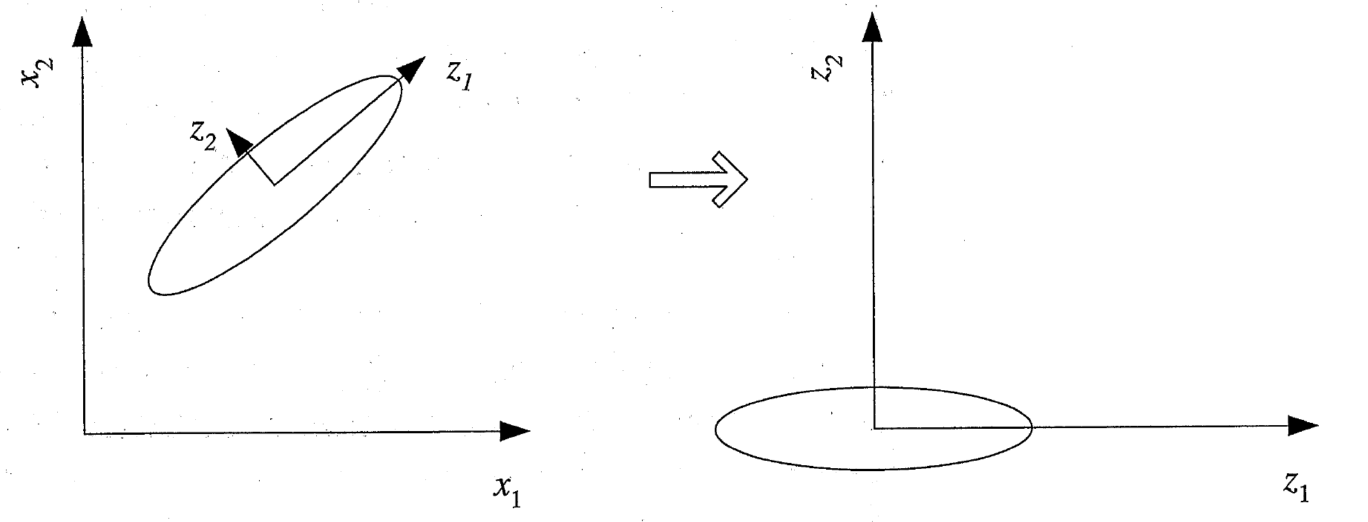
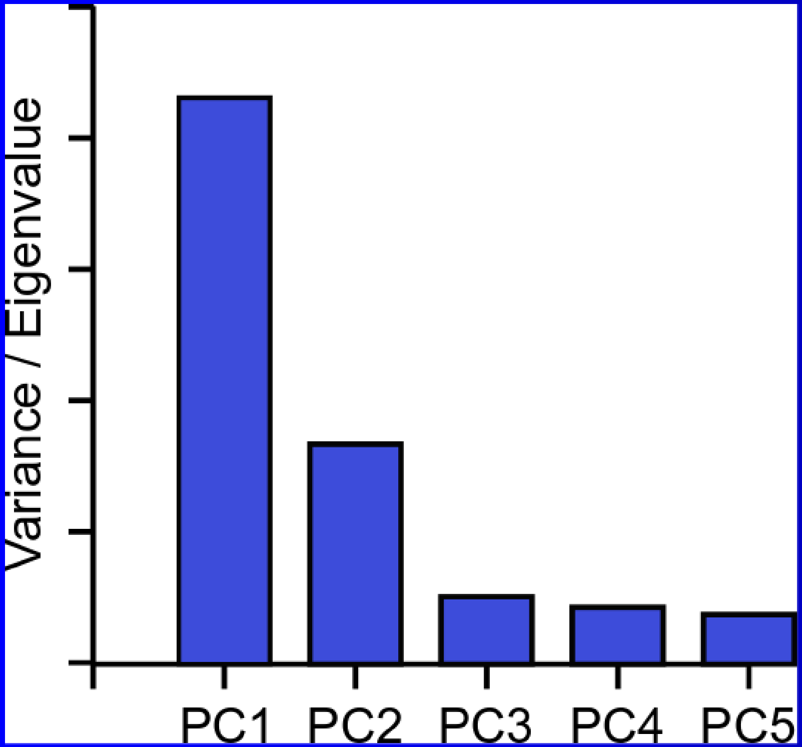
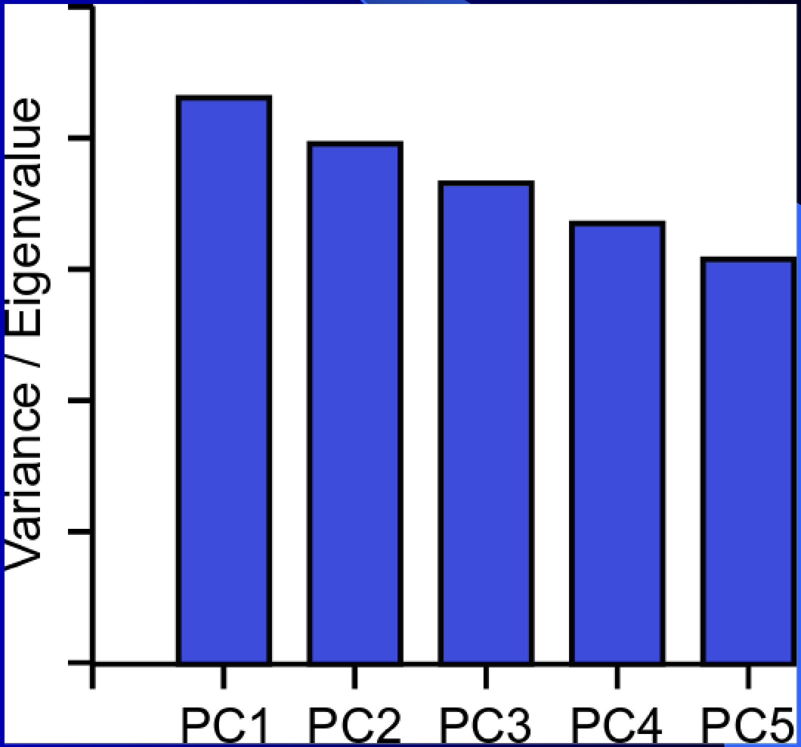


Figure 6.1 Principal components analysis centers the sample and then rotates the axes to line up with the directions of highest variance. If the variance on z_2 is too small, it can be ignored and we have dimensionality reduction from two to one.

Explaining Variance

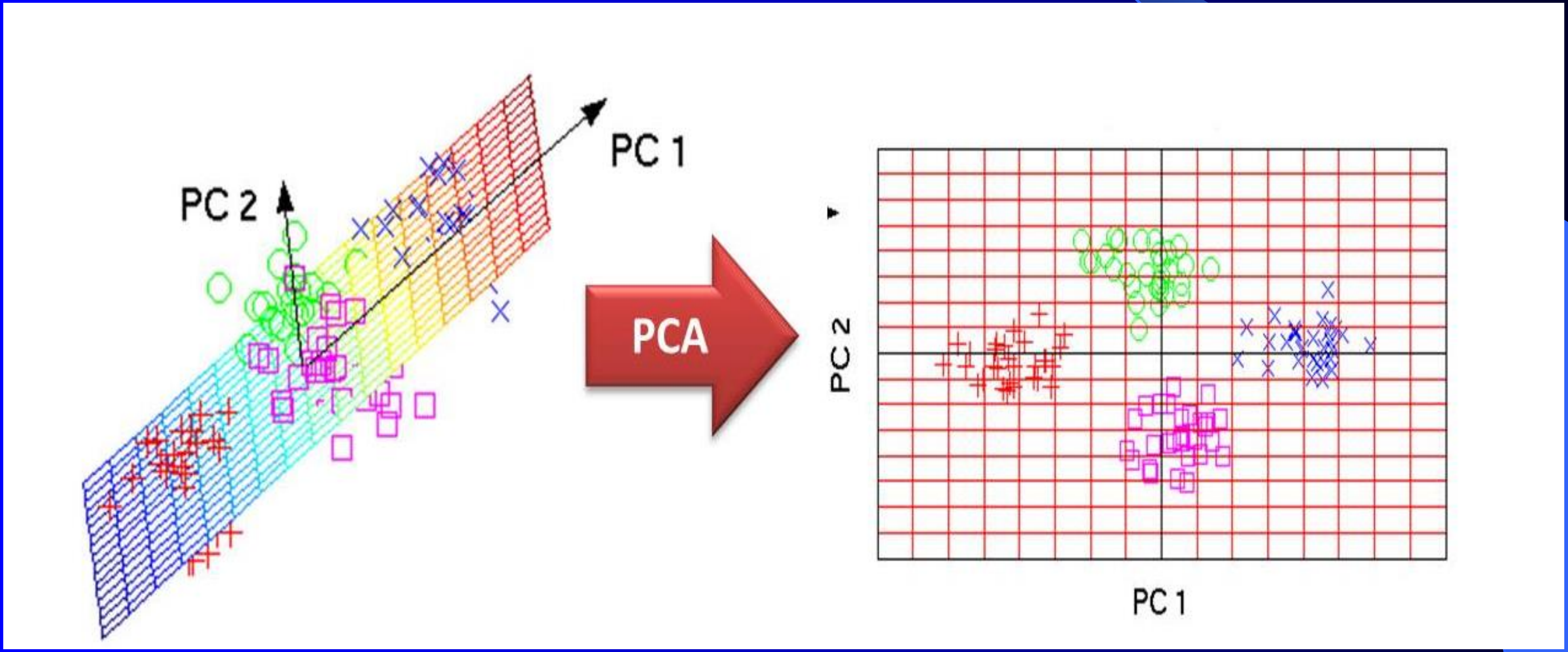
- All PC's are some linear combination of current features
- Largest PC is the new axis that shows the most variance





PCA Overview

- Can drop dimensions for which there is little variance



Variance and Covariance

- Variance is a measure of data spread in one feature/dimension
  - *n* features, *m* instances in data set
  - Note *n* in variance/covariance equations is number of instances in the data set, apologies
- Covariance measures how two dimensions (features) vary with respect to each other
- Standardize data features so they have similar magnitudes else covariance may not be as informative

$$\mathrm{var}(X) = \frac{\sum_{i=1}^{n} (X_i - \bar{X})(X_i - \bar{X})}{(n-1)}$$

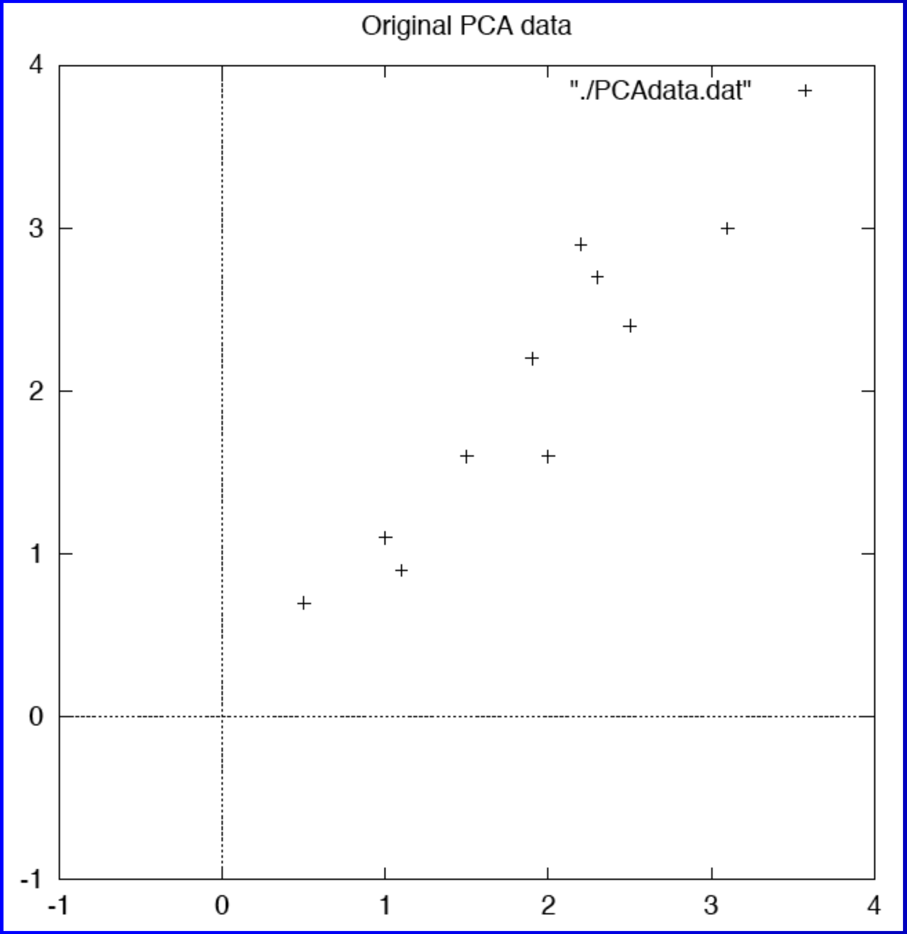
$$\mathrm{cov}(X,Y) = \frac{\sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})}{(n-1)}$$

Covariance and the Covariance Matrix

- Considering the sign (rather than exact value) of covariance:
  - Positive value means that as one feature increases or decreases the other does also (positively correlated)
  - Negative value means that as one feature increases the other decreases and vice versa (negatively correlated)
  - A value close to zero means the features are independent
  - If highly covariant, are both features necessary?
- Covariance matrix is an $n \times n$ matrix containing the covariance values for all pairs of features in a data set with n features (dimensions)
- The diagonal contains the covariance of a feature with itself which is the variance (i.e. the square of the standard deviation)
- The matrix is symmetric

PCA Example

- First step is to standardize the data by subtracting the mean in each dimension and then dividing by the standard deviation



Data	x	y	x'	y'
	2.5	2.4	0.92	0.61
	0.5	0.7	-1.79	-1.51
	2.2	2.9	0.52	1.23
	1.9	2.2	0.11	0.36
	3.1	3.0	1.74	1.36
	2.3	2.7	0.65	0.98
	2.0	1.6	0.24	-0.39
	1.0	1.1	-1.11	-1.01
	1.5	1.6	-0.43	-0.39
	1.2	0.9	-0.84	-1.26
Mean	1.82	1.91		
StDev	0.736	0.803		

Can Normalize rather than standardize

PCA Example

- Second: Calculate the covariance matrix of the centered data
- Only 2 × 2 for this case

Data	x	y	x'	y'
	2.5	2.4	0.92	0.61
	0.5	0.7	-1.79	-1.51
	2.2	2.9	0.52	1.23
	1.9	2.2	0.11	0.36
	3.1	3.0	1.74	1.36
	2.3	2.7	0.65	0.98
	2.0	1.6	0.24	-0.39
	1.0	1.1	-1.11	-1.01
	1.5	1.6	-0.43	-0.39
	1.2	0.9	-0.84	-1.26
Mean	1.82	1.91		
StDev	0.736	0.803		

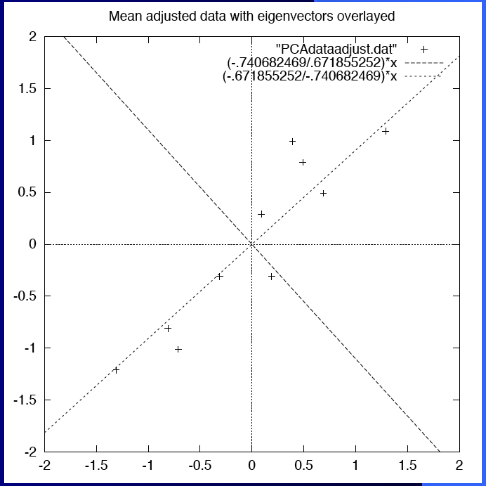
$$\operatorname{cov}(X,Y) = \frac{\sum_{i=1}^{n}\left(X_i - \overline{X}\right)\left(Y_i - \overline{Y}\right)}{(n-1)}$$

Covariance	**Matrix**
1.111111	1.022377
1.022377	1.111111

PCA Example

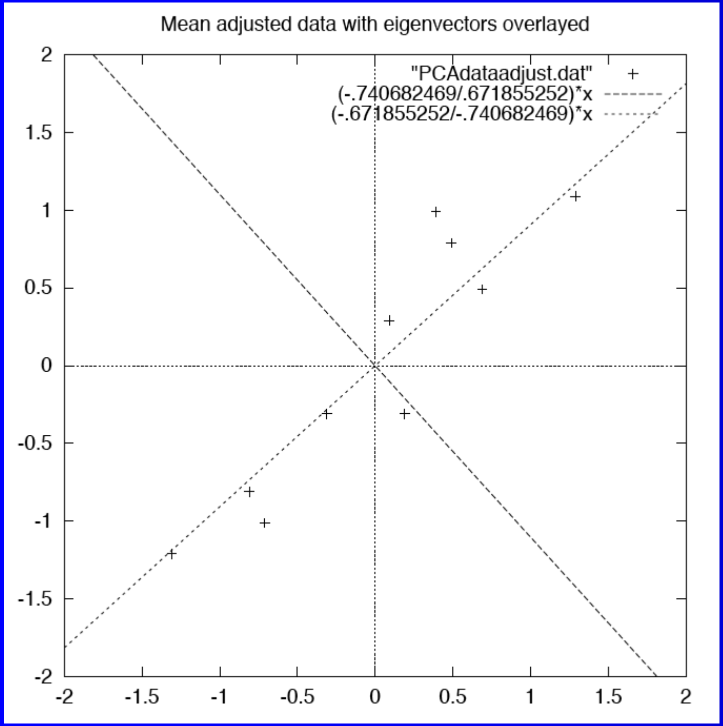
- Third: Calculate the unit eigenvectors and eigenvalues of the covariance matrix (remember your linear algebra)
  - Covariance matrix is always square $n \times n$ and positive semi-definite, thus n non-negative eigenvalues will exist
  - All eigenvectors (principal components) are orthogonal to each other and form the new set of bases/dimensions for the data (columns)
  - The magnitude of each eigenvalue corresponds to the variance along each new dimension – Just what we wanted!
  - We can sort the principal components according to their eigenvalues
  - Just keep those dimensions with the largest eigenvalues

Principal Component	Eigenvalues	Eigenvectors	
1	2.13348836	0.70710678	0.70710678
2	0.08873385	-0.70710678	0.70710678



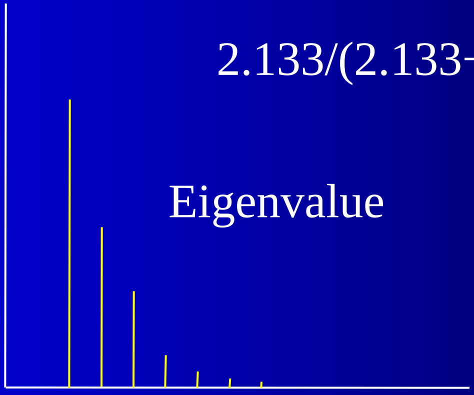
PCA Example

- Below are the two eigenvectors overlaying the centered data
- Which eigenvector has the largest eigenvalue?
- Fourth Step: Just keep the *p* eigenvectors with the largest eigenvalues
  - Do lose some information, but if we just drop dimensions with small eigenvalues then we lose only a little information, hopefully noise
  - We can then have *p* input features rather than *n*
  - The *p* features contain the most pertinent *combined* information from all *n* original features
  - How many dimensions *p* should we keep?



Principal Component	Eigenvalues	Eigenvectors	
1	2.13348836	0.70710678	0.70710678
2	0.08873385	-0.70710678	0.70710678

2.133/(2.133+ 0.088) = .96



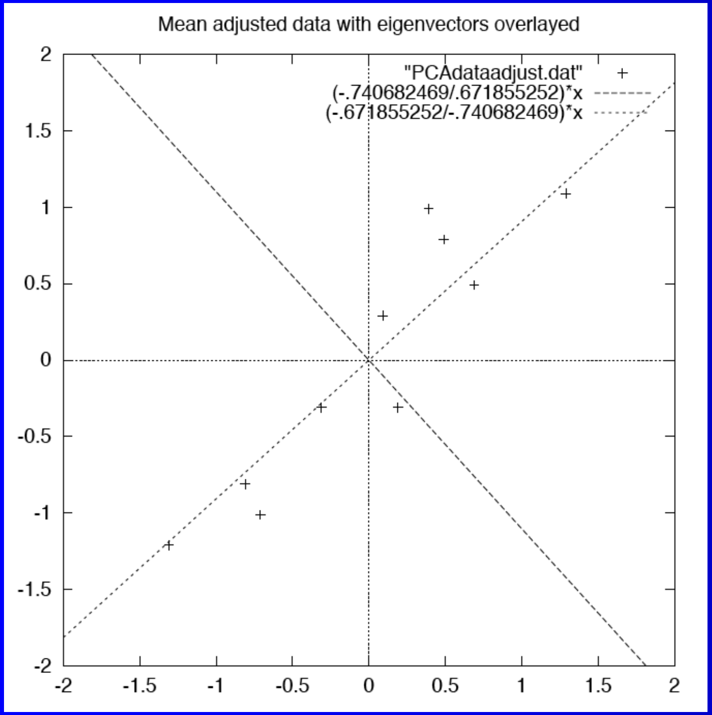
1 2 3 4 5 6 7 … *n*

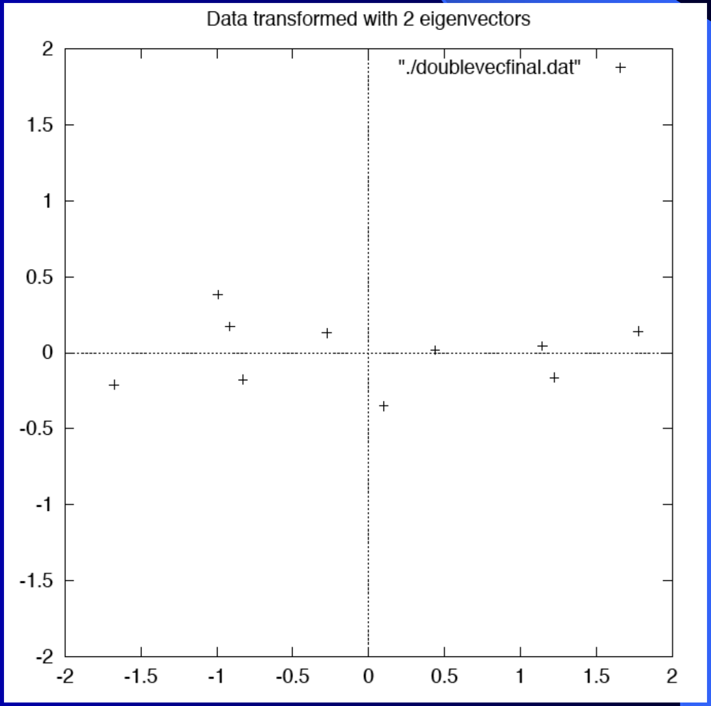
Proportion of Variance

$$\frac{\sum_{i=1}^{p} \lambda_i}{\sum_{i=1}^{n} \lambda_i} = \frac{\lambda_1 + \lambda_2 + \ldots + \lambda_p}{\lambda_1 + \lambda_2 + \ldots + \lambda_p + \ldots + \lambda_n}$$

PCA Example

- Last Step: Transform the n features to the p ($< n$) chosen bases (Eigenvectors)
- Transform data (m instances) with a matrix multiply $T = A \times B$
  - A is a $p \times n$ matrix with the p principal components in the rows, component one on top
  - B is a $n \times m$ matrix containing the transposed centered original data set
  - T^T is a $m \times p$ matrix containing the transformed data set
- Now we have the new transformed data set with p features
- Keep matrix A to transform future centered data instances
- Below is the transform of both dimensions. What if we just kept the 1st component for this case?





PCA Algorithm Summary

1. Standardize the TS features
2. Calculate the covariance matrix of the standardized TS
3. Calculate the unit eigenvectors and eigenvalues of the covariance matrix
4. Keep the *p* (< *n*) eigenvectors with the largest eigenvalues
5. Matrix multiply the *p* eigenvectors with the standardized TS to get a new TS with only *p* features

- Given a novel instance during execution
    1. Standardize the instance (use the mean and std. dev. of the TS)
    2. Do the matrix multiply (step 5 above) to change the new instance from *n* to *p* features

Note: TS = Training Set

PCA Algorithm Summary

1. Standardize the TS features

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X)
scaled_data = scaler.transform(X)
```

2. Calculate the covariance matrix of the standardized TS

```
cov_matrix = np.cov(scaled_data.T)
```

3. Calculate the unit eigenvectors and eigenvalues of the covariance matrix

```
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
```

4. Keep the p ($< n$) eigenvectors with the largest eigenvalues

```
projection_matrix = (eigenvectors.T[:][:num_components]).T
```

1. Matrix multiply the p eigenvectors with the standardized TS to get a new TS with only p features

```
X_pca = scaled_data.dot(projection_matrix)
```

Note: TS = Training Set

PCA with sklearn

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# We will start from the original data in X
# We need to scale the data
scaler = StandardScaler()
scaler.fit(X)
new_scaled_data = scaler.transform(X)

# Create a PCA object using a reduced number of components
pca = PCA(n_components=num_components)

pca.fit(new_scaled_data)
X_pca = pca.transform(new_scaled_data)
```

PCA Homework

Terms		
m	5	Number of instances in data set
n	2	Number of input features
p	1	Final number of principal components chosen

Original Data			
	x1	*x2*	*Out*
m1	.2	-.3	-
m2	-1.1	2	-
m3	1	-2.2	-
m4	.5	-1	-
m5	-.6	1	-
mean			

- Use PCA on the given data set to get a transformed data set with just one feature (the first principal component (PC)). Show your work along the way.
- Show what % of the total information is contained in the 1st PC.
- Do not use a PCA package to do it. You need to go through the steps yourself, or program it yourself. You may use a spreadsheet, Matlab, etc. to do the arithmetic for you.
- You may use Python or any web tool to calculate the eigenvectors/eigenvalues from the covariance matrix.
- Optional: After, use any PCA solver (e.g. sklearn) and use it to solve the problem and check your answers.

PCA Summary

- PCA is a linear transformation, so if the features have highly non-linear correlations, the transformed data will be less useful
  - Non linear dimensionality reduction techniques can sometimes handle these situations better (e.g. LLE, Isomap, Manifold-Sculpting)
  - PCA is good at removing redundant linearly correlated features
- With high dimensional data the eigenvector is a hyper-plane
- Interesting note: The 1st principal component is the multiple regression plane that delta rule will always discover
- Caution: Not a "cure all" and can lose important info in some cases
  - How would you know if it is effective?
  - Just compare accuracies of original vs transformed data set

Practical Feature Reduction

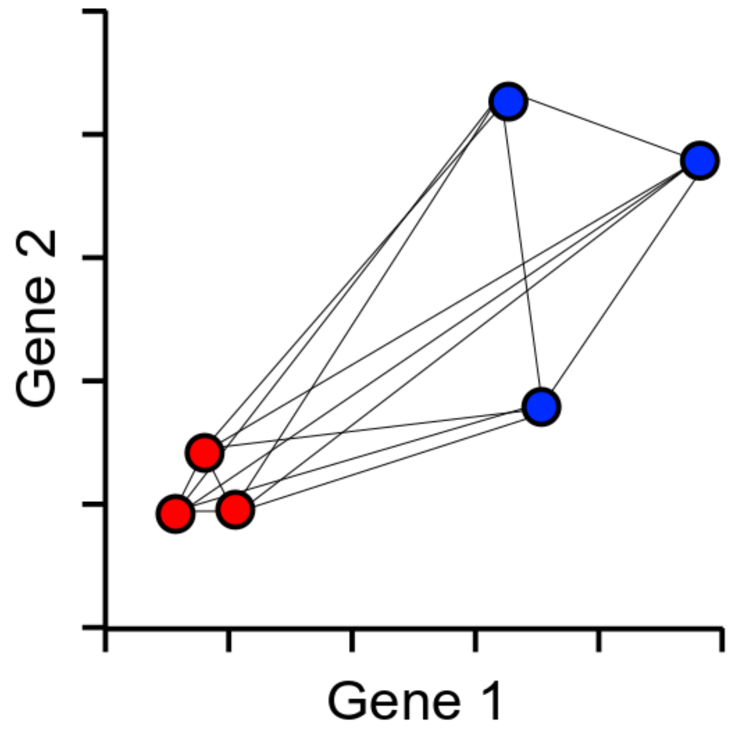
- Assume you have a data set with 50 features
- You might like to reduce if possible (you might hope for 10 or so, but let the results decide)
- Could try PCA – Compare with non-PCA results to see how effective PCA is for the particular data set
- Could also try a wrapper (e.g. backward greedy) and compare its results and then go with what gives the best accuracy
- PCA
  - Pro: Potentially fuses most information from all features into new smaller set of features
  - Con: Will fail if features have lots of non-linear correlations
- Wrappers
  - Pro: Can handle data features with arbitrary non-linear correlations
  - Con: Does not fuse info, those features which are dropped are completely gone

Other Dimensionality Reduction Techniques

- LDA (Linear Discriminant Analysis)
  - projects data from a higher-dimensional space to a lower-dimensional space, but it aims to maximize the separation between classes while minimizing the variance within each class.
- t-SNE (t-Distributed Stochastic Neighbor Embedding)
  - non-linear dimensionality reduction technique commonly used for visualizing high-dimensional data in a lower-dimensional space.
  - works by minimizing the divergence between two probability distributions: a Gaussian distribution that represents pairwise similarities between data points in the high-dimensional space and a Student's t-distribution that represents pairwise similarities in the low-dimensional space.
- UMAP (Uniform Manifold Approximation and Projection)
  - similar to t-SNE, but with better scalability and preservation of global structure.
  - Generally faster than t-SNE

t-SNE

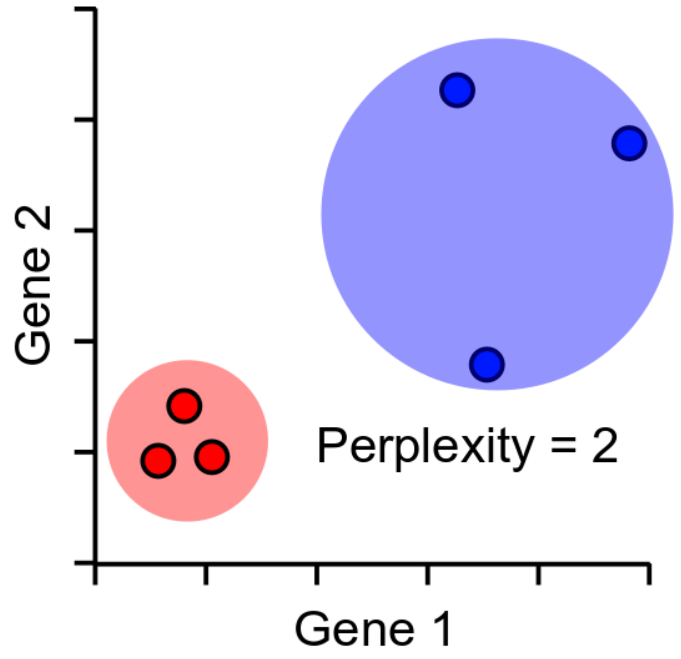
- t-distributed Stochastic Neighbor Embedding
- Non-linear
- Based on all-to-all table of pairwise distances



	🟥	🟥	🟥	🟦	🟦	🟦
🟥	0	10	10	295	158	153
🟥	9	0	1	217	227	213
🟥	1	8	0	154	225	238
🟦	205	189	260	0	23	45
🟦	248	227	246	44	0	54
🟦	233	176	184	41	36	0

t-SNE

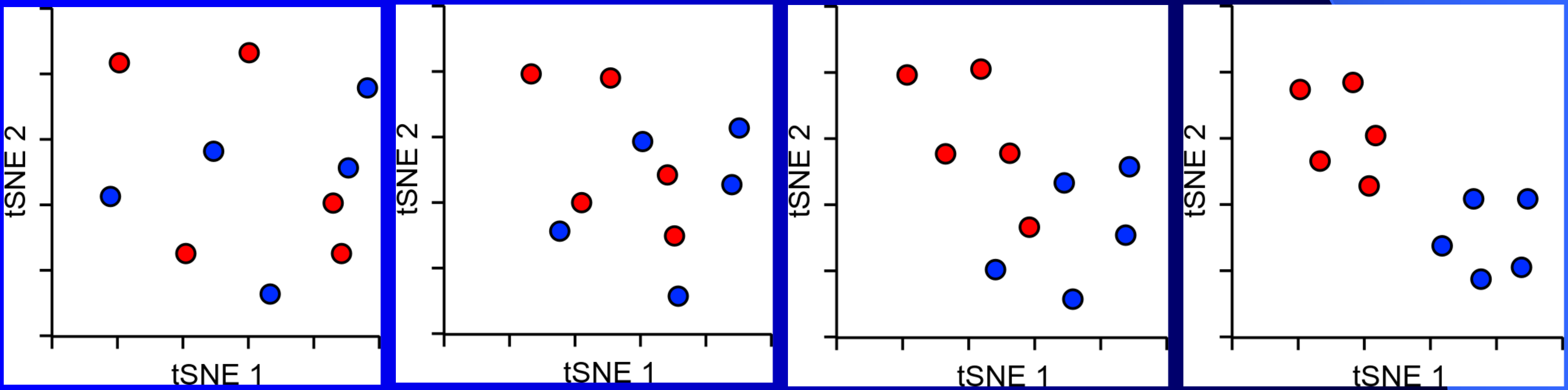
- Perplexity – hyperparameter
  - Expected number of neighbors within a cluster
- Distances are scaled relative to perplexity cluster neighbors

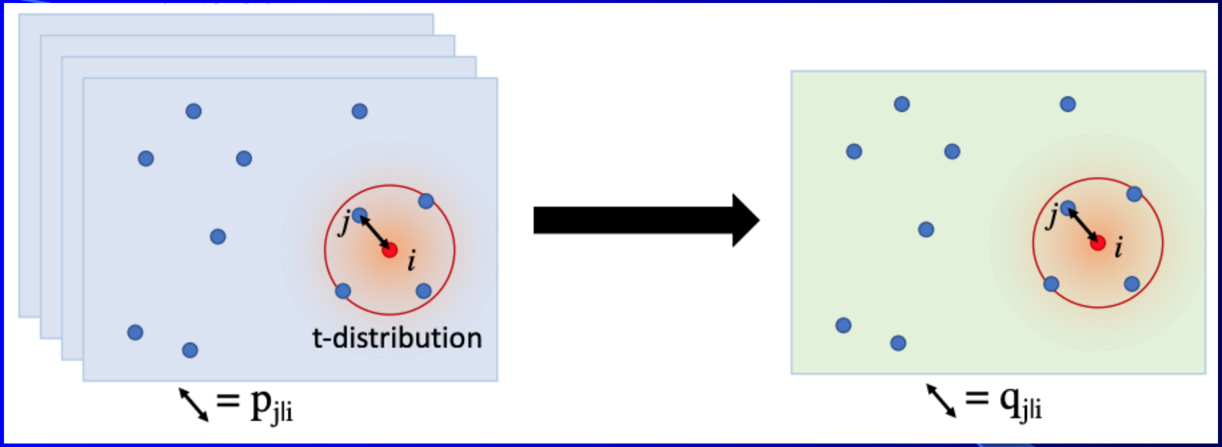


	🟥	🟥	🟥	🟦	🟦	🟦
🟥	0	4	6	586	657	836
🟥	4	0	4	815	527	776
🟥	9	3	0	752	656	732
🟦	31	28	29	0	4	7
🟦	31	24	25	4	0	7
🟦	40	37	32	8	8	0

t-SNE

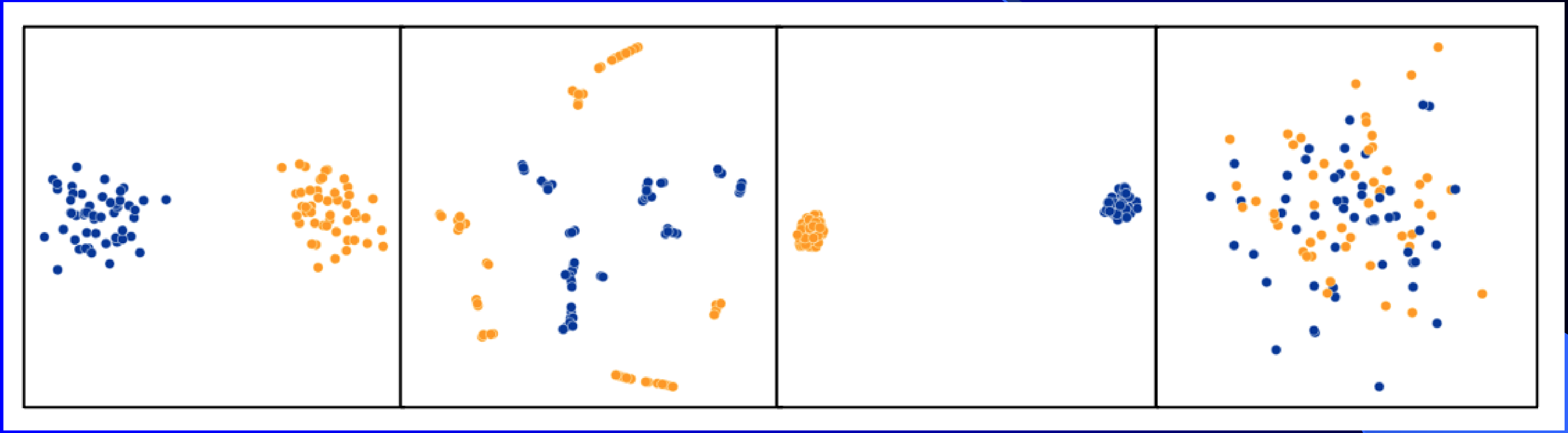
- Pick a new dimensionality
- Randomly scatter points in the new space
- Shuffle points based on how well they match the original distance matrix
- Continue until distances converge



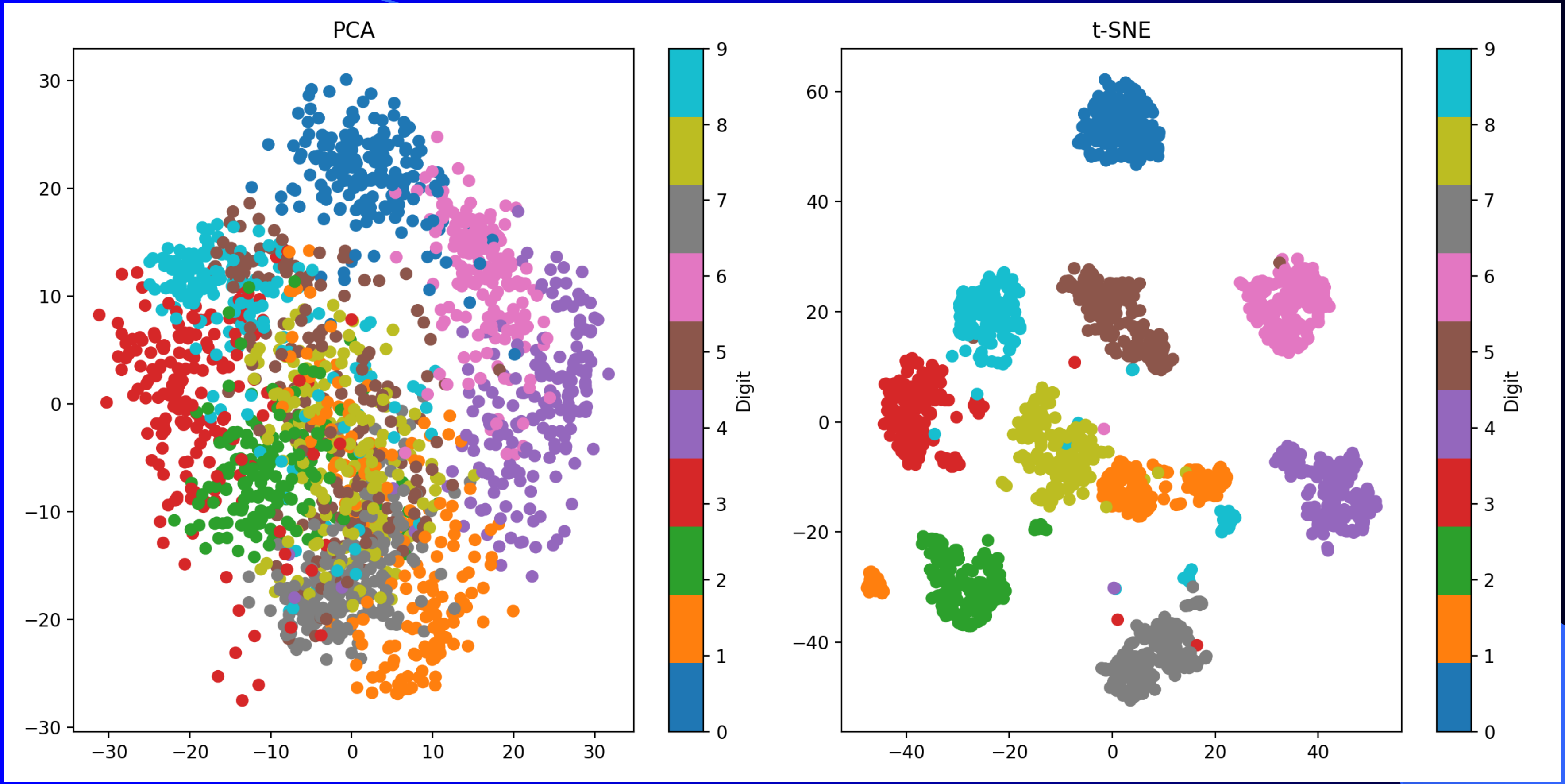

j
i
t-distribution
$\nwarrow\!\!\searrow = p_{j|i}$
j
i
$\nwarrow\!\!\searrow = q_{j|i}$

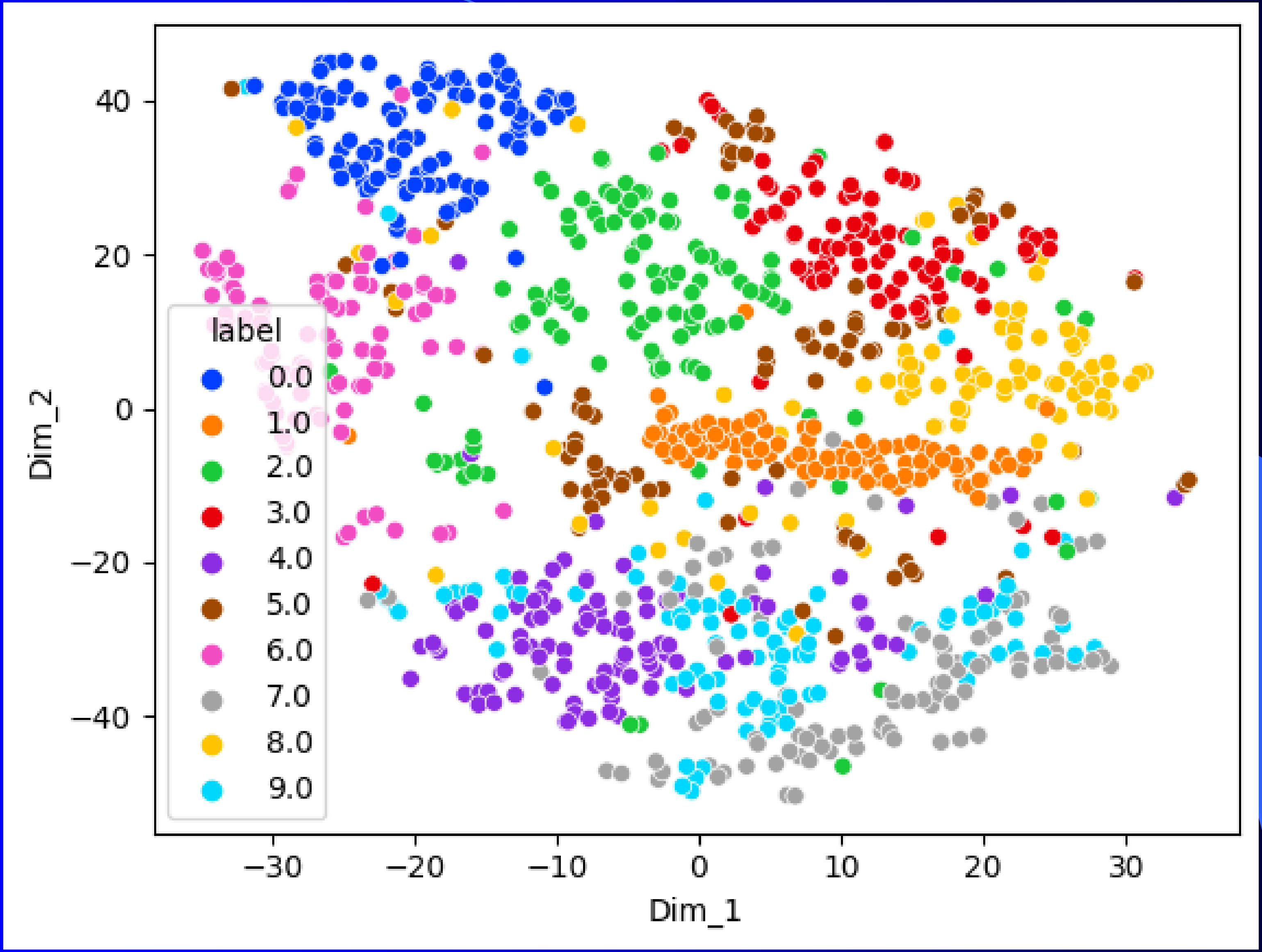
- Goal is to keep the distance within the perpexity cluster the same
  - Minimize the distance difference locally but not globally
  - The distances within a perplexity group are meaningful, but not between perplexity groups
- t-SNE is a stochastic algorithm and won't produce the same output twice
  - Can set the random seed
- To add more data, you need to re-run the algorithm from the beginning

t-SNE Perplexity



Original Perplexity=2 Perplexity=30 Perplexity=100


PCA
t-SNE
Digit
Digit
0
1
2
3
4
5
6
7
8
9


label
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
Dim_2
Dim_1
40
20
0
−20
−40
−30
−20
−10
0
10
20
30

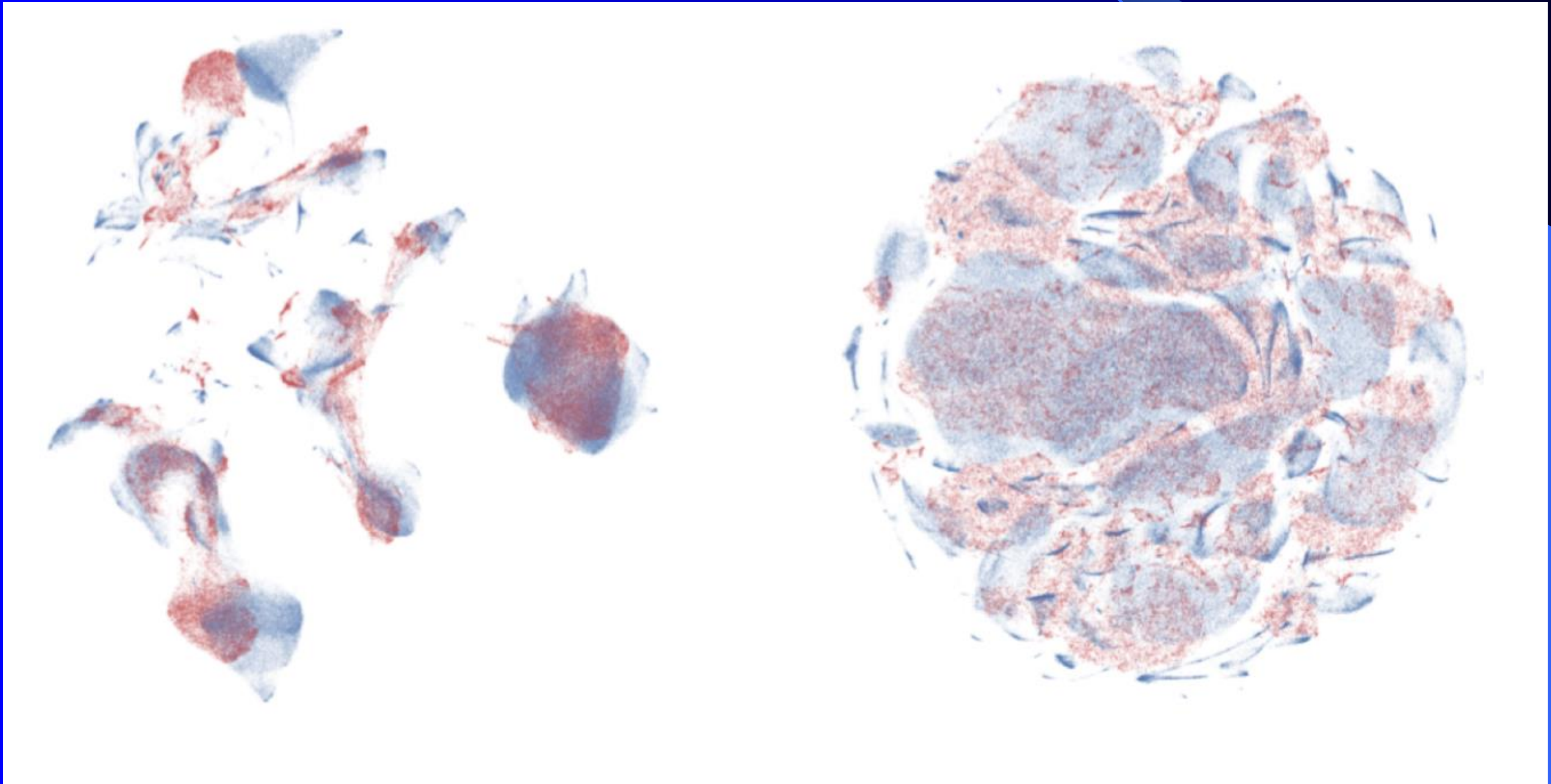
UMAP

- Uniform Manifold Approximation and Projection
- Similar to t-SNE but assumes the data is distributed on a manifold in less dimensions
  - Topological structures in multidimensional space
- Faster than t-SNE
- Can preserve more global structure than t-SNE
- Can allow new data to be added to the projection

- Instead of perplexity
  - The expected number of nearest neighbors (very similar to perplexity)
  - Minimum distance to pack the points which are close together.
    - Points are connected if they are within the minimum distance

UMAP

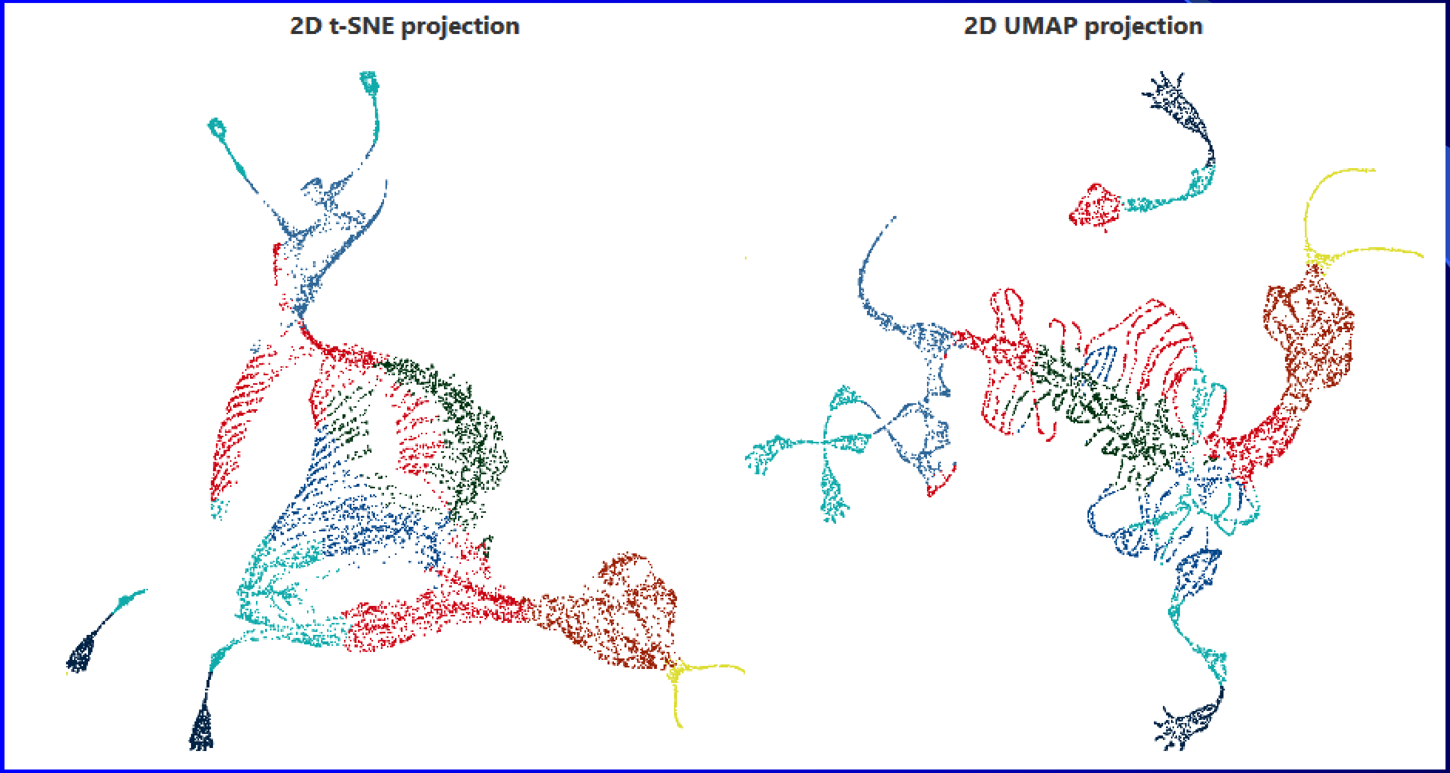
- Learns the global data structure
- Doesn't depend on random initial values
- Can recreate the lower dimensional embedding regardless of the dataset size



(a) UMAP (b) t-SNE

UMAP

- Can be great for the right data



3D mammoth skeleton projected into 2D

tSNE:	Perplexity 2000	2h 5min
UMAP:	Nneigh 200, mindist 0.25,	3min

Summary

- Dimensionality reduction techniques can be very beneficial
- PCA yields linear transformations → interpretable
- t-SNE and UMAP are non-linear
  - Can provide better dimensionality reduction especially for visualization
- Can be used in concert
  - PCA first to reduce dimensionality with linear transformations
  - t-SNE or UMAP to further reduce

  - Very good for visualization