Lecture 13:

Unsupervised Learning: Dimensionality

Week of February 27, 2023

University California, Berkeley

Machine Learning Algorithms

MSSE 277B, 3 Units

Spring 2023

Prof. Teresa Head-Gordon

Departments of Chemistry, Bioengineering, Chemical and Biomolecular Engineering

Summary Previous Lecture

We considered K-Means, Hierarchical, Density and Parameteric GMM. Summarizing the pros and cons:

- Hierarchical clustering scales quadratically $O(n^2)$ while K-Means clustering is linear i.e. O(n)
- K Means clustering is stochastic while results are deterministic in Hierarchical clustering.
- K Means work swell when shape of clusters is hyperspherical, whereas GMM works for non-spherical shapes
- K Means and GMM require prior knowledge of K. For hierarchical clustering the number of clusters is a cut in the dendrogram.
- DBSCAN doesn't need specification of cluster number and is by design less sensitive to outliers than other approaches
- All have hyperparameters to tune, but DBSCAN can be very sensitive to them

Dimensionality Reduction

Purpose of Today's Lecture: In ML classification problems, there are often too many features on the basis of which the final classification is done. The higher the number of features, the harder it gets to visualize the training set. Often most of the features are correlated, and hence redundant storage requirements are large.

Dimensionality reduction is the process of reducing the number of features by obtaining a set of "principal" feature variables. It can be divided into feature selection and feature extraction.

Feature Selection: only keeping the most relevant features from the original dataset

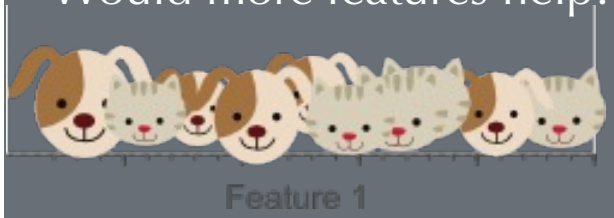
Feature Extraction: By finding a smaller set of new features, each being a combination of the input variables, containing basically the same information as the original input feature set

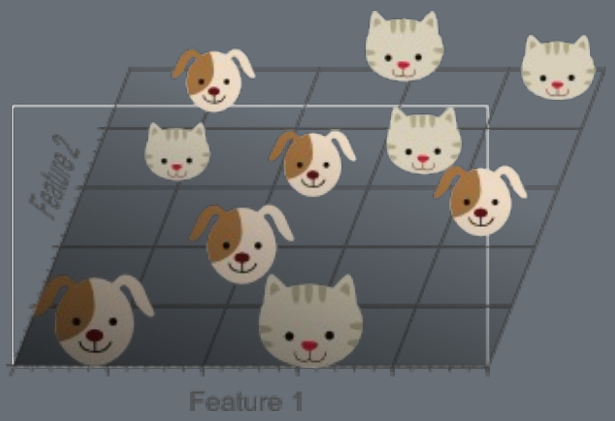
Curse of Dimensionality

Given a data set matrix of N × p, where N represents number of observations and p represents number of features.

I.e. N observations (rows) with p features (columns): $\bar{\bar{X}} = \{\vec{x}_1, \ldots \vec{x}_p\}$.

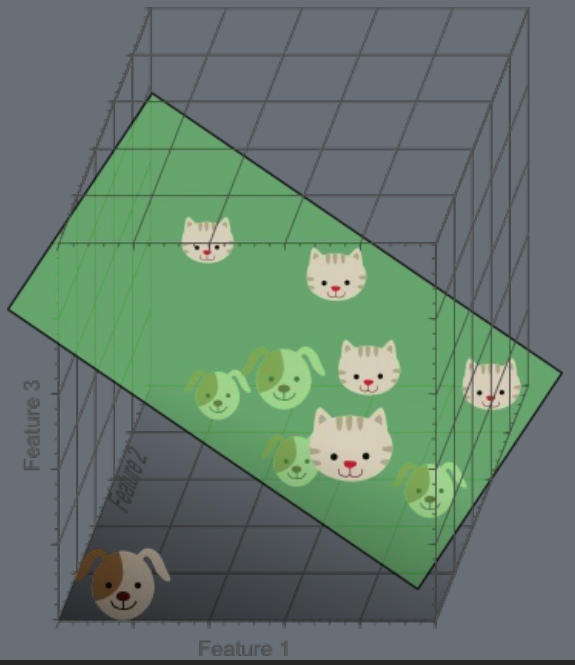
This one feature does not define a good decision line that separates cats from dogs. Would more features help?

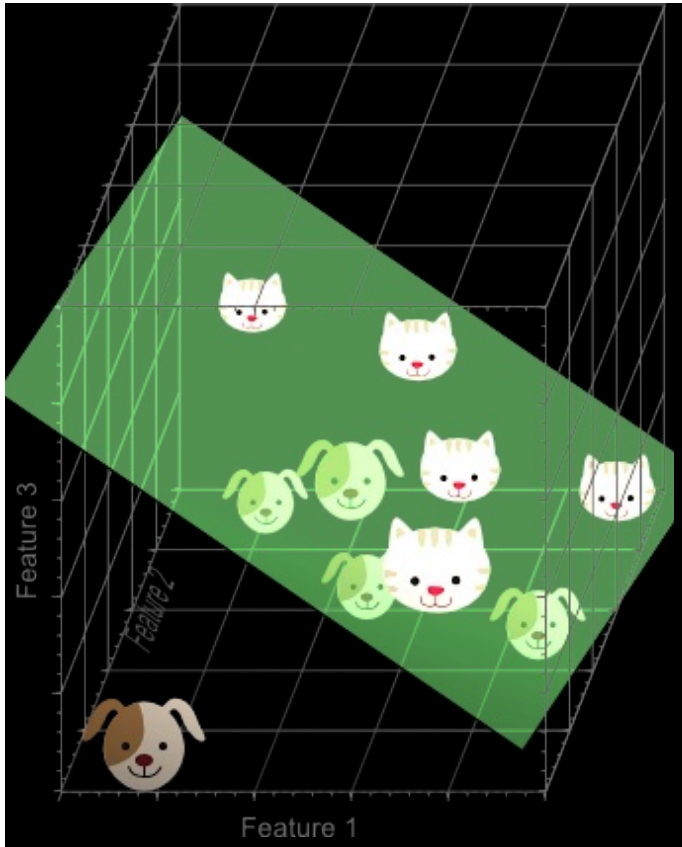


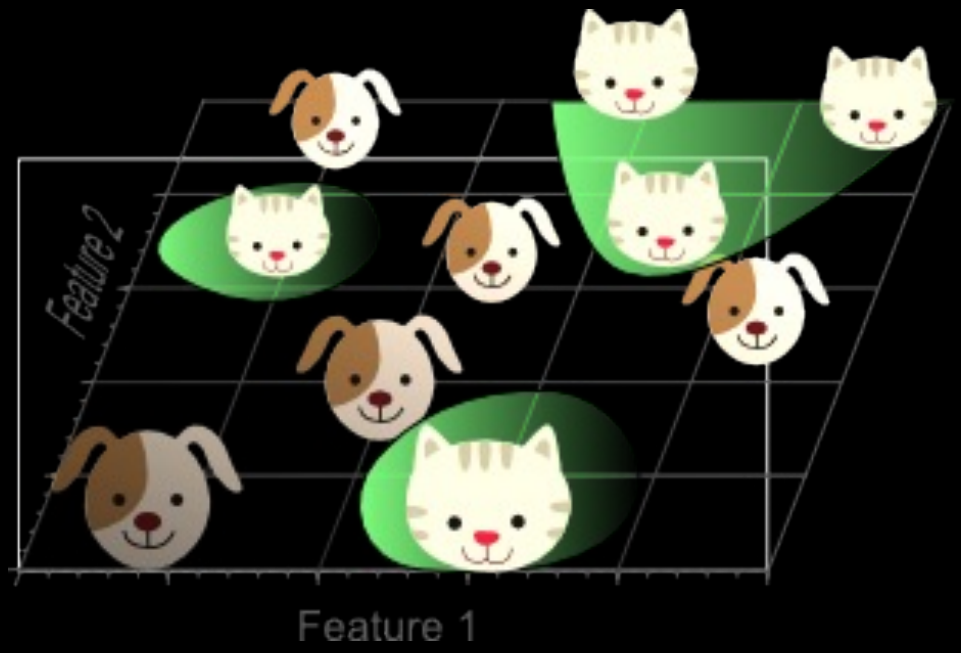


Two features does does help define a better decision plane that separates cats from dogs, but mis-classifications occur. Would more features help?

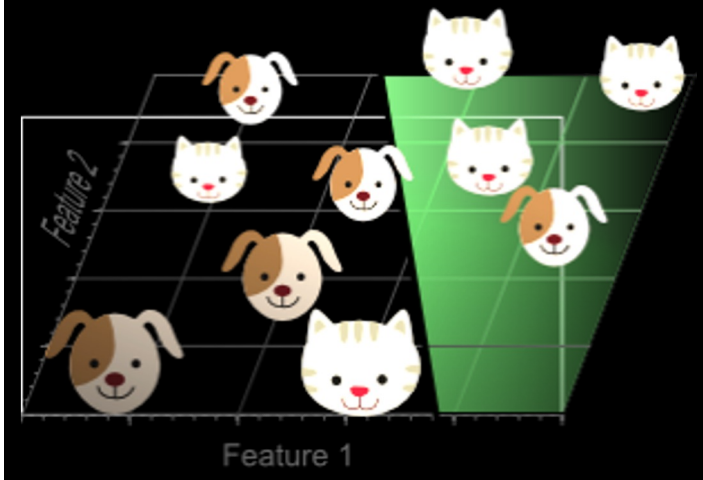
Three features defines a hyperplane that separates cats from dogs!







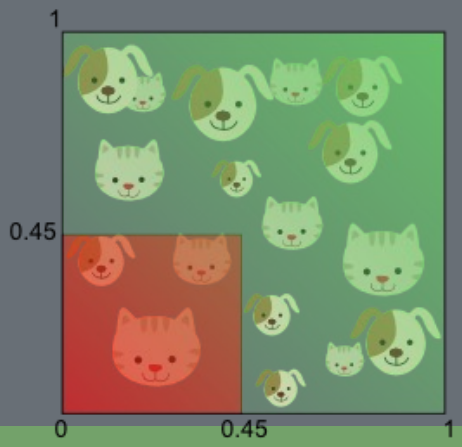
But in unseen data (we want to generalize!), the original 2 features actually performs better!

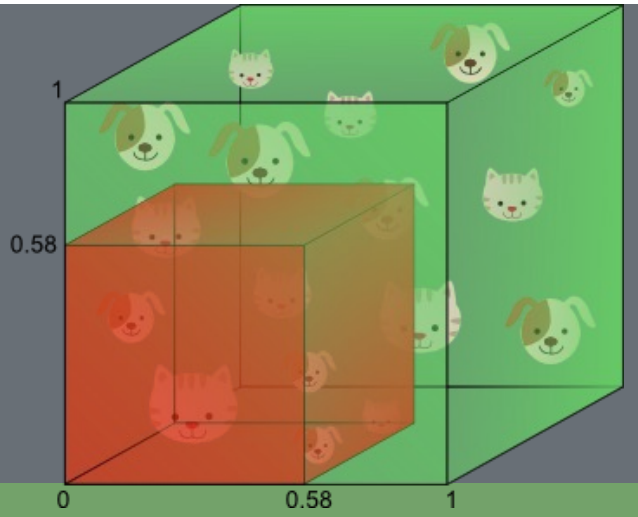


But when projected from 3D classification to 2D, the linear classifier in 3D is a non-linear classifier in 2D. The 3D classifier learned how to classify singles incidences of cats and dogs and not general features. Why is this?

Curse of Dimensionality

To estimate a function of several variables to a given degree of accuracy, we have two choices



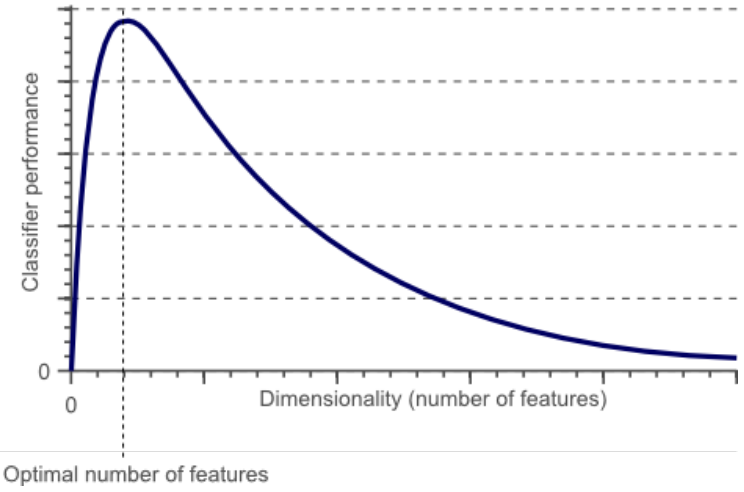


(1) Sample size N needs to grow with the number of variables, p.

The curse of dimensionality, *d*, is that data requirements grow exponentially with *d*. It is not tractable to generate very large N

(2) Increase number of features, p, at fixed density of points N.

We see that data coverage is getting sparser as p-dimensions increases. Its easier to define a decision plane in high dimensions, but at the risk of overfitting



Therefore, we mitigate this problem and indirectly reduce the value of N is to significantly reduce the data dimensionality, p.

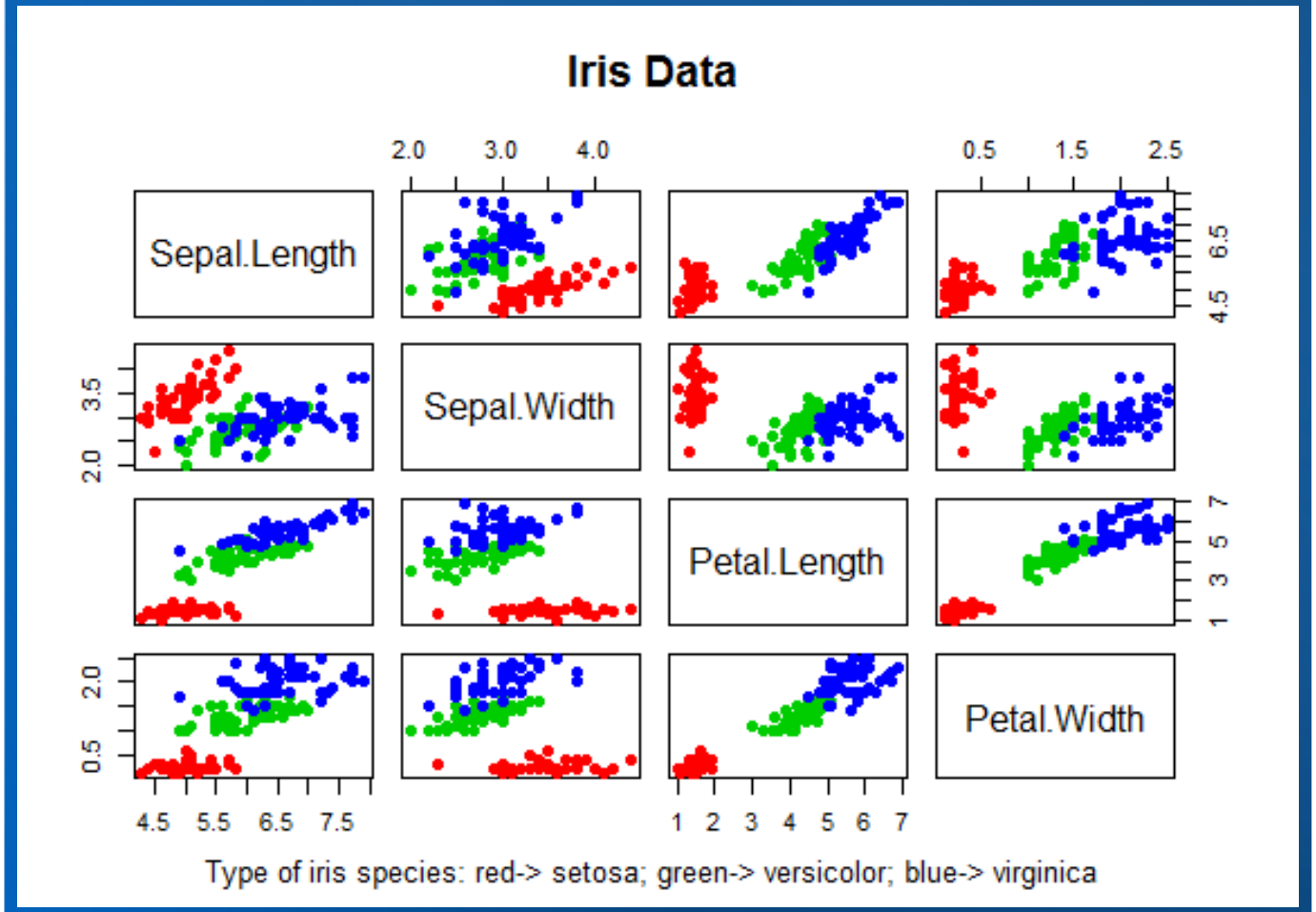
Curse of Dimensionality

Note that species setosa (red dots) is well separated in all 2D plots, but versicolor (blue dots) and virginica (green dots) are harder to separate.

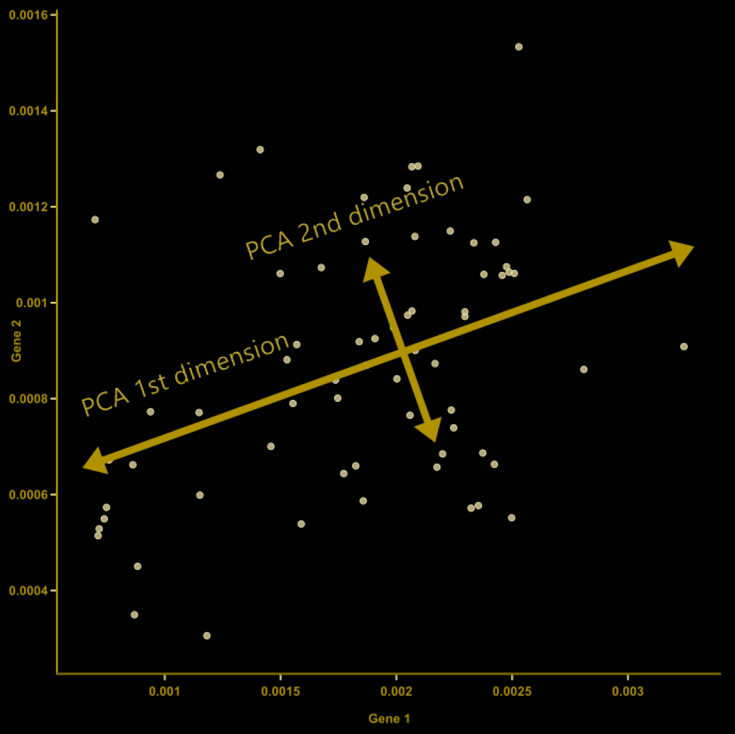
The features of versicolor and virginica are correlated and can't differentiate between the two iris species.

PCA finds a new set of dimensions such that all the dimensions are orthogonal, and ranked according to the greatest variance of data along them.

In HW you will work with the Iris data set which uses different measurements or features (sepal length and width and petal length and width) to differentiate between species of irises: setosa, versicolor, and virginica. The data is comprised of N=150 flowers (50 entries from each of 3 species of iris)



Principal Component Analysis



Principal components analysis (PCA) takes the original data and computes new data that are along directions of greatest variation.

Given data matrix of N × p, when p > 2, there are p(p-1)/2 scatter plots to analyze the variable relationship (which we showed for the Iris data set). PCA can help with this.

Most of variation is along direction of the long diagonal line (first principal direction)

Smaller amount of variation is in direction of short line (second principal direction orthogonal to first)

PCA finds linear combinations of original data that explain as much variation in the data as possible. Let's see this mathematically

Principal Component Analysis

Start with N observations (rows) with p features (columns): $\bar{\bar{X}} = \{\vec{x}_1, \ldots \vec{x}_p\}$. Because we are focused on *variation* in the data, we normalize data in all of the columns to have a mean 0.

$$\frac{1}{N}\sum_{\mu}^{N} x_{\mu j} = m_j \qquad \vec{x}_j' = \vec{x}_j - \vec{1}m_j$$

And $\bar{\bar{X}}' = \{\overrightarrow{x'}_1, \ldots \overrightarrow{x'}_p\}$, let's drop the prime (′) understanding that it is our new centered data. The co-variance matrix elements are averaged over all samples

$$\sigma_{jk} = \frac{1}{N}\sum_{\mu}^{N}(x_{\mu j} - m_j)(x_{\mu k} - m_k)$$

In general matrix form

$$\bar{\bar{C}} = \frac{1}{N}\left(\bar{\bar{X}} - \vec{m}\right)^T\left(\bar{\bar{X}} - \vec{m}\right) \rightarrow \frac{1}{N}\bar{\bar{X}}^T\bar{\bar{X}}$$

and recognizing $\bar{\bar{X}}^T\bar{\bar{X}}$ as that of the mean centered data

Principal Component Analysis

We now want to calculate eigenvectors and eigenvalues of $\bar{\bar{X}}^T \bar{\bar{X}}$ (p x p matrix) whose eigen decomposition is

$$\bar{\bar{X}}^T \bar{\bar{X}} \rightarrow \bar{\bar{P}} \bar{\bar{\Lambda}} \bar{\bar{P}}^T$$

The eigenvectors $\vec{p}$ are unit vectors that are all orthogonal to each other (very much like H-orthogonal but for P!) representing the direction of the largest variance of the data.

The eigenvalues $\bar{\bar{\Lambda}}$ are a diagonal matrix with eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_p$ on the diagonal and zeros off-diagonal, and represent the magnitude of the variance in the corresponding directions.

This is solved by standard singular valued decomposition (SVD). We want to maximize the variance along new directions $\vec{p}$ under the condition they remain orthogonal to each other.

$$\vec{p}_1 = \max var(\bar{\bar{X}} \vec{p}_1) \; subject \; to \; \vec{p}_1{}^T \vec{p}_1 = 1$$

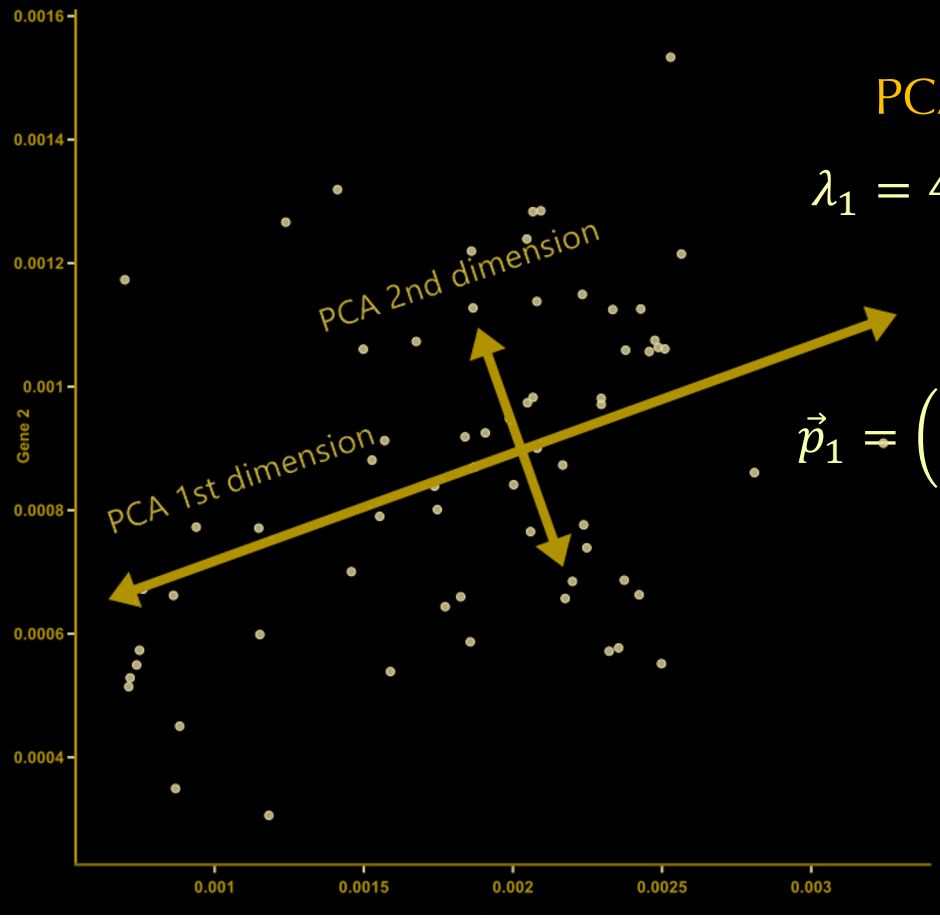
Principal Component Analysis

Sort the eigenvalues from largest to smallest. In doing so, the corresponding eigenvectors of $\bar{\bar{P}}$ must be sorted accordingly. This sorted matrix of eigenvectors define the directions of greatest variance to least variance by column

Original Data

$$\vec{m} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\bar{\bar{C}} = \begin{pmatrix} 2 & 2 \\ 2 & 3 \end{pmatrix}$$



PCA transformed data

$\lambda_1 = 4.562$ $\lambda_2 = 0.438$

$$\vec{p}_1 = \begin{pmatrix} 0.615 \\ 0.788 \end{pmatrix} \qquad \vec{p}_2 = \begin{pmatrix} -0.788 \\ 0.615 \end{pmatrix}$$

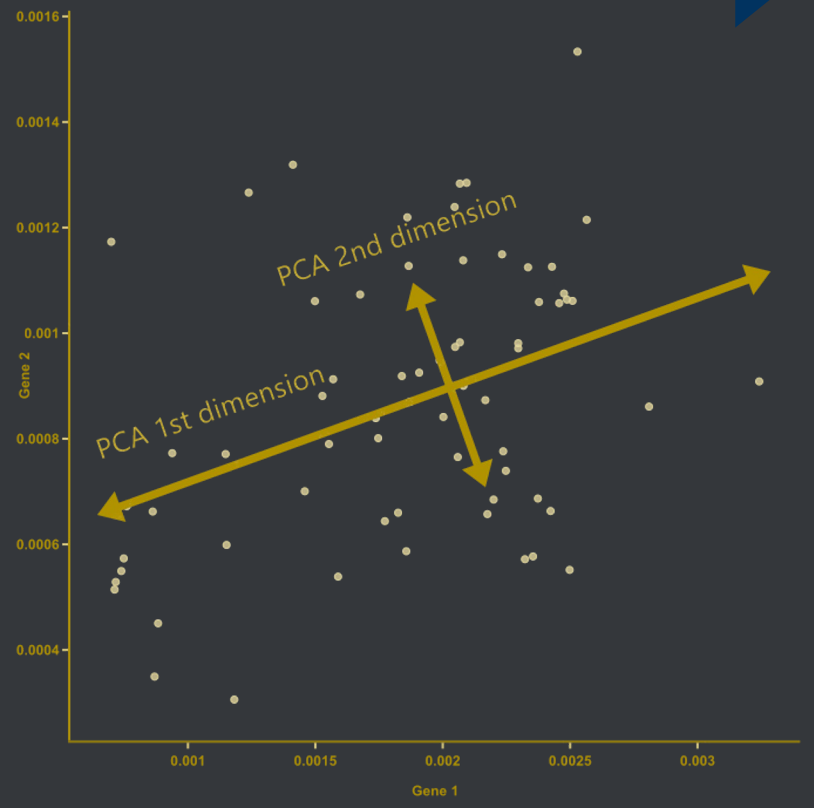
Principal Component Analysis

Calculate $\bar{\bar{X}}^* = \bar{\bar{X}}\bar{\bar{P}}$ (which is size N x p). This new data matrix, $\bar{\bar{X}}^*$, is a centered version of $\bar{\bar{X}}$.

But now each i observation is a combination of the original features/ variables, where the weights are determined by the p eigenvector elements ϕ.

$$x_i^* = \phi_{11}x_{i1} + \phi_{21}x_{i2} \ldots + \phi_{p1}x_{ip}$$

Finally, we need to determine how many features to keep versus how many to drop. This is the goal of data reduction techniques.

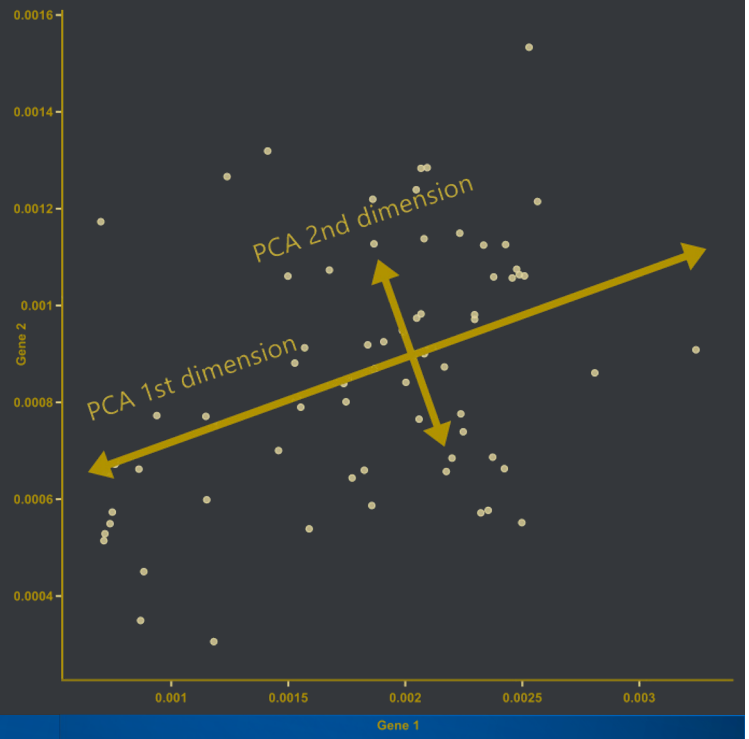


There are three common methods to determine the "feature extraction", which will then provide the data reduction.

Because each eigenvalue is a measure of the importance of its corresponding eigenvector, the proportion of total variance explained is the sum of the eigenvalues of the features you keep divided by the sum of the eigenvalues of all features.

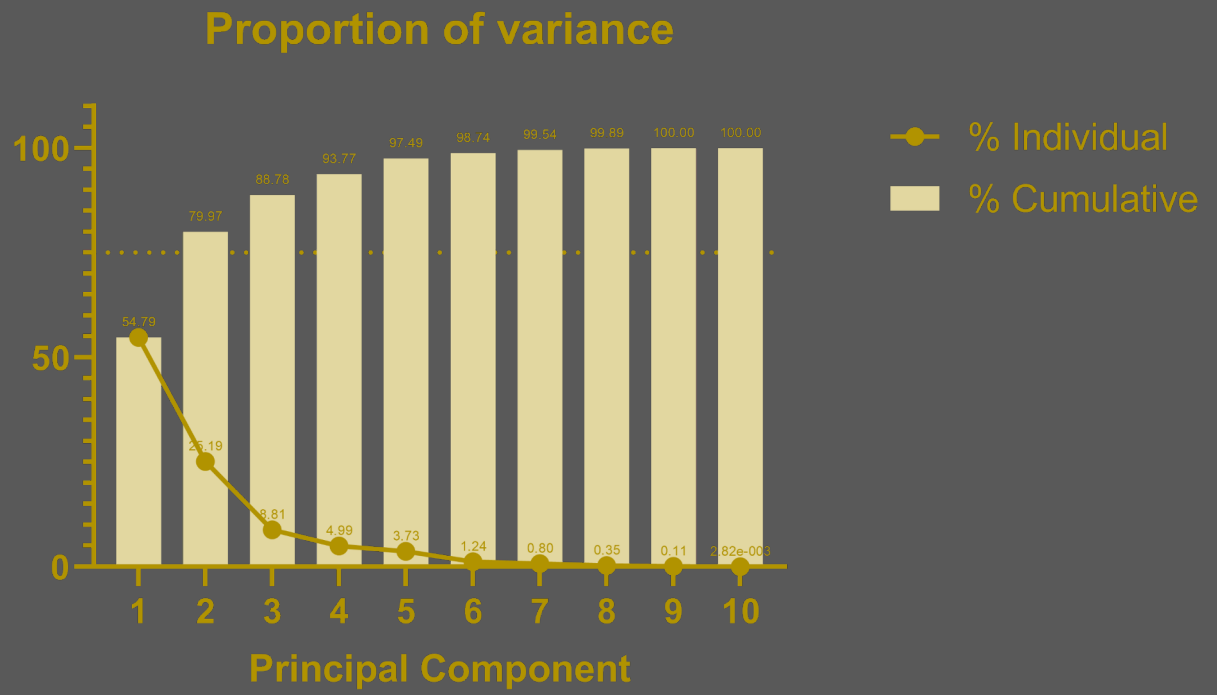
$$\% \, variance = \frac{\sum_j^{r<p} \lambda_j}{\sum_j^{p} \lambda_j}$$

Method 1: Arbitrarily select how many new features we want to keep. Often this is driven to visually represent things in much fewer dimensions.



Principal Component Analysis

Method 2 (Cumulative): Pick a threshold (say 75% of the variability), and add features until you hit that threshold to determine r.

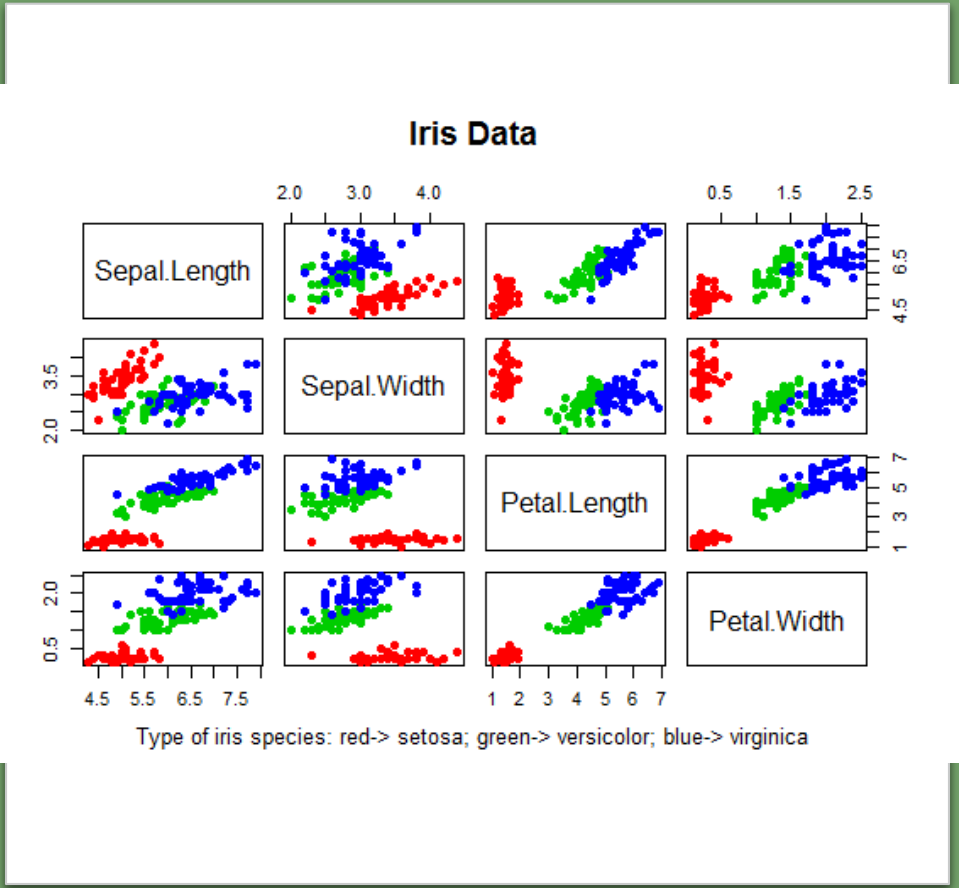


Method 3 (Scree plot): Beyond 3 principle components the gain is becoming more negligible as each feature is added. This bend in the curve – fast gain followed by flat – is the cutoff (although in this case only explains ~75% of the variance!)

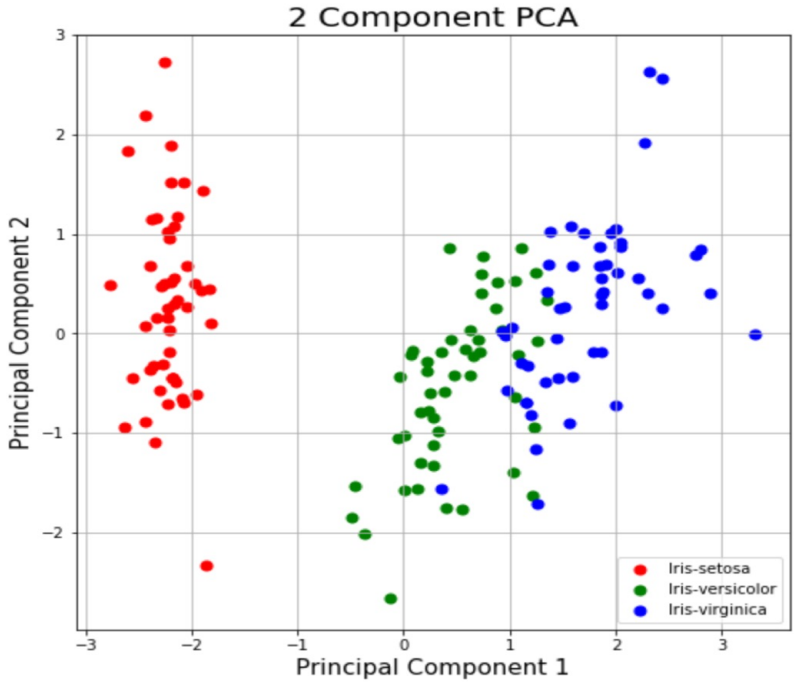
Principal Component Analysis

We can now drop the “least important” new variables while still retaining the most valuable parts of all of the feature variables!

- Pros: each of the “new variables” after PCA are all independent of one another
- Cons: the new variables don’t have the same meaning as the feature variables (loss of interpretability)



Versicolor and Virginica are much better separated.



Principal Component Analysis

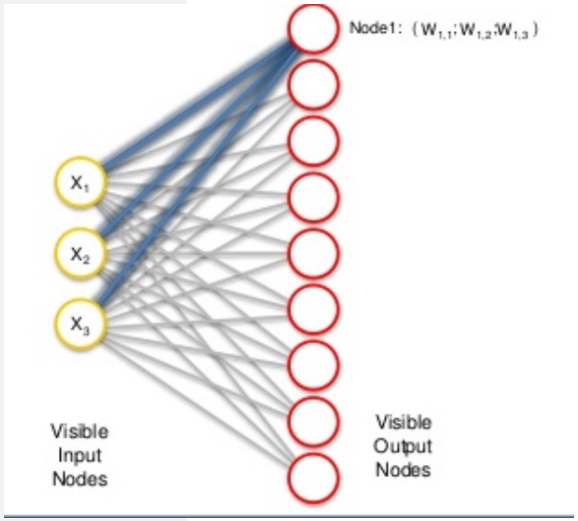
We have previously looked at artificial neural networks with supervised training using error correction (i.e. back-propagation), i.e. there is a target output for each input pattern, and the network learns to produce the required outputs (labeled data).

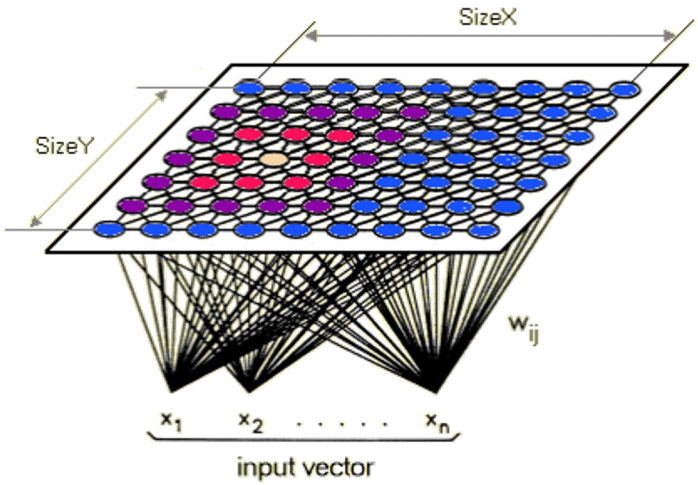
We can also use ANNs for unsupervised training, in which the networks learn to form their own classifications of the training data without labelled data. It assumes that class membership is defined by the input patterns sharing common features, and that the network will be able to identify those features across the range of input patterns.

Self-organized maps (SOMs) are a class of unsupervised ANNs based on competitive learning, in which the output neurons compete amongst themselves to be activated based on its similarity to the input, with the result that the neurons are forced to organize themselves into a lower dimensional space (typically 2D)

Self Organizing Maps

Self Organizing Maps





For example, the three input nodes x_i where $i = 1,2,3$ represent three columns (features) in the dataset, but each of these columns can contain thousands of rows of observations. The input nodes are connected to k output nodes by weights, w_{ki}. Beyond this, the supervised ANN's and unsupervised SOMs differ substantially

- Output nodes in SOM are usually 1D or 2D.
- For supervised learning ANNs we multiply input node values by edge weights and use an activation function. For SOMs there is no activation function and weights are coordinates of outputs
- SOM compresses N x p inputs into output node that carries p weights to accomplish the data reduction

Basic ideas of the SOM algorithm

Initialization: All the connection weights are initialized with small random values.

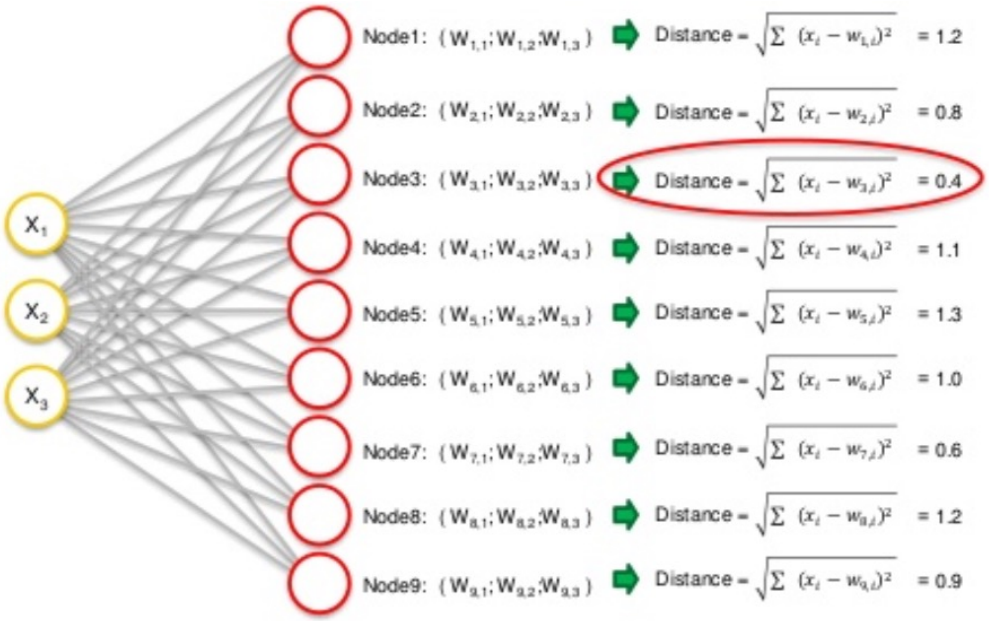
Competition: For each input pattern, the neurons compute their respective values of a discriminant function which provides the basis for competition. The particular neuron with the smallest value of the discriminant function is declared the winner.

Cooperation: The winning neuron determines the spatial location of a topological neighbourhood of excited neurons, thereby providing the basis for cooperation among neighbouring neurons.

Adaptation: The excited neurons decrease their individual values of the discriminant function in relation to the input pattern through suitable adjustment of the associated connection weights. Thus the winning neuron will more likely be activated to the subsequent application of a similar input pattern.

Self Organizing Maps

Self Organizing Maps: Initialization and Competition



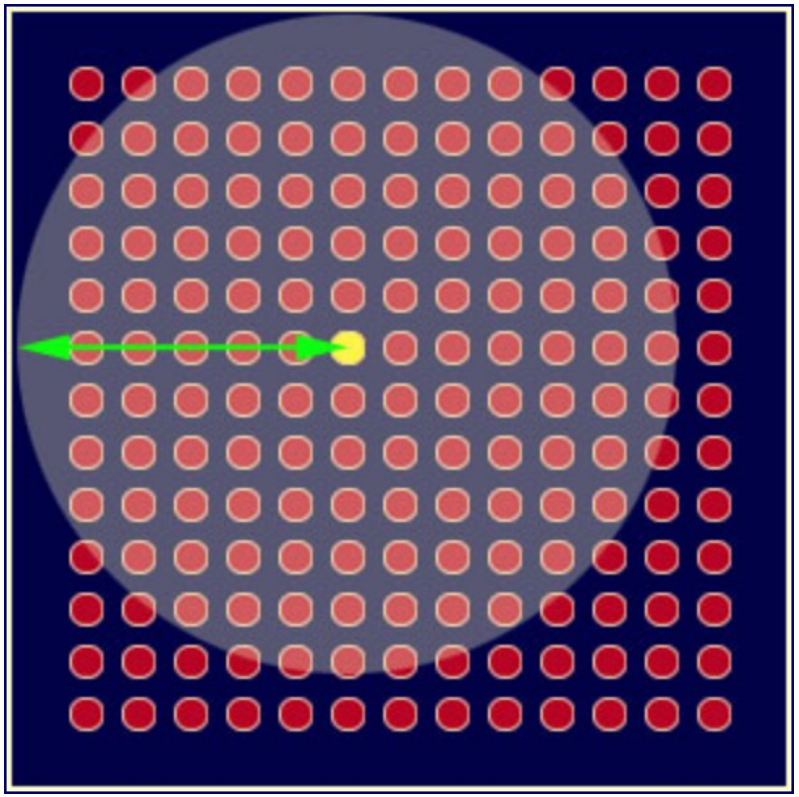
We start by initializing the weights randomly for the k output nodes.

Then for each of the observations (rows) in our dataset, we evaluate the Euclidean distance between the input node and the output node through its weight vector

$$d_k = \sqrt{\sum_i^N (x_i - w_{ki})^2}$$

In this example, output node k=3 is the closest match to the input example. This node is called the best matching unit (BMU).

Self Organizing Maps: Cooperation



Above is how the BMU looks on the SO map. It and its neighbors will be activated, i.e. the SOM procedure will update the weights in the local region of the BMU with the assumption that features are correlated in the BMU neighborhood.

The area of the BMU neighborhood (originally the entire map) shrinks over time heuristically according to an update formula such as an exponential decay function:

$$r_{BMU}(t) = r_{lattice} exp\left(\frac{-t}{\tau}\right)$$

where τ denotes an adjustable time constant and t is the current iteration of the SOM loop.

Self Organizing Maps: Adaptation

If a node is within the neighborhood of the BMU (or the BMU itself) then each element of its weight vector is adjusted as follows.

$$w_i(t+1) = w_i(t) + \gamma(t)\big(x_i(t) - w_i(t)\big)$$

i.e. we are adjusting weights to look more like the input/feature space. Furthermore the learning rate changes in time

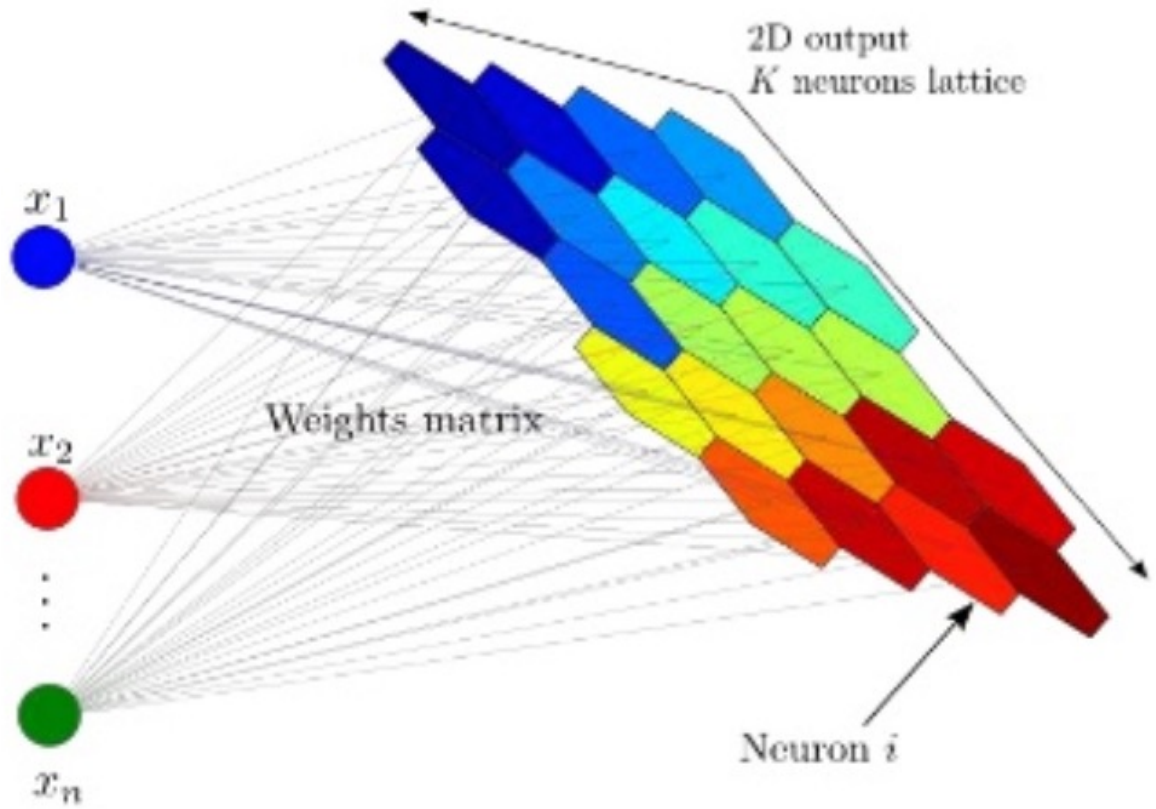
$$\gamma(t) = \gamma_0 exp\left(\frac{-t}{\tau}\right)$$

where $\boldsymbol{\tau}$ denotes another (same?) adjustable time constant and $\boldsymbol{t}$ is the current iteration of the SOM loop. Of course the BMU should have the larger adjustment than nodes distant from it, so we adjust the weight update to reflect that through Gaussian decay from BMU

$$w_i(t+1) = w_i(t) + \mathcal{N}_i(t)\gamma(t)\big(x_i(t) - w_i(t)\big)$$

$$\mathcal{N}_i(t) = exp\left(\frac{-d_{i,bmu}^2}{r_{bmu}^2(t)}\right)$$

Self Organizing Maps



The process repeats on the next observation, and the SOM completes when the feature map stops changing.

The SOM organizes input space so we can see how features are correlated (hence can try to do feature selection) or how data is clustered.

A well converged map with several clusters trained on 48 distinct colors

The choice of data reduction is then up to you! (Cool colors and warm colors only? Reds, greens and blues?, etc)