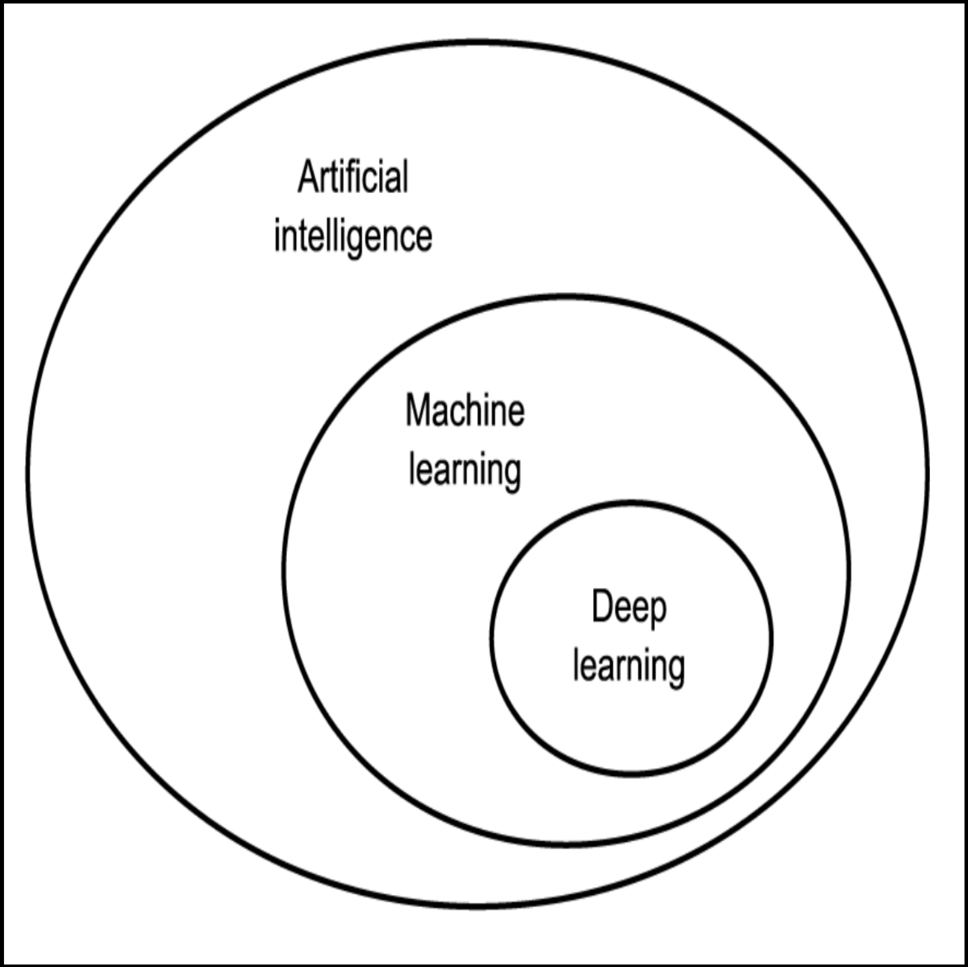
Basic Overview of Machine Learning Methods

JEFF GILL
Distinguished Professor
Department of Government, Department of Mathematics & Statistics
Member, Center for Neuroscience and Behavior
Founding Director, Center for Data Science
American University

Background: What is Artificial Intelligence?

- AI was actually born in the 1950s and used to solve toy problems for the most part.
- This led to symbolic AI, up to the 1980s, including the develop-opmet of expert systems.
- In one sense AI can be thought of as the real-world application of algorithms produced by machine learning although there are other sources.



Background: What is Machine Learning?

- Machine Learning is a collection of tools mostly for classification and prediction.
- Most of these you already know or are close to something you already know, and the vocabulary is simply different (logit).
- The focus is mostly on prediction, regression, and classification.
- The term is not as new as one would think (Samuel, *IBM Journal of Research and Development*, 1959).
- Modern definition: "A computer program is said to learn from experience E with regard to some class of tasks T and performance measure P, if its performance at tasks T, as measured by P, improves with experience E." (Mitchell, *Machine Learning*, 1997).
- Common applications: credit card data analysis, speech recognition, text analysis, fraud detection, self-driving cars, website ads, and many more.
- A lot of these applications were previously addressed with rigid rule-based systems.

Supervised Learning

- There is a target variable Y that we want to predict given feature variables $\mathbf{X}$ by learning a function such that $F(\mathbf{X}) = Y$.
- When Y is interval measured this is regression and when Y is categorical this is classification.
- A key goal is to find the *best* F possible to estimate/predict future (unseen) data.
- This is a different approach than classical statistical inference that focuses on the data at hand.
- Both SL and regular statistical regression specification select the covariates with human control.

Unsupervised Learning

- Here there is an identified target Y.
- The goal is to identify groupings within the data in different ways.
- Cluster identification, principal components analysis, are the classic examples.
- Sometimes there are covariates involved but often not.
- Note that this dichotomy is not strict and there are lots of tools in between supervised and unsupervised learning: weakly supervised and hybrids/combinations.

More Differences from Classic Statistical Inference

- In most social science settings the first emphasis is on selecting a parsimonious set of control variables ("under the horizon") and set of theory-based variables ("over the horizon").
- But in ML the typical strategy is to have start with many explaining variables and reduce the number with a hold-out/test strategy to winnow the number down possibly.
- There usually is not a concern about how many are left, unlike regular statistical inference (AIC, BIC, DIC, etc.).
- A primary reason for this is that ML is most often used with big data so the $p \gg n$ is unlikely to be a concern.
- A lot of the work in ML is done to "process" the data with transformations of the data as part of the fitting process and then have a relatively simple $F()$.
- This is a different trend than statistics.

Vocabulary (this is important!)

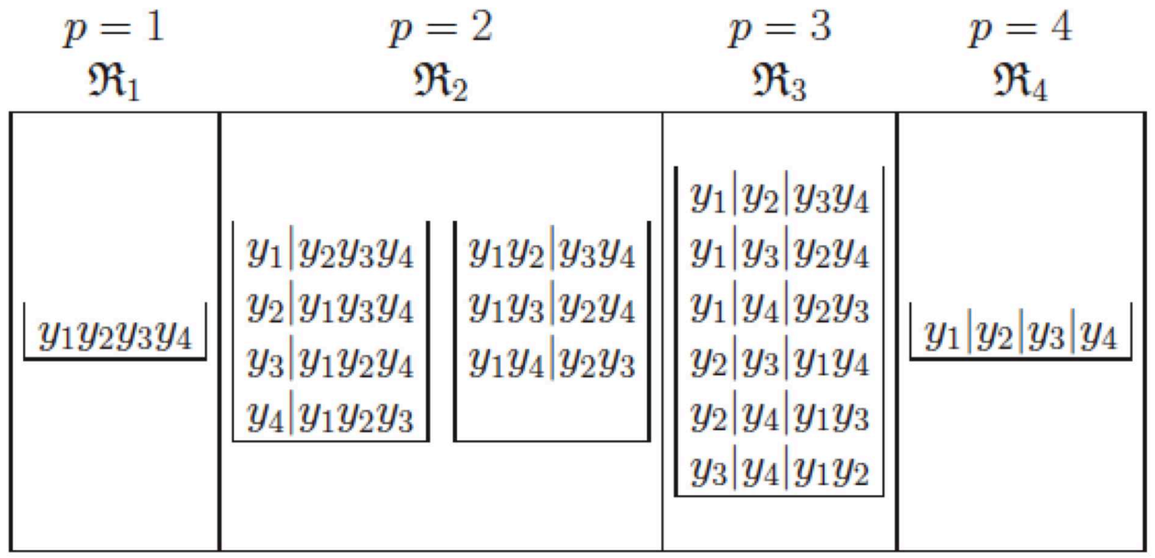
- Features: explanatory variables.
- Labels: outcome variables.
- Examples: the subjects (data).
- Learning: training with the data, estimating a function, building a model (the process).
- Deep Learning: a hierarchical process wherein complex representations (models) are created in the algorithm from simple representations in a dynamic process ($F()$ is created from combining many far simpler $f()$ functions in multilevel structure).
- Overfitting: a sin in ML, the model is too closely aligned with a single dataset, including it's error component, ruining generality (application to future datasets).
- Underfitting: the model does not adequately explain the underlying phenomenon.

Clustering

- The most common unsupervised learning method: asserting/inferring substantive groups in the data cases.
- This is hardly new/modern, e.g. statisticians have been arguing about the number of clusters in the "galaxy" data for at least 50 years.
- Most often the determination of clusters is done spatially with respect to fixed data points based on a distance measure like Euclidean, Manhattan, or Mahalanobis ($d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})\text{COV}(\mathbf{x} - \mathbf{y})}$).
- Not always so though, e.g. text analysis, genetics, social network analysis.
- Two different settings: users specifies the number of clusters in advance (relatively easy), or the number of clusters is not known in advance (relatively hard).
- We often get different cluster arrangements with different algorithms.

Example of Cluster Configurations

Partitions for $n = 4$



- Four Cluster Classes, Five Configuration Classes
- The number of configuration classes in each cluster class is $b(n, p)$
  - $b(n, p)$ = partitions of the integer n into p components ≥ 1
  - $b(4,1) = 1, \quad b(4,2) = 2, \quad b(4,3) = 1, \quad b(4,4) = 1,$

Notes On Cluster Configurations

- So for this $n = 4$ illustration in each of the cluster classes, $p = (1, 2, 3, 4)$, there are: $b(n, p) = (1, 2, 1, 1)$ configuration classes, and $(1, 7, 6, 1)$ partition types.
- The number of partition types, for a given n and p is a Stirling number of the second kind from:

$$\left\{ \begin{matrix} n \\ p \end{matrix} \right\} = \frac{1}{p!} \sum_{j=0}^{p} (-1)^{p-j} \binom{p}{j} j^n.$$

- In the example there are 15 total possible partitions (models), the Bell number for $n = 4$ from:

$$B_n = \frac{1}{e} \sum_{j=0}^{\infty} \frac{j^n}{j!}$$

- We connect these because a Bell number can be expressed as the sum of Stirling numbers of the second kind:

$$B_n = \sum_{p=0}^{n} \left\{ \begin{matrix} n \\ p \end{matrix} \right\}$$

- For a fixed m, the number of configuration classes $b(n, m)$ grows as $\frac{n^{m-1}}{m!(m-1)!}$ with increasing n.

A Typology of Clustering

- Agglomerative Clustering: at the beginning each data point is its own cluster then the algorithm combines the points into clusters (e.g. K-Means).
- Divisive: at the beginning all the data points are in the same cluster then the algorithm breaks the apart into a number of clusters (e.g. Mean Shift).
- We can actually get very different cluster configurations based on which algorithm is used.

K-Means Clustering Algorithm

- This the oldest and most common method.
- It is basically a variance minimizer in the ANOVA sense (sum of squares).
- Guaranteed to converge (always "works").
- You have to assume the number of clusters in advance.
- Process:
  - distribute k centroids in the data space, randomly, uniformly, or purposefully
  - each data point gets assigned to the nearest centroid creating clusters
  - within each cluster move the centroid to the spatial mean of the data points and calculate the variance around this point
  - since the centroids have moved repeat the last two steps
  - continue until the centroids do not move anymore.
- Users often do this process multiple times with different starting points to gain confidence.
- K-Means is very fast and therefore very useful with big data.

K-Means in R

```
# GET FISHER'S IRIS DATA
data(iris)
head(iris, 10)
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
3           4.7         3.2          1.3         0.2  setosa
4           4.6         3.1          1.5         0.2  setosa
5           5.0         3.6          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
7           4.6         3.4          1.4         0.3  setosa
8           5.0         3.4          1.5         0.2  setosa
9           4.4         2.9          1.4         0.2  setosa
10          4.9         3.1          1.5         0.1  setosa

# REMOVE SPECIES LABEL
iris2 <- iris[,-5]
```

K-Means in R

```
# LOAD PACKAGES
library(ClusterR); library(cluster)

# RUN K-MEANS, centers is k, nstarts is the number of random starts
set.seed(1234)
( k.means.out <- kmeans(x=iris2, centers = 3, nstart = 25) )

K-means clustering with 3 clusters of sizes 50, 62, 38

Cluster means:
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1     5.006000    3.428000     1.462000    0.246000
2     5.901613    2.748387     4.393548    1.433871
3     6.850000    3.073684     5.742105    2.071053
```

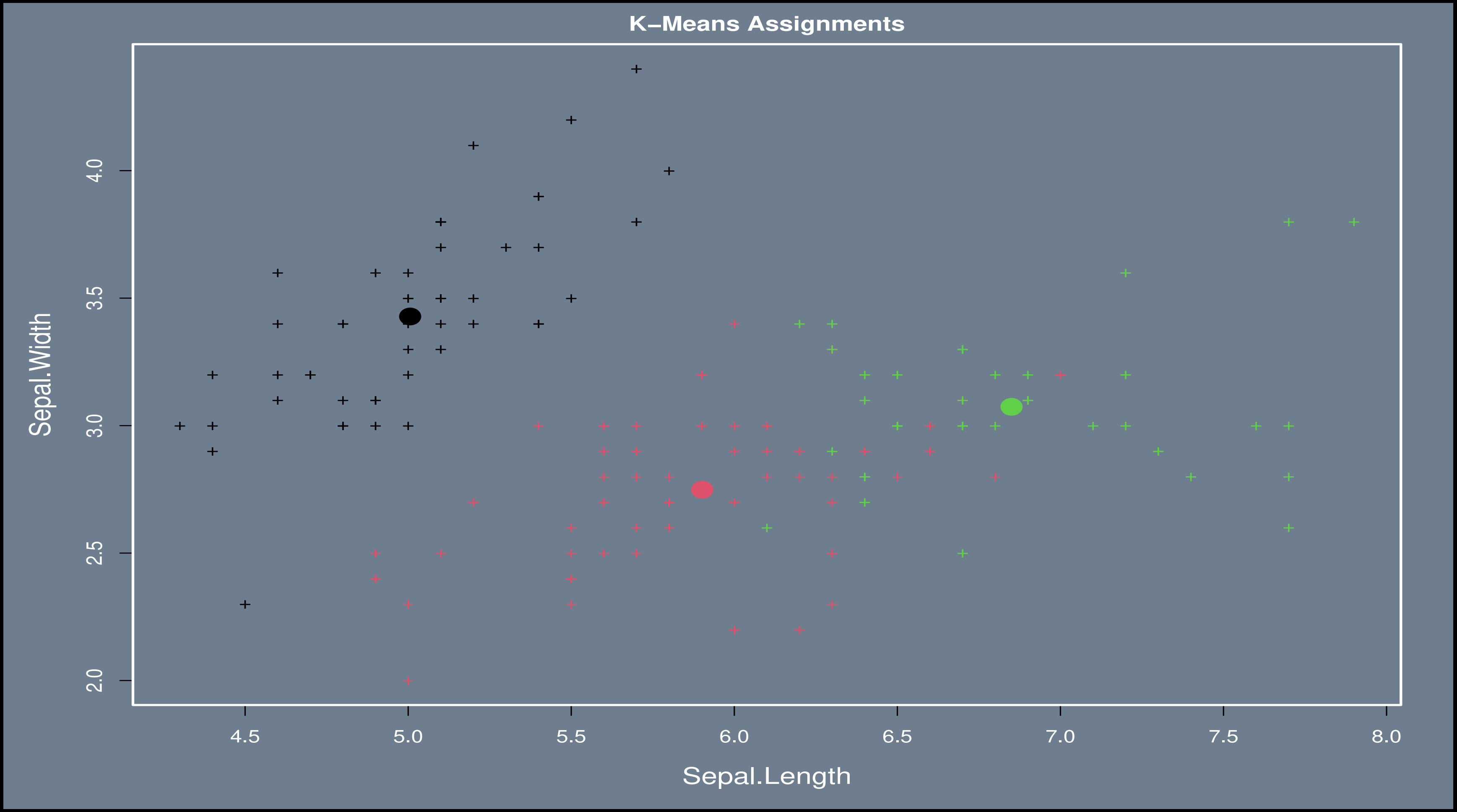
K-Means in R

```
Clustering vector:
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [35] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [69] 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2
[103] 3 3 3 3 2 3 3 3 3 3 3 2 2 3 3 3 3 2 3 2 3 2 3 3 2 2 3 3 3 3 3 2 3 3
[137] 3 3 2 3 3 3 2 3 3 3 2 3 3 2

Within cluster sum of squares by cluster:
[1] 15.15100 39.82097 23.87947
 (between_SS / total_SS =  88.4 %)

# CONFUSION MATRIX
table(iris$Species, k.means.out$cluster)
              1  2  3
  setosa     50  0  0
  versicolor  0 48  2
  virginica   0 14 36
```

K-Means in R

```
# PLOT 1
par(mfrow=c(1,1),mar=c(5,5,2,2),lwd=2,col.axis="white",col.lab="white",
    col.main="white", col.sub="white", col="white",bg="slategray", cex.lab=1.3)
plot(iris2[c("Sepal.Length", "Sepal.Width")], col = k.means.out$cluster,
     main = "K-Means Assignments", pch="+")
points(k.means.out$centers[, c("Sepal.Length", "Sepal.Width")],
       col = 1:3, pch = 19, cex = 2)
```

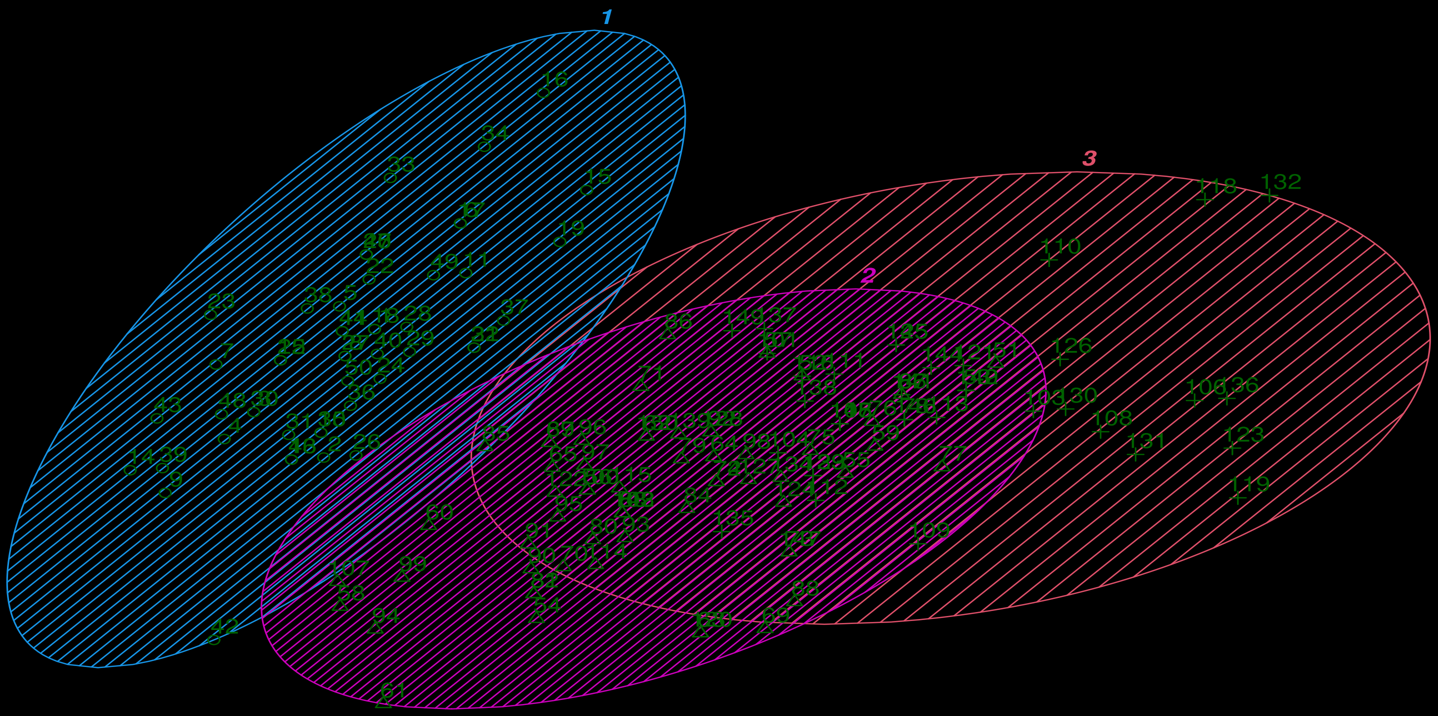
K-Means Plot 1



K-Means in R

```
# PLOT 2
clusplot(iris2[, c("Sepal.Length", "Sepal.Width")], k.means.out$cluster,
         lines = 1, shade = TRUE, color = TRUE, labels = 2, plotchar = TRUE,
         span = FALSE, main = paste("K-Means Assignments"),
         xlab = "Sepal.Length", ylab = "Sepal.Width")
```

K-Means Plot 2



Limitations of K-Means

- The need to choose k in advance.
- A heavy dependence on initial centroid locations.
- Performs poorly when real clusters are of very different sizes as it makes cluster sizes equal.
- Sensitivity to spatial outliers.
- Does poorly in high dimensions.
- Assumes that the variance within each cluster is the same.
- Will find clusters even if there aren't any in the data.

Mean Shift Clustering

- A somewhat different approach that starts with the points rather than the centroids and is based on density.
- Instead of k we have to define a bandwidth parameter h.
- Algorithm:
  - create a circular window around every data point of radius h so every data point defines a cluster
  - get the mean position of the points inside each window, make this a centroid
  - move the center of the window to this centroid, meaning all of the points in the original circular windows get a new assignment around this centroid
  - apply a kernel density estimator (smoother) to create a response surface around the new centroids
  - repeat until convergence to a finite number of clusters.
- Eventually all points in the same cluster will end up with basically the same steps towards convergence.
- This is conditional on a reasonable bandwidth value, which must be chosen carefully.

Mean Shift in R

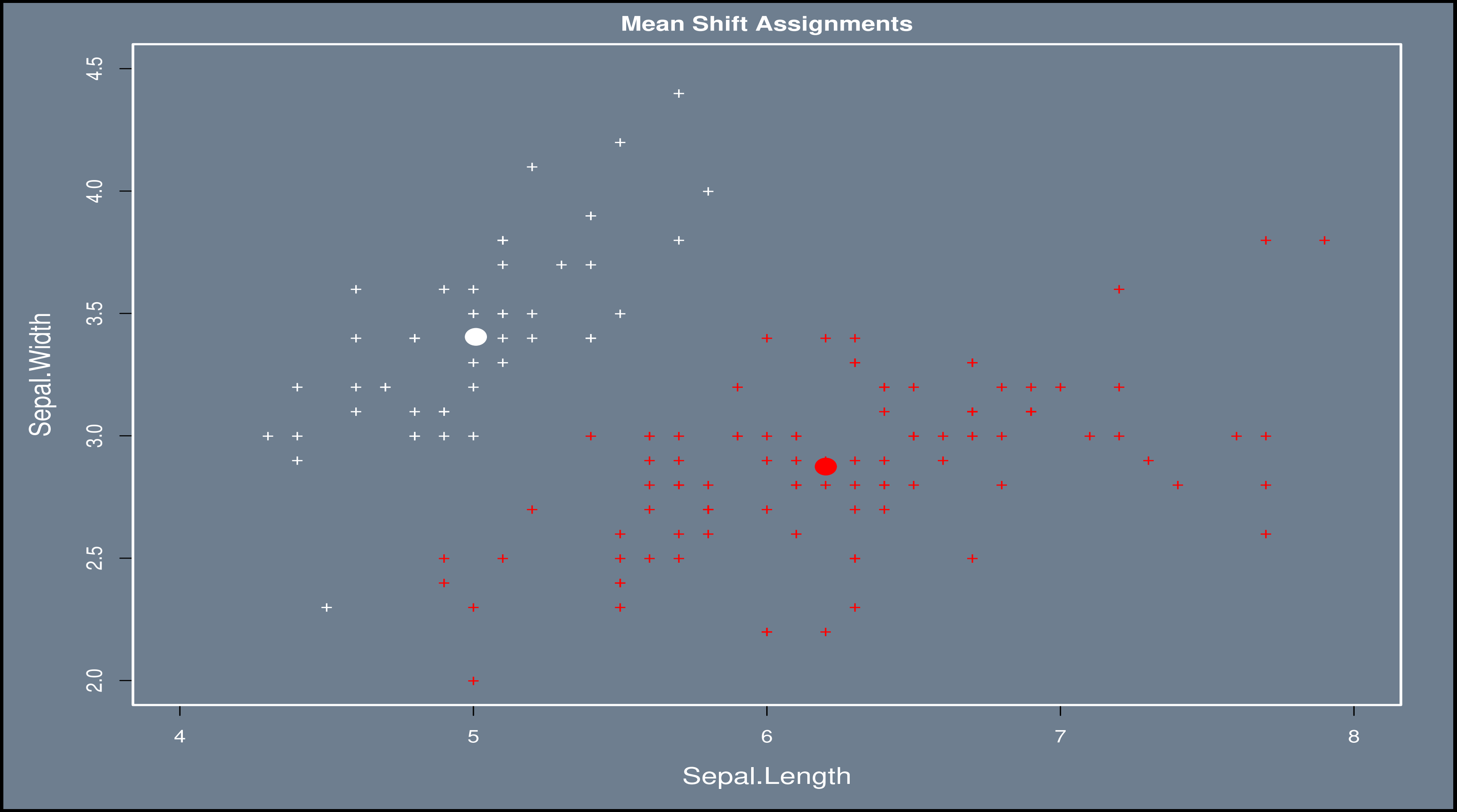
```
library(meanShiftR)
( ms.out <- meanShift(queryData = as.matrix(iris2),trainData = as.matrix(iris2),
    algorithm="LINEAR") )
t(ms.out$assignment)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20]
[1,]    1    1    1    1    1    1    1    1    1     1     1     1     1     1     1     1     1     1     1     1
     [,21] [,22] [,23] [,24] [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33] [,34] [,35] [,36] [,37] [,38] [,39]
[1,]     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1
     [,40] [,41] [,42] [,43] [,44] [,45] [,46] [,47] [,48] [,49] [,50] [,51] [,52] [,53] [,54] [,55] [,56] [,57] [,58]
[1,]     1     1     1     1     1     1     1     1     1     1     1     2     2     2     2     2     2     2     2
     [,59] [,60] [,61] [,62] [,63] [,64] [,65] [,66] [,67] [,68] [,69] [,70] [,71] [,72] [,73] [,74] [,75] [,76] [,77]
[1,]     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
     [,78] [,79] [,80] [,81] [,82] [,83] [,84] [,85] [,86] [,87] [,88] [,89] [,90] [,91] [,92] [,93] [,94] [,95] [,96]
[1,]     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2     2
     [,97] [,98] [,99] [,100] [,101] [,102] [,103] [,104] [,105] [,106] [,107] [,108] [,109] [,110] [,111] [,112]
[1,]     2     2     2      2      2      2      2      2      2      2      2      2      2      2      2      2
     [,113] [,114] [,115] [,116] [,117] [,118] [,119] [,120] [,121] [,122] [,123] [,124] [,125] [,126] [,127] [,128]
[1,]      2      2      2      2      2      2      2      2      2      2      2      2      2      2      2      2
     [,129] [,130] [,131] [,132] [,133] [,134] [,135] [,136] [,137] [,138] [,139] [,140] [,141] [,142] [,143] [,144]
[1,]      2      2      2      2      2      2      2      2      2      2      2      2      2      2      2      2
     [,145] [,146] [,147] [,148] [,149] [,150]
[1,]      2      2      2      2      2      2
```

Mean Shift in R

```
# PLOT
par(mfrow=c(1,1),mar=c(5,5,2,2),lwd=2,col.axis="white",col.lab="white",
    col.main="white", col.sub="white", col="white",bg="slategray", cex.lab=1.3)
plot(iris2[c("Sepal.Length", "Sepal.Width")][ms.out$assignment==1,], col = "white",
     xlim=c(4,8), ylim=c(2,4.5),main = "Mean Shift Assignments", pch="+")
points(iris2[c("Sepal.Length", "Sepal.Width")][ms.out$assignment==2,], col = "red",
    pch="+")
points(ms.out$value[c(1,150),], col = c("white","red"), pch = 19, cex = 2)
```

Mean Shift Plot



Limitations of Mean Shift

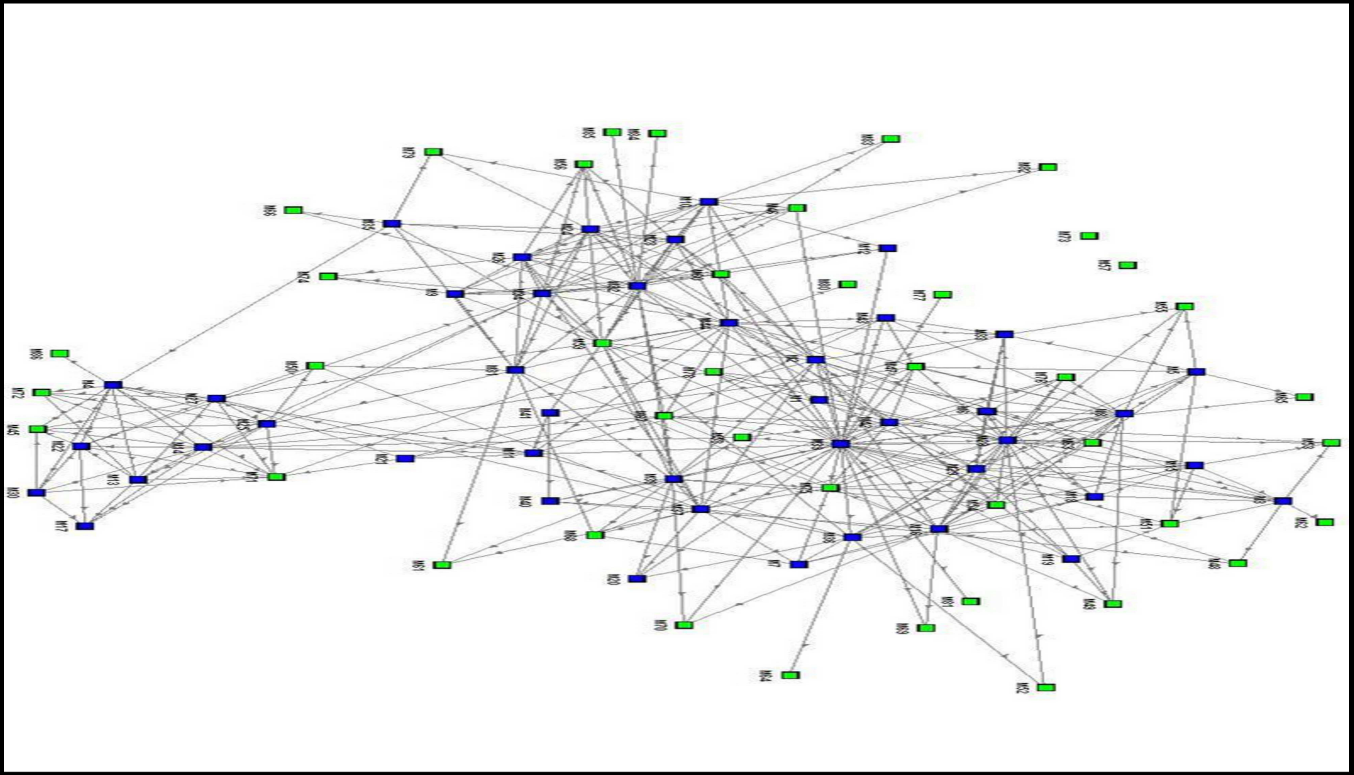
- Due to the cycling through the points and then cycling through the clusters on each iteration it can be very compute-intensive, especially with big data.

- The choice of h is critical: too small and convergence may not happen, too large and distinct actual clusters in the data get be merged.

- Cluster distinctions are "sharp" and not overlapping, which makes this method popular in computer vision, but that may be less appropriate in the social sciences.

Spectral Clustering

- Spectral clustering works very differently than what we've seen so far by valuing "connectivity" over convex boundaries.
- Spectral clustering can find non-convex clusters.
- It uses a graph (network) setup where the data are nodes and the edges are similarity.



Spectral Clustering

- Algorithm:
  - Create the similarity matrix from the distance between each pair of data points: which is square with values $n(n-1)/2$ that we care about.
  - Perform *graph partitioning* whereby the edges between clusters are given low weights and the edges within clusters are given high weights.
  - Compute the eigenvalues and eigenvectors.
  - The k eigenvectors are treated as data and supplied to k-means to create the clusters for the original data.
- SC performs well but is slower with big data.
- There are also some important selection parameters and algorithmic decisions to be made.

Spectral Clustering Technical Details

- We have n points $x_{\cdot}$ in $\Re^p$, where p is the dimension of the data (two so far in the slides).
- And d_{ij} is the distance between x_i and x_j (generally Euclidean but any type).
- As a similarity graph $\mathbf{S}$ use the *radial-kernel grain matrix*, whose elements are defined by:

$$s_{ij} = \exp\left[\frac{-d_{ij}}{c}\right]$$

  where positive c is a (selection) scale parameter.

- Alternatively we could use as a similarity graph the *mutual K-nearest neighbor graph* that starts with defining N_K as the symmetric set of nearby points: s_{ij} is positive in N_K if i is in the *K-nearest neighbors* of j. Then assemble the sets of nearest neighbors and assign them the edge weight w_{ij} and the excluded relationships are assigned zero.
- There are many other alternatives.

Spectral Clustering Technical Details

- Generally the result of the similarity process with its edge weights is called an adjacency matrix and denoted $\mathbf{W}$ with elements w_{ij}.
- Vertex i has *degree* that is the sum of its weights:

$$g_i = \sum_{i=1:n,\neg i} w_{ij}.$$

- Now define the diagonal matrix G that collects $g_i, i = 1 \ldots n$.
- The *standardized graph Laplacian* is given by:

$$\mathbf{L} = \mathbf{I} - \mathbf{G}^{-1}\mathbf{W}$$

- Now find the m smallest eigenvectors (selection parameter) of $\mathbf{L}$ corresponding to the m smallest eigenvalues: $\mathbf{E}_{n\times m}$.
- Cluster the rows of $\mathbf{E}_{n\times m}$ with K-Means (or some alternative).

Eigen-Analysis of Matrices

- Every $p \times p$ matrix $\mathbf{X}$ has p scalar values, $\lambda_i, i = 1, \ldots, p$, such that

  $$\mathbf{X}\mathbf{e}_i = \lambda_i \mathbf{e}_i$$

  for some corresponding vector $\mathbf{e}_i$.

- In this *decomposition*, λ_i is called an eigenvalue of $\mathbf{X}$ and $\mathbf{e}_i$ is called an *eigenvector* of $\mathbf{X}$.

- These eigenvectors are linearly independent.

- These are also called the *characteristic roots* and *characteristic vectors* of $\mathbf{X}$, and the process is also called *spectral decomposition*.

- The full *eigen-decomposition* of the original square matrix is given by: $\mathbf{X} = \mathbf{E}(\lambda \mathbf{I})\mathbf{E}^{-1}$ where $\mathbf{E}$ is a matrix with the eigenvectors down columns.

- The eigenvalues and eigenvectors are only guaranteed to be real-valued if the original square matrix is symmetric.

- The *characteristic equation* is given by: $\text{diag}((\mathbf{X} - \boldsymbol{\lambda}\mathbf{I})\mathbf{E}) = \mathbf{0}$.

- The eigenvalues and eigenvectors are found by solving the characteristic equation.

Basic Eigenanalysis

- A (contrived) symmetric square matrix $\mathbf{X}$ is given by: $$\mathbf{X} = \begin{bmatrix} 1.000 & 0.880 & 0.619 \\ 0.880 & 1.000 & 0.716 \\ 0.619 & 0.716 & 1.000 \end{bmatrix}.$$

- Using R (note the descending order of the eigenvalues given):

```
X <- matrix(c(1,0.88,0.619,0.88,1,0.716,0.619,0.716,1),3,3)
( eigen.X <- eigen(X) )
eigen() decomposition
$values
[1] 2.4820708 0.4100160 0.1079132

$vectors
          [,1]       [,2]       [,3]
[1,] 0.5850593  0.5127477  0.6283274
[2,] 0.6071390  0.2367286 -0.7585128
[3,] 0.5376688 -0.8252571  0.1728089
```

- Note: in the eigenvector matrix returned by R the eigenvectors are the *columns*.

Basic Eigenanalysis

- Returning to the Eigen-analysis definition:

$$\mathbf{X}\mathbf{e}_i = \lambda_i \mathbf{e}_i$$

- Test for $i = 1$in $\mathbf{X}\mathbf{e}_i = \lambda_i \mathbf{e}_i$:

```
cbind(
    X %*% eigen.X$vectors[,1],
    t(eigen.X$values[1] %*% eigen.X$vectors[,1])
)
```

```
             [,1]     [,2]
    [1,] 1.452159 1.452159
    [2,] 1.506962 1.506962
    [3,] 1.334532 1.334532
```

Eigenanalysis Uniqueness

- Eigenvalues and eigenvectors are associated.

- For each set of eigenvectors of a given matrix $\mathbf{X}$ there is exactly one corresponding eigenvalue vector such that
$$\lambda = \frac{\mathbf{e}'\mathbf{X}\mathbf{e}}{\mathbf{e}'\mathbf{e}}.$$

```
diag(t(eigen.X$vectors) %*% X %*% eigen.X$vectors)
    /diag((t(eigen.X$vectors) %*% eigen.X$vectors))
[1] 2.4820708 0.4100160 0.1079132
> eigen.X$values
[1] 2.4820708 0.4100160 0.1079132
```

- But for each eigenvalue vector of the matrix there is an infinite number of eigenvectors, all determined by scalar multiplication.

- Meaning that if $\mathbf{e}$ is an eigenvector corresponding to the eigenvalue λ, then $s\mathbf{e}$ is also an eigenvector corresponding to this same eigenvalue where s is any nonzero scalar.

Eigenanalysis General Properties

- The number of nonzero eigenvalues is the rank of the $\mathbf{X}$.
- The sum of the eigenvalues is the trace of $\mathbf{X}$.
- The product of the eigenvalues is the determinant of $\mathbf{X}$.
- A matrix is singular if and only if it has a zero eigenvalue (and thus the determinant is zero).
- If there are no zero-value eigenvalues, then the eigenvectors determine a basis for the space determined by the size of the matrix ($\mathfrak{R}^2$, $\mathfrak{R}^3$, etc.).
- Symmetric nonsingular matrices have eigenvectors that are perpendicular to each other (orthogonal).

Spectral Clustering in R

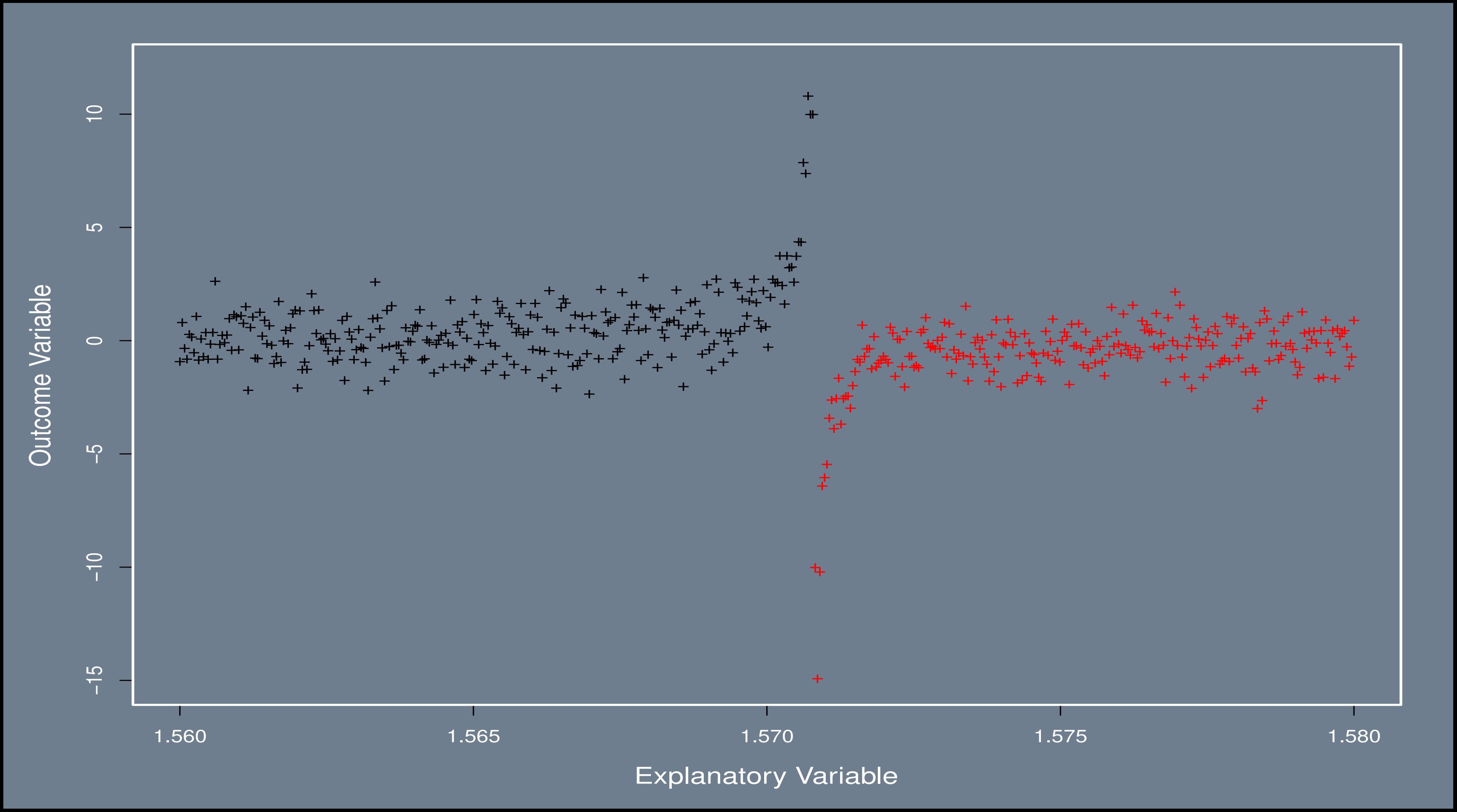
```
library("speccalt")
kern.partitioning <- local.rbfdot(iris2) # USES A KDE TO DEFINE SIMILARITY THRESHOLD
speccalt(kern.partitioning) # AUTOMATIC CHOICE OF m CLUSTERS
```

```
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
 [61] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[121] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
speccalt(kern.partitioning,5) # CHOOSE c
```

```
  [1] 2 5 5 5 2 2 2 2 5 5 2 2 5 5 2 2 2 2 2 2 2 2 2 2 2 5 2 2 2 5 5 2 2 2 5 5 2 2 5 2 2 1 5 2 2 5 2 5 2 2 3 3 3 1 3 1 3 1 3 1
 [61] 1 3 1 3 1 3 1 1 1 1 3 3 3 3 3 3 3 3 3 1 1 1 1 3 1 3 3 1 1 1 1 3 1 1 1 1 1 3 1 1 4 3 4 3 4 4 1 4 3 4 4 3 4 3 3 4 3 4 4 1
[121] 4 3 4 3 4 4 3 3 3 4 4 4 3 3 3 4 4 3 3 4 4 4 3 4 4 4 3 3 4 3
```

Contrived Very Difficult Case



Contrived Very Difficult Case

```
ruler <- seq(1.56,1.58,length=500)
y1 <- tan(ruler)/1000 + rnorm(500,0,1.0)
y1[y1 > 20] <- 10; y1[y1 < -20] <- -10
synth.dat <- cbind(ruler,y1)
kern.partitioning <- local.rbfdot(synth.dat)
sc.fit <- speccalt(kern.partitioning,2)

par(mfrow=c(1,1),mar=c(5,5,2,2),lwd=2,col.axis="white",col.lab="white",
    col.main="white", col.sub="white", col="white",bg="slategray", cex.lab=1.3)
plot(synth.dat[1:270,],pch="+",col="black", xlim=c(1.560,1.580), ylim=c(-15,12),
    xlab="Explanatory Variable",ylab="Outcome Variable")
points(synth.dat[271:500,],pch="+",col="red")
```

Principal Components Analysis

- Principal components analysis (PCA) is a means of data reduction with big data through rotation in the sample space of observations.
- Principal components are the orthogonal directions in which the data varies and reduce the size of the data down to a set of vectors that explain the variance: summarize the relationships among a set of features with a smaller set of linear combinations.
- This is very useful in big data analysis where $p \gg n$.
- PCA re-expresses the variability of the data such that the total amount of variance is preserved but:
  - Axes are enumerated in descending order of variance explained. That is, the first dimension explains the most variance, the second dimension explains the second-most variance, and so on.
  - The new axes are uncorrelated with each other: they are orthogonal.
  - If there exists correlation in the original data then it is expressed as zero length along some dimensions after the rotation of axes.

Principal Components Analysis

- If there are p explanatory variables in the original data and there is some correlation between these variables, then PCA produces a rotation in the p dimensions except that some number of these, q, will be of zero length, where the magnitude of q indicates the extent of the correlation between variables and $p - q$ indicates the extent of orthogonal information in the data.

- NOTE: this is almost always the situation in the social sciences that variables are correlated: there is no such thing as a real set of "independent variables."

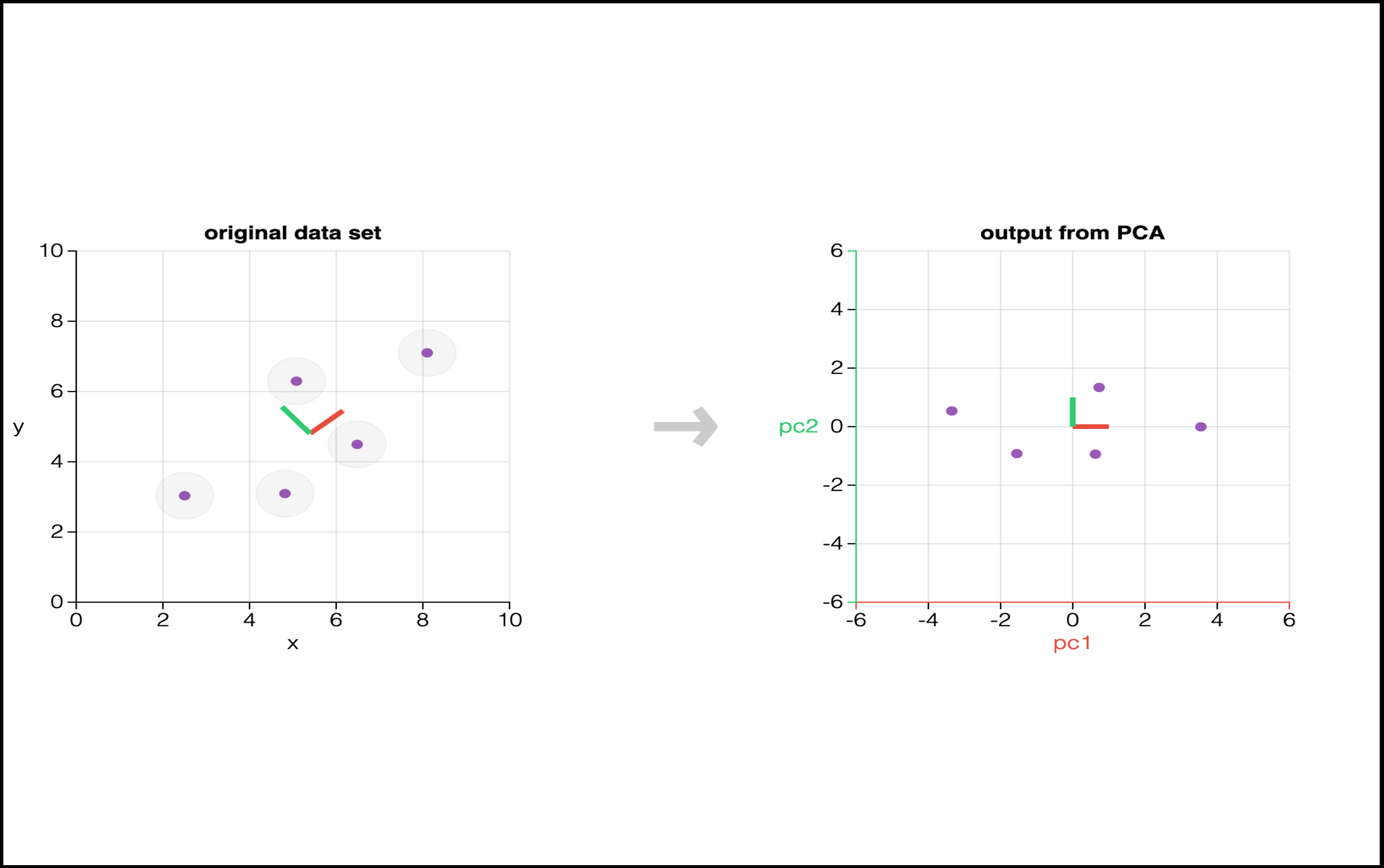
- Therefore if all p variables are uncorrelated then the axes are already orthogonal and there is no need to perform PCA (let me know if you have a real dataset like this because it would be amazing).

- Conversely, if the p variables are perfectly correlated then there exists only one dimension worth of information in the data and all but one of the axes will have data of zero-length after PCA.

Principal Components Analysis

- Consider for a moment only two variables: X_1 and X_2, which are assumed for simplicity to have mean zero each and unit variance.
- With this variance assumption the correlation reduces to covariance.
- If the correlation between these two variables is actually zero, then the equiprobability contours (concentric lines indicating equal probability of occurrence) of these two variables is circular.
- On the other hand, if there is a non-zero ρ value then the shape of the equiprobability contours will be elliptical where the cosine of the angle of intersection from the longest elliptical axis to the original x-axis (measured at the origin since zero mean is assumed for both variables) is equal to ρ.
- In the extreme case of perfect correlation between X_1 and X_2 the equiprobability contours condense to a single line.

Illustration of 2-D PCA



From `https://setosa.io/ev/principal-component-analysis/`.

Principal Components Analysis

- Begin with an $n \times p$ data matrix $\mathbf{X}$, where variables are organized in columns, and standardize.
- Define $\mathbf{R}$ as the correlation matrix corresponding to $\mathbf{X}$ along with a matrix $\mathbf{E}$ of the eigenvectors of the $\mathbf{R}$ matrix with the constraint that squared rows and columns of $\mathbf{E}$ sum to one.
- Then by standard spectral theory (Lax 1997, Chapter 6), the matrix defined by:

  $$\boldsymbol{\lambda} = \mathbf{E}'\mathbf{R}\mathbf{E}$$

  is a matrix containing the descending eigenvalues along the diagonal and zeros elsewhere.
- It is in fact the variance-covariance matrix of the rotation defined by the principal components.
- So each eigenvalue, $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2, \ldots$, is the variance of a principal component where the first principal component now accounts for the largest variance by construction.
- Furthermore, since the off-diagonal elements are all zero, then the correlation has been removed in the new coordinate system.

Principal Components Analysis

- The importance of the $\mathbf{E}$ matrix is that it provides the transformation of the data points from the original metric to the PCA metric through simple matrix multiplication: $\mathbf{Y} = \mathbf{XE}$.

- Thus $\mathbf{Y}$ are the points in the new rotated coordinate system where the variance structure is preserved: the principal component scores.

- The usefulness of this transformation is that it is one-to-one and therefore reversible, $\mathbf{X} = \mathbf{E}'\mathbf{Y}$ because of the orthogonal property of the $\mathbf{E}$ matrix.

- The $\mathbf{E}$ matrix of normalized eigenvectors is orthogonal, meaning that:

$$\mathbf{E}'\mathbf{E} = \mathbf{E}\mathbf{E}' = \mathbf{I}.$$

- Also this property also allows us to modify $\boldsymbol{\lambda} = \mathbf{E}'\mathbf{R}\mathbf{E}$ according to:

$$\boldsymbol{\lambda}\mathbf{E}' = \mathbf{E}'\mathbf{R}\mathbf{E}\mathbf{E}' = \mathbf{E}'\mathbf{R}$$

  since $\mathbf{E}\mathbf{E}' = \mathbf{I}$.

Principal Components Analysis

- We can also define a new matrix called the component loadings according to:

$$\mathbf{L} = \mathbf{E}\boldsymbol{\lambda}^{\frac{1}{2}}$$

  where the square root on $\boldsymbol{\lambda}$ is simply the square root of each diagonal element.

- The $\mathbf{L}$ matrix is theoretically important due to two related multiplicative properties:

$$\begin{aligned} \mathbf{L}\mathbf{L}' &= \mathbf{E}\boldsymbol{\lambda}^{\frac{1}{2}}(\mathbf{E}\boldsymbol{\lambda}^{\frac{1}{2}})' \\ &= \mathbf{E}\boldsymbol{\lambda}\mathbf{E}' \\ &= \mathbf{E}(\mathbf{E}'\mathbf{R}\mathbf{E})\mathbf{E}' \\ &= \mathbf{R} \end{aligned} \qquad\qquad \begin{aligned} \mathbf{L}'\mathbf{L} &= (\mathbf{E}\boldsymbol{\lambda}^{\frac{1}{2}})'\mathbf{E}\boldsymbol{\lambda}^{\frac{1}{2}} \\ &= (\boldsymbol{\lambda}^{\frac{1}{2}})'\mathbf{E}'\mathbf{E}\boldsymbol{\lambda}^{\frac{1}{2}} \\ &= \boldsymbol{\lambda} \end{aligned}$$

- So the product of component loadings is either equal to the correlation matrix ($\mathbf{R}$) or the diagonal eigenvalue matrix ($\boldsymbol{\lambda}$), depending on the order of matrix multiplication.

- This is a very, very cool property.

Principal Components Analysis, Worked Example

- Consider the following contrived dataset and its standardized column variables:

$$\tilde{\mathbf{X}} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 3 \\ 3 & 1 & 2 \\ 4 & 3 & 6 \\ 5 & 5 & 5 \\ 6 & 7 & 6 \\ 7 & 9 & 9 \\ 8 & 8 & 8 \\ 9 & 8 & 3 \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} -1.461 & -1.109 & -1.385 \\ -1.095 & -0.421 & -0.652 \\ -0.730 & -1.453 & -1.018 \\ -0.365 & -0.765 & 0.448 \\ 0.000 & -0.076 & 0.081 \\ 0.365 & 0.612 & 0.448 \\ 0.730 & 1.300 & 1.547 \\ 1.095 & 0.956 & 1.181 \\ 1.461 & 0.956 & -0.652 \end{bmatrix}$$

```
tilde.X <- matrix(c(1,2,1,2,4,3,3,1,2,4,3,6,5,5,5,6,7,6,7,9,9,8,8,8,9,8,3),
    ncol=3,byrow=TRUE)
X <- scale(tilde.X)
```

Principal Components Analysis, Worked Example

- The correlation matrix from $\mathbf{X}$ is:

$$\mathbf{R} = \begin{bmatrix} 1.000 & 0.880 & 0.619 \\ 0.880 & 1.000 & 0.716 \\ 0.619 & 0.716 & 1.000 \end{bmatrix}.$$

  The eigenvalues and eigenvectors are found by solving the characteristic equation: $|\mathbf{R} - \boldsymbol{\lambda}| = 0$.

- This produces the matrices:

$$\mathbf{E} = \begin{bmatrix} -0.585 & -0.514 & 0.628 \\ -0.607 & -0.236 & -0.759 \\ -0.538 & 0.825 & 0.174 \end{bmatrix} \boldsymbol{\lambda} = \begin{bmatrix} 2.482 & 0.00 & 0.000 \\ 0.000 & 0.41 & 0.000 \\ 0.000 & 0.00 & 0.108 \end{bmatrix} \mathbf{L} = \begin{bmatrix} -0.921 & 0.328 & 0.206 \\ -0.956 & 0.150 & -0.249 \\ -0.847 & -0.528 & 0.057 \end{bmatrix}.$$

- Calculated by:

```
R <- cor(X)
E <- eigen(R)$vectors
lambda <- eigen(R)$values * diag(3)
L <- E %*% chol(lambda)
```

Principal Components Analysis, Worked Example

- This means that the proportion of the total variance explained by each of the three principal components are $2.482/3 = 0.827, 0.41/3 = 0.137, 0.108/3 = 0.036$.

- By the same reasoning, the second principal component explains 13.7% of the total variance and the third principal component explains 3.6% of the total variance.

- The component scores are produced by pre-multiplying the original data matrix by $\mathbf{E}$, producing:

$$\mathbf{Y} = \begin{bmatrix} 2.272 & -0.130 & -0.316 \\ 1.247 & 0.124 & -0.482 \\ 1.857 & -0.122 & 0.467 \\ 0.437 & 0.737 & 0.429 \\ 0.003 & 0.085 & 0.072 \\ -0.826 & 0.038 & -0.157 \\ -2.049 & 0.595 & -0.259 \\ -1.856 & 0.186 & 0.168 \\ -1.084 & -1.513 & 0.078 \end{bmatrix}.$$

```
diag(lambda)/3
Y <- X %*% E
```

Principal Components Analysis, Worked Example

- Here the mean for each $\mathbf{Y}$ variable (the columns) remains zero and the corresponding variance is no longer unity but rather the corresponding diagonal value of $\boldsymbol{\lambda}$.

- Because of these component scores result from the simple matrix multiplication defined in $\mathbf{Y} = \mathbf{XE}$, they are in fact linear combinations of the original data with weights determined by the eigenvector matrix $\mathbf{E}$.

- For instance, the first value of $\mathbf{Y}$ is produced from:

$$\begin{aligned} Y_{11} &= \sum_{j=1}^{3} E_{.1} X_{1.} \\ &= 0.585 \times -1.461 + 0.607 \times -1.109 + 0.538 \times -1.385 \\ &= 2.272. \end{aligned}$$

```
Y11 <- sum(E[,1] %*% X[1,])
```

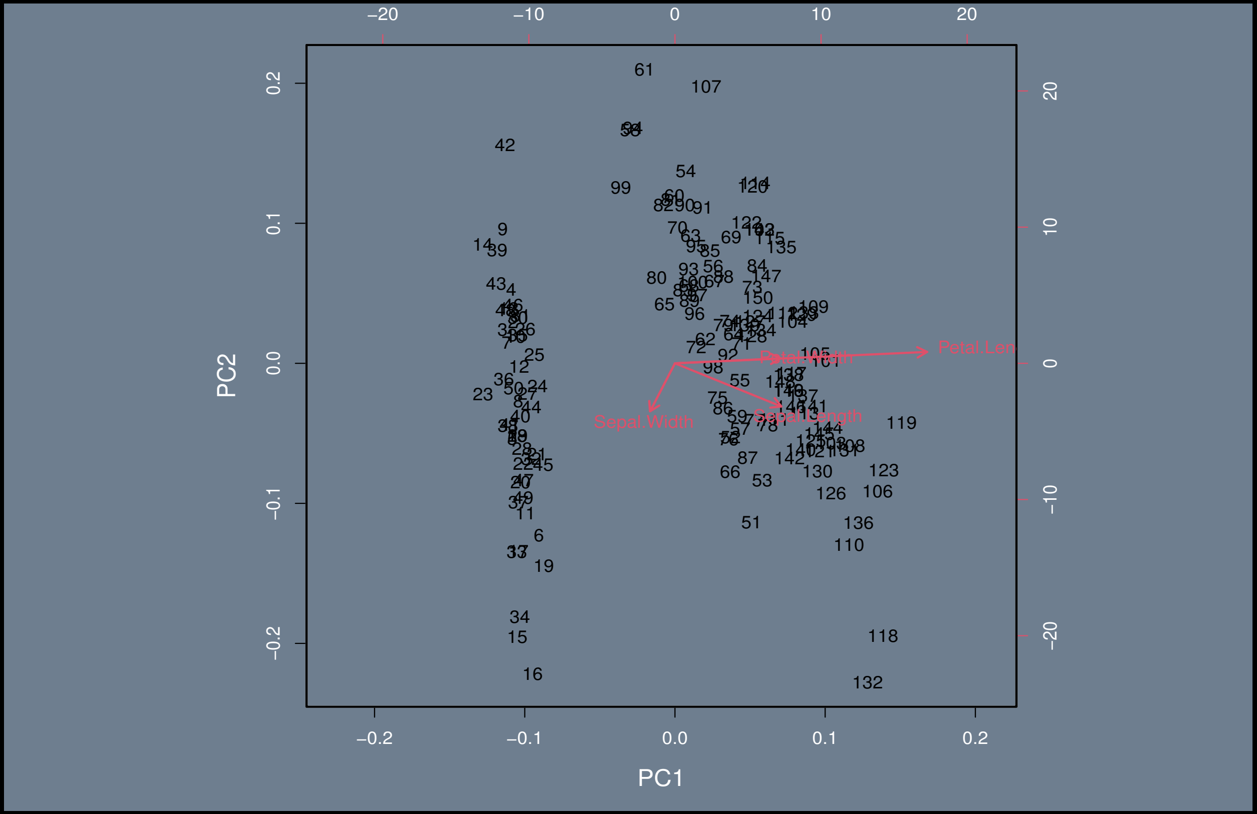
PCA In Practice

- PCA was shown with eigen-analysis to understand the process and the intution.
- In practice it is done by:

```
prcomp(iris2)
Standard deviations (1, .., p=4):
[1] 2.0562689 0.4926162 0.2796596 0.1543862

Rotation (n x k) = (4 x 4):
                     PC1         PC2         PC3        PC4
Sepal.Length  0.36138659 -0.65658877  0.58202985  0.3154872
Sepal.Width  -0.08452251 -0.73016143 -0.59791083 -0.3197231
Petal.Length  0.85667061  0.17337266 -0.07623608 -0.4798390
Petal.Width   0.35828920  0.07548102 -0.54583143  0.7536574
```

PCA In Practice

```
par(mfrow=c(1,1),mar=c(5,5,2,2),lwd=2,col.axis="white",col.lab="white",
    col.main="white", col.sub="white", col="white",bg="slategray", cex.lab=1.3)
biplot(prcomp(iris2))
```



Association Rules

- This is a data mining tool (unsupervised) that answers the question of finding whether certain "items" occur together more frequently than randomness predicts.
- The general setup is $X_1 \ldots X_k \longrightarrow Y$ having support S and confidence C.
- So when a purchase/event contains $X_1 \ldots X_k$ at least C% of the time then **Y** also occurs, and there are at least S% of these transactions out of those observed.
- So we care about frequency and statistical significance of these associations.
- For two individual items X and Y *lift* is the ratio of the probability of joint occurrence over the product of individual occurrences under the assumption of independence:

$$\text{lift} = \frac{p(X,Y)}{p(X)p(Y)}.$$

- So a value near one implies that the items are not associated, small values less than one imply that these rarely occur together, and large values imply common joint occurrence.
- There are then two elements: Antecedent (if) this is an item/group of items that are found in the itemsets, and Consequent (then): these are later associated with an Antecedent or set of Antecedents.

Association Rules

- Commonly association rules are applied to big commercial datasets: $p \approx 10^5, n \approx 10^{10}$.
- Algorithm:
  - Tabulate all combinations of items in a dataset that occur together with a minimum frequency: frequent itemsets.
  - Assert association rules that parameterize co-occurrence within the frequent itemsets.
- This is implemented with rule mining algorithms that identify a basket of items $X_1 \ldots X_k$ relative to Y.
- Most often the Xs are binary such as bought/didn't buy, and the dataset is checkout summaries so that x_{ij} is person i's realized (observed) purchase status of item j.

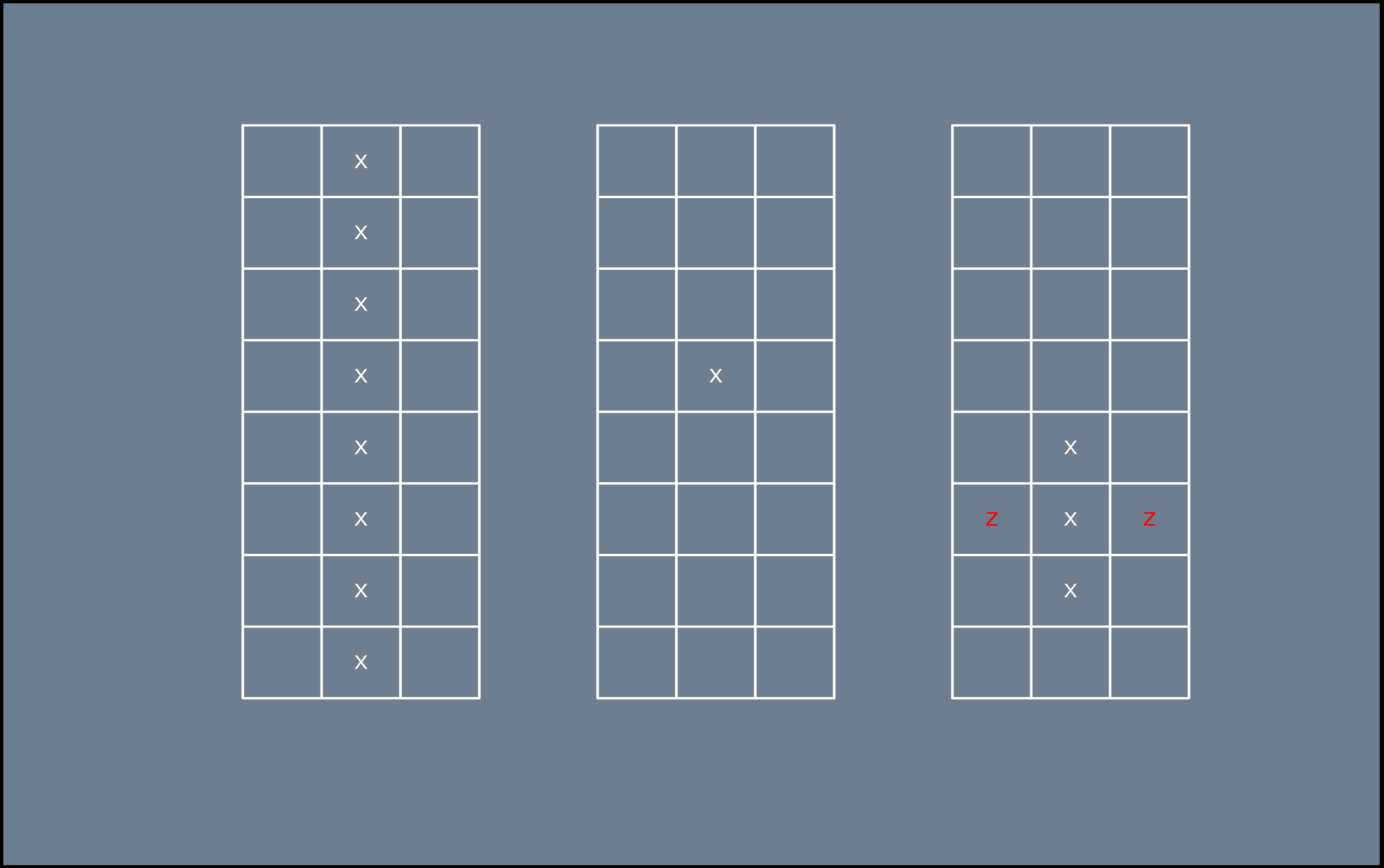
Association Rules

- Often the events do not happen often enough for a single unit of study so *regions* of the event space are considered: groups of events, events over time, groups of people, etc.
- Define S_j as the support of events for the jth variable and $s_j \subseteq S_j$ as a subset.
- We want to find a subset of variable values such that the intersection has relatively large probability:

$$\text{conjunctive rule} = p\left[\bigcap_{j=1}^{p}(X_j \in s_j)\right].$$

- Only 2 types of subsets are considered due to data size frequently encountered:
  - s_j is a single value of X_j denoted $s_j = \eta_{j0}$
  - all values that X_j can assume: $s_j = S_j$.

Illustration of Association Rules



Association Example: Market Basket Analysis

- Consider a dataset of 9,409 questionnaires filled out by shoppers in the SF Bay Area where the demographic questions only are studied:

Feature Number	Feature	Categories	Type
1	Sex	2	Categorical
2	Marital Status	5	Categorical
3	Age	7	Ordinal
4	Education	6	Ordinal
5	Occupation	9	Categorical
6	Income	9	Ordinal
7	Years in Bay Area	5	Ordinal
8	Dual Income	3	Categorical
9	Number in Household	9	Ordinal
10	Number of Children	9	Ordinal
11	Householder Status	3	Categorical
12	Type of Home	5	Categorical
13	Ethnic Classification	8	Categorical
14	Language in Home	3	Categorical

Association Example: Market Basket Analysis

- There are missing data, which was case-wise deleted by Hastie, Tibshirani, and Friedman.
- They used software called `Apriori` by Christian Borgelt.
- Ordinal features were cut at their median to produce dichotomous features (dummy variables).
- Categorical variables given a treatment contrast.
- The final dataset after pre-processing was 6,875 $\times$ 50.
- `Apriori` found 6,288 association rules that had 5 or less predictors and support of at least 10%.

Association Example: Market Basket Analysis

- Association Rule # 1: Support 25%, Confidence 97%, Lift 1.03

Antecedents	Consequent
Number in Household = 1 Number of Children = 0	Language in Home = English

- Association Rule # 2: Support 13.4%, Confidence 80.8%, Lift 2.13

Antecedents	Consequent
Language in Home = English Householder Status = own Occupation = Professional/Managerial	Income $\geq$ \$40,000

- Association Rule # 3: Support 26.5%, Confidence 82.8%, Lift 2.15

Antecedents	Consequent
Language in Home = English Income $<$ \$40,000 Marital Status = Not Married Number of Children = 0	Education <u>not</u> college graduate or graduate study

K-Nearest Neighbors

- This is a classifier in the family called *memory-based models*.
- It is very simple and very fast spatial approach, even with very large or very fast data.
- For the basic method there are only two Decisions: neighborhood size k, and the distance metric (Euclidean, Manhattan, etc.).
- In picking a specific k, users usually do some trial and error since too small means high variance and too large may miss important local features.
- Basic algorithm:
  - Identify a multidimensional query point x_0.
  - Find the k points nearest to this points: x_r for $r = 1, \ldots, k$, usually using Euclidean distance in feature space: $d(r) = \|x_{(r)} - x_0\|$. ($\|\boldsymbol{v}\| = (v_1^2 + v_2^2 + \cdots + v_n^2)^{\frac{1}{2}} = (\boldsymbol{v}' \cdot \boldsymbol{v})^{\frac{1}{2}}$).
  - For the feature of interest assign an attribute to x_o based on some "voting" criteria amongst the k, usually majority rule (ties settled at random).
  - Repeat as new starting points are identified.

K-Nearest Neighbors

- Typically the variables (dimensions) are standardized to mean zero and standard deviation one such that the measurement of variables does not lead to domination/sublimation.
- This is a old and popular tool that is well-suited to big data problems, including EKG patterns, handwriting analysis, image analysis, satellite data, internet traffic, political ideology, and more.
- It performs well even when the decision barrier is very irregular.
- There are some challenges when the features are both continuous and categorical since more decisions need to be made.
- Other challenges include sparsity and very high dimensions.
- There are many, many extensions/enhancements like weighting.

K-Nearest Neighbors, Example

```
library(neighbr)
data(iris); head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species ID
1          5.1         3.5          1.4         0.2  setosa  1
2          4.9         3.0          1.4         0.2  setosa  2
3          4.7         3.2          1.3         0.2  setosa  3
4          4.6         3.1          1.5         0.2  setosa  4
5          5.0         3.6          1.4         0.2  setosa  5
6          5.4         3.9          1.7         0.4  setosa  6

iris$ID <- c(1:150) # APPEND AN ID NUMBER COLUMN

train_set <- iris[1:145,] # USE FIRST 145 CASES AS TRAINING WITH ALL FEATURES

test_set <- iris[146:150,-c(4,5,6)] # REMOVE PREDICTED VARIABLES FROM TEST DATA
```

K-Nearest Neighbors, Example

```
fit <- knn(train_set=train_set,test_set=test_set, k=9, categorical_target="Species",
    continuous_target= "Petal.Width", comparison_measure="euclidean",
    return_ranked_neighbors=9, id="ID")
fit$test_set_scores
   categorical_target continuous_target neighbor1 neighbor2 neighbor3 neighbor4
146          virginica          2.022222        78       142       140       111
147          virginica          1.577778        73       124       134        84
148          virginica          1.933333       111       116        78       117
149          virginica          2.133333       137       116       138       111
150          virginica          1.855556       115       128        84       139
    neighbor5 neighbor6 neighbor7 neighbor8 neighbor9
146       113       117       116        53       141
147       127       112       120        74        64
148       134       112       138       113       142
149       117       104       125       145       101
150       102       143        71       122       134
```

Support Vector Machines

- This is a very useful and very popular supervised classifier on interval measured data.
- Consider multidimensional, interval-measured data where we want to separate (classify) points by specifying lines in each dimension which collectively determine a hyperplane.
- The idea is to get the best separation possible and to not have the line too close to data points to increase generalizability with future data.
- This is done by finding the hyperplane maximizes the *margin* of the training data.
- Unlike many other classifiers SVM does not provide probabilities.

SVM Illustration

Support Vector Machines

- Training sample Points are called *support vectors* and those closest to the margin are the most influential.
- For non-linear structures in the features kernels (e.g. radial basis function) provide an efficient tool for separation (not discussed here).
- The training data consists of n pairs: $\{(x_1, y_1) \ldots (x_n, y_n)\}$, where $x_i \in \Re^p$ and $y_i \in \{-1, 1\}$.
- Define a hyperplane by:
$$x : f(x) = \beta_0 + x\beta = 0$$
 where β is of length 1.
- If the points are separable then $y_i = f(x_i) > 0 \; \forall i$.
- This means that we can find the hyperplane that provides the maximum margin between the $y = -1$ group and the $y = 1$ group:
$$\max_{\beta_0, \beta} = M \quad \text{subject to} \quad y_i = (f(x_i) - \beta_0 + x_i\beta) \geq M, \quad \forall i.$$
- So the band is M away from the hyperplane in both directions.

Support Vector Machines

- If there is overlap in the feature space that cause some points to be on the wrong side of any specified hyperplane then specify slack variables, $\xi_i \geq 0,\ \sum \xi_i \leq C$, for some specified constant C, and we now specify:

$$\max_{\beta_0,\beta} = M(1 - \mathbf{x}_i) \quad \text{subject to} \quad y_i = (f(x_i) - \beta_0 + x_i\beta) \geq M, \quad \forall i.$$

- Since miscalculations occur when $\mathbf{x}_i > 0$ then C bounds the total number of training misclassifications.

- With overlap we drop the norm constraint on β and now define $M = 1/||\beta||$.

- This leads to a modified procedure:

$$\min ||\beta|| \quad \text{subject to} \begin{cases} y_i = (f(x_i) - \beta_0 + x_i\beta) \geq 1 = \xi_i, \quad \forall i \\ \xi_i \geq \sum \mathbf{x}_i \leq C \end{cases}$$

SVM Example in R

```
set.seed(999)
x <- matrix(rnorm(60), 30, 2)
y <- rep(c(-1, 1), c(15,15))
x[y==1,] = x[y==1,] + 1
par(mfrow=c(1,1),mar=c(5,5,2,2),lwd=2,col.axis="white",col.lab="white",
    col.main="white", col.sub="white", col="white",bg="slategray", cex.lab=1.3)
plot(x, col = y+51, pch = "+", cex=2)

library(e1071)
svm.example <- data.frame(x, y = factor(y))
svm.out <- svm(y ~ ., data = svm.example, kernel = "linear", cost = 5, scale = FALSE)
```

SVM Example in R

```
summary(svm.out)

Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  5
Number of Support Vectors:  9
 ( 4 5 )
Number of Classes:  2
Levels:
 -1 1

plot(svm.out,svm.example)
```

SVM Default Graph

SVM Example in R

```
Beta.times.Label <- svm.out$coefs
cbind(svm.out$SV,Beta.times.Label)  # THE 9 SUPPORT VECTORS
           X1        X2
1  -0.2817402 2.3826642  5.000000
11  2.3254637 0.5143633  5.000000
12  1.1339774 1.0084981  5.000000
15  1.9576504 1.3006654  5.000000
17  3.0683351 0.9491223 -1.601791
21  0.7714367 1.8833359 -5.000000
25  0.8747315 1.9331901 -3.398209
28  1.1151596 0.7207841 -5.000000
30  1.8733210 1.4344990 -5.000000
```

SVM Example in R

```
svm.pred <- fitted(svm.out) # TEST WITH TRAINING DATA
rbind(svm.pred[1:15],svm.pred[16:30])
     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
[1,] 2 1 1 1 1 1 1 1 1  1  1  1  1  1  2
[2,] 2 2 2 2 2 2 2 2 2  2  2  2  1  2  2
Levels: -1 1

table(svm.pred,y) # ACCURACY SUMMARY
        y
svm.pred -1  1
      -1 13  1
      1   2 14
```

Bootstrapping for Standard Errors

- **Big Idea**: sometimes it is difficult to get the sampling properties of an estimator, even a commonly used one.
- Some statistics have known variance properties for finite samples and some do not. Does this mean we should only use the former unless we have population data?
- Definitive citations: Efron (1979), Efron and Tibshirani (1993).
- **Case Study:** suppose we have a dataset on leukemia, ignoring a whole host of things and condensing our analysis down to two variables: *CD4 Count/10* a dichotomous outcome indicating that there was a relapse from a remission stage:

Relapse	94	197	16	38	99	141	23		
No Relapse	52	104	146	10	50	31	40	27	46

- Note that these data are *imbalanced*.

Bootstrapping for Standard Errors

- The question is whether there is a difference by CD4 count, and the natural choice of test is the difference of means: $\bar{x}_{\text{relapse}} = 86.86$, $\bar{x}_{\text{no relapse}} = 56.22$.

- This is easy since we know that:

$$SE(\bar{x}_{\text{relapse}}) = \sqrt{(s^2_{\text{relapse}}/n_{\text{relapse}})} = 25.24,$$
$$SE(\bar{x}_{\text{no relapse}}) = \sqrt{(s^2_{\text{no relapse}}/n_{\text{no relapse}})} = 14.14$$

- But we also know that the mean is not very resistant to outliers and it could be that a notable case, and one could be driving the subsequent findings.

- So what about using the median instead of the mean? This is obvious choice in one sense, but it leaves us with no closed form solution for the standard error.

Bootstrapping for Standard Errors

- So consider the following algorithm, for some statistic of interest, θ:

1. Draw B "bootstrap" samples of size n, independently, **with replacement** from the sample $\mathbf{x}$:
$$\mathbf{x}^{*1}, \mathbf{x}^{*2}, \ldots, \mathbf{x}^{*B}$$
(note the notation to differentiate the bootstrap sample from the original sample).
2. Calculate the sample statistic of interest, θ^{*b} for each bootstrap sample, and the mean of these statistics:
$$\bar{\theta}^{*} = \frac{1}{B}\sum_{b=1}^{B}\theta^{*b}$$
3. Estimate the bootstrap standard error of the statistic by:
$$\text{Var}(\theta) = \frac{1}{B-1}\sum_{b=1}^{B}\left(\theta^{*b} - \bar{\theta}^{*}\right)^2$$
where obviously $SE(\theta) = \sqrt{\text{Var}(\theta)}$.

- We call the limit of this standard error as B goes to infinity is called the *ideal* bootstrap estimate, and this procedure is called the *nonparametric* bootstrap estimate.

Bootstrapping for Standard Errors

```
relapse <- c(94,197,16,38,99,141,23)
no.relapse <- c(52,104,146,10,50,31,40,27,46)
B <- 1000
no.relapse.mat <- relapse.mat <- NULL
for (i in 1:B)  {
    relapse.mat    <- rbind(relapse.mat,   sample(relapse,length(relapse),
                            replace=TRUE))
    no.relapse.mat <- rbind(no.relapse.mat,sample(no.relapse,length(no.relapse),
                            replace=TRUE))
}

relapse.mean <- mean(apply(relapse.mat,1,mean))
relapse.se <- sqrt(var(apply(relapse.mat,1,mean)))
no.relapse.mean <- mean(apply(no.relapse.mat,1,mean))
no.relapse.se <- sqrt(var(apply(no.relapse.mat,1,mean)))
relapse.median <- mean(apply(relapse.mat,1,median))
relapse.median.se <- sqrt(var(apply(relapse.mat,1,median)))
no.relapse.median <- mean(apply(no.relapse.mat,1,median))
no.relapse.median.se <- sqrt(var(apply(no.relapse.mat,1,median)))
```

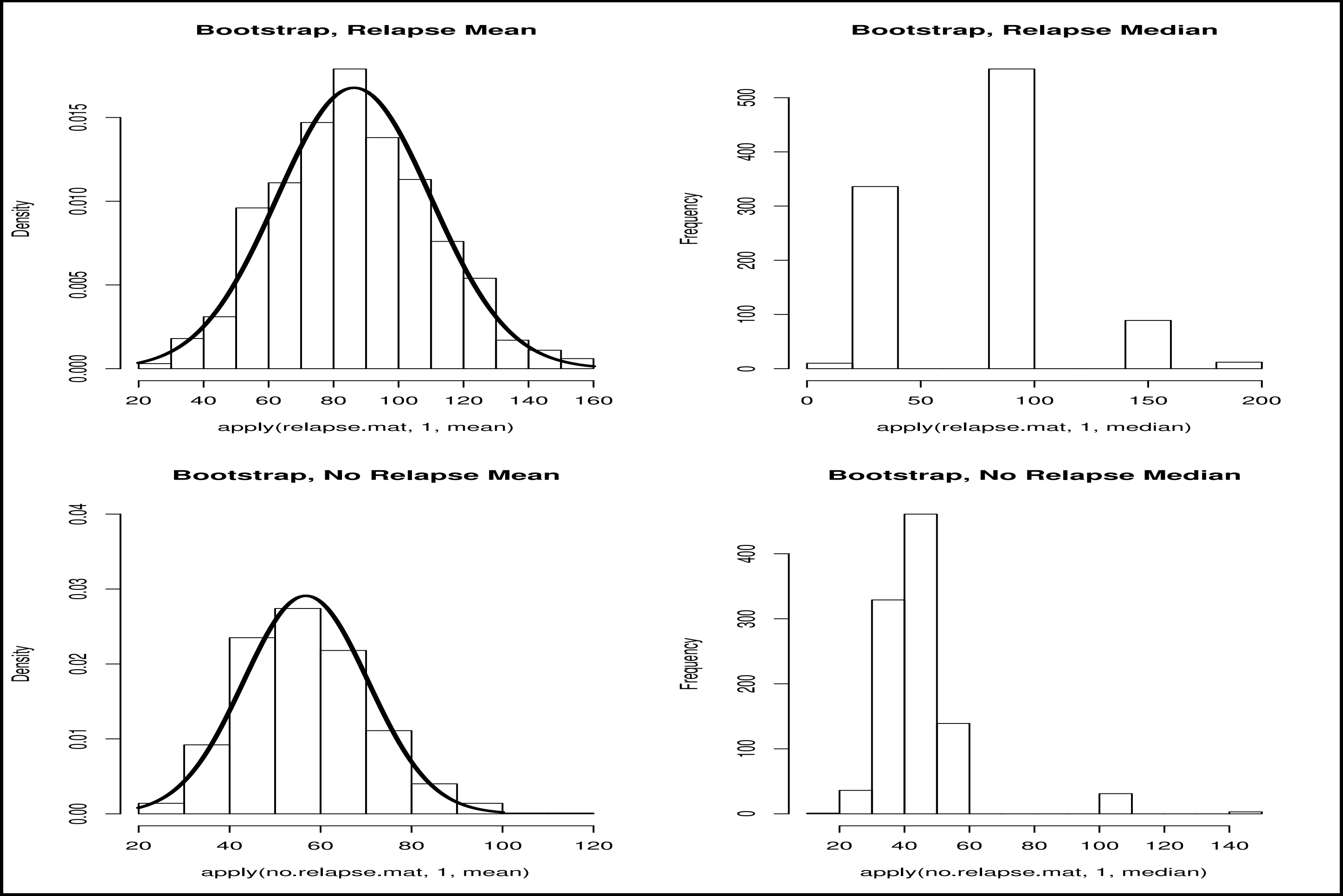
Bootstrapping for Standard Errors

```
final.relapse.mat <- rbind( c(relapse.mean, relapse.se,
                              no.relapse.mean, no.relapse.se),
    c(relapse.median, relapse.median.se, no.relapse.median, no.relapse.median.se) )
dimnames(final.relapse.mat) <-
    list( c("Mean","Median"), c("Relapse Est","Relapse SE",
                                "No Relapse Est","No Relapse SE") )

final.relapse.mat
    Relapse Est  Relapse SE  No Relapse Est  No Relapse SE
Mean   88.19143    21.57597        54.86444       12.45360
Median 84.37000    36.40226        44.07000       12.65810
```

Bootstrapping for Standard Errors

```
par(mfrow=c(2,2),bg="white")
hist(apply(relapse.mat,1,mean), freq=FALSE, main="Bootstrap, Relapse Mean")
ruler <- seq(20,160,length=100)
lines(ruler,dnorm(ruler,relapse.mean,relapse.se),lwd=3)
hist(apply(relapse.mat,1,median), main="Bootstrap, Relapse Median")
hist(apply(no.relapse.mat,1,mean), ylim=c(0,0.04),freq=FALSE,
     main="Bootstrap, No Relapse Mean")
ruler <- seq(20,100,length=100)
lines(ruler,dnorm(ruler,no.relapse.mean,no.relapse.se),lwd=3)
hist(apply(no.relapse.mat,1,median), main="Bootstrap, No Relapse Median")
```

Bootstrapping for Standard Errors



Random Forests

- Suppose we have a dataset with N cases and M features and we want to classify.
- General Algorithm:
  - Draw B bootstrap samples of size n, independently, **with replacement** from the data. The size of n depends on the context of the problem.
  - For each of these B train a decision tree sampling $m \ll M$ features uniformly from the full M and the collection of decision trees picks the best features given the features that they individually have.
  - New data is tested with all of the decision trees and the final result is an aggregation such as majority vote.
- RFs can handle large problems very easily.
- Notice that this process can be parallelized for computational efficiency.
- RFs also provide a proximity matrix showing similarity between all of the points by counting the proportion of times two selected data points are classified together at the bottom of the trees.

Bagging: Bootstrap Aggregation

- This is more general *ensemble* method than random forests since any ML process can be used.
- Draw B bootstrap samples of size n, independently, **with replacement** from the data. The size of n depends on the context of the problem and the size of N.
- Train a model on each of these B datasets.
- Obtain a out-of-sample test dataset in the typical fashion and use each of the B models to predict.
- The procedure can also easily be parallelized.

Boosting

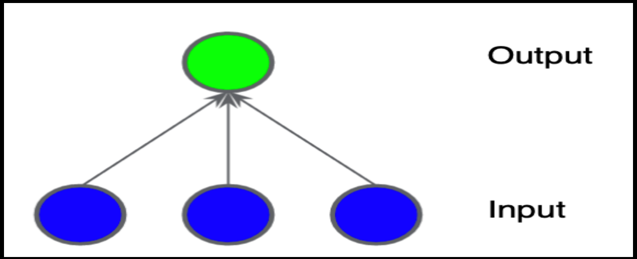
- This is another very general procedure whereby one iteratively trains classifiers using the data cases where the previous model produced misclassifications.
- Therefore each iteration gets a smaller dataset than all of the previous models.
- Algorithm:
  - Give all the data points equal weights.
  - For the jth model in the series:
    - train a classifier using the current weights
    - predict from the training data
    - determine the error from this prediction
    - calculate new weights based on the errors in the jth classification
  - Repeat.
  - At the end of the run a weighted average of the predictions from all of the models where the weight is proportional to each of the model's prediction accuracy.

Introduction to Neural Networks

- NN are a "black box" machine learning to explain some set of outcomes given a set of inputs.
- Also a classification problem.
- This means that the coefficients and values on the internal layers are not interpretable, and the only result with any value is the final result.
- The network analysis is neurological/cerebrial wherein learning is done through a serial training processes, which can be quite complex computationally.
- Unlike more conventional social science regression-style inference the (almost) only concern is explaining the outcome regardless of complexity or generalizability.

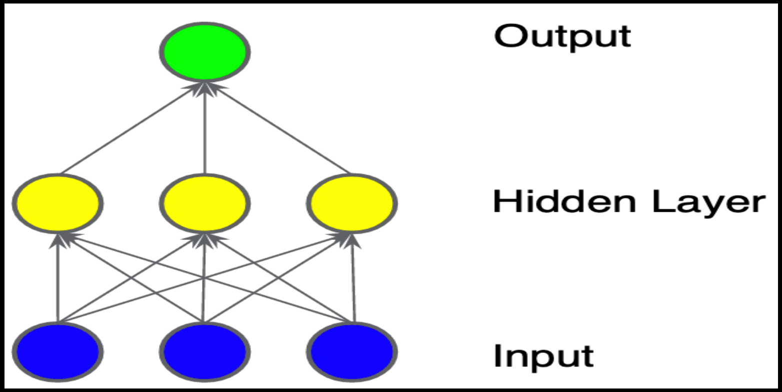
Introduction to Neural Networks

- The starting point is a representation of a simple *linear* model as a graph.
- The blue circles are input features.
- The green circle is the *weighted* sum of the inputs, which is the output (classification) of interest).
- This looks like a causal diagram, but it is not meant to imply causality.



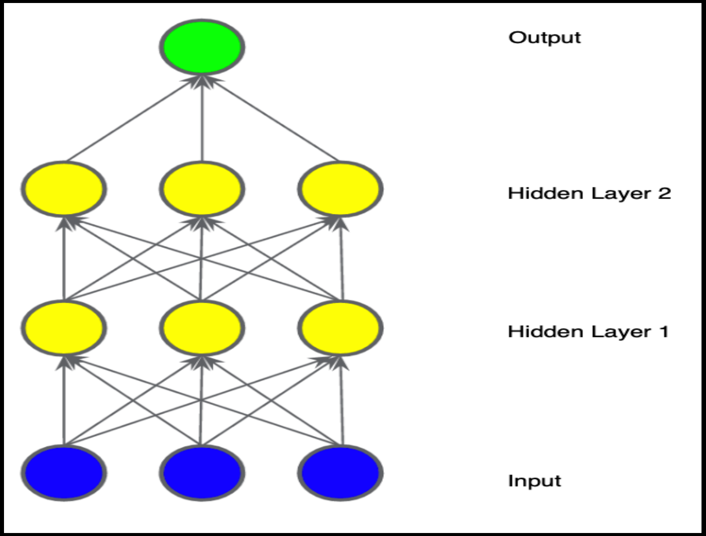
Introduction to Neural Networks

- Now consider a single *hidden layer* of intermediate values from an intermediate weighting process that is conditional on each input features as a weighted sum of these.
- This is the yellow circles in the figure.
- The weights could be (and many are in complex models) equal to zero in the first level relationships.
- Now the output is a direct weighted linear sum of the hidden layer nodes.



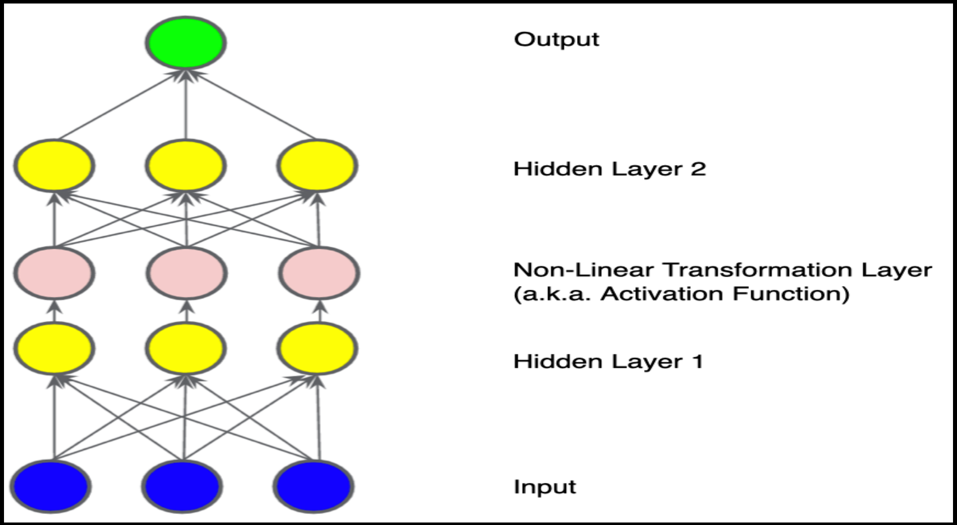
Introduction to Neural Networks

- A single hidden layer is unusually simple for real problems, so we can have multiple iterations, each conditional on the previous hidden layer with weighting.
- This is still a linear construction involving matrix algebra.



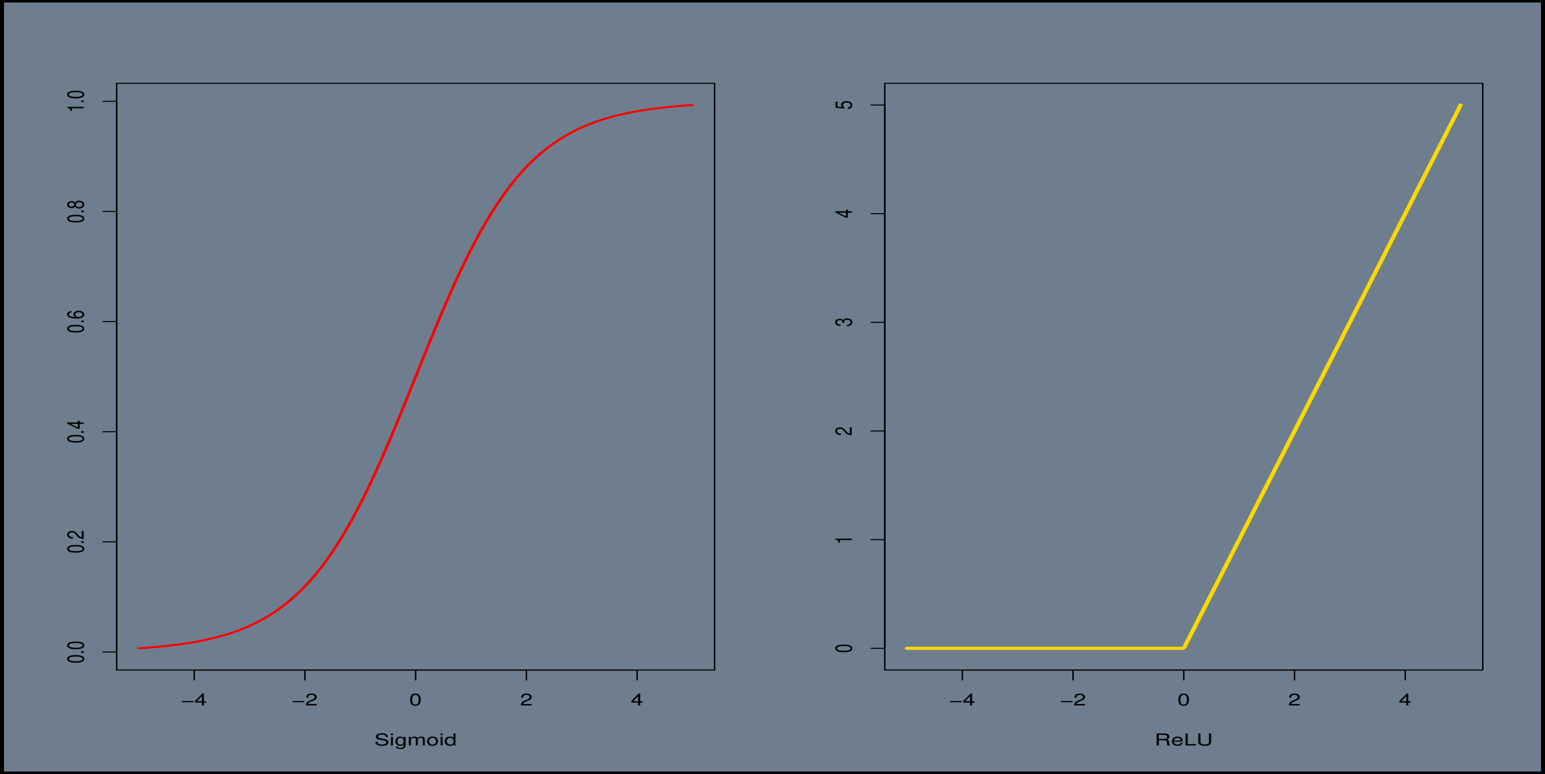
Introduction to Neural Networks

- Of course we want to represent/solve nonlinear relationships and this is done by piping the linear weighted relationships through a nonlinear function called an *activation function*.
- In the figure there is a single layer of activation functions between two hidden layers.
- In practice we can model very complicated relationships between the inputs and the final output by stacking many linear and nonlinear layers inbetween, creating higher level functionality.



Activation Functions

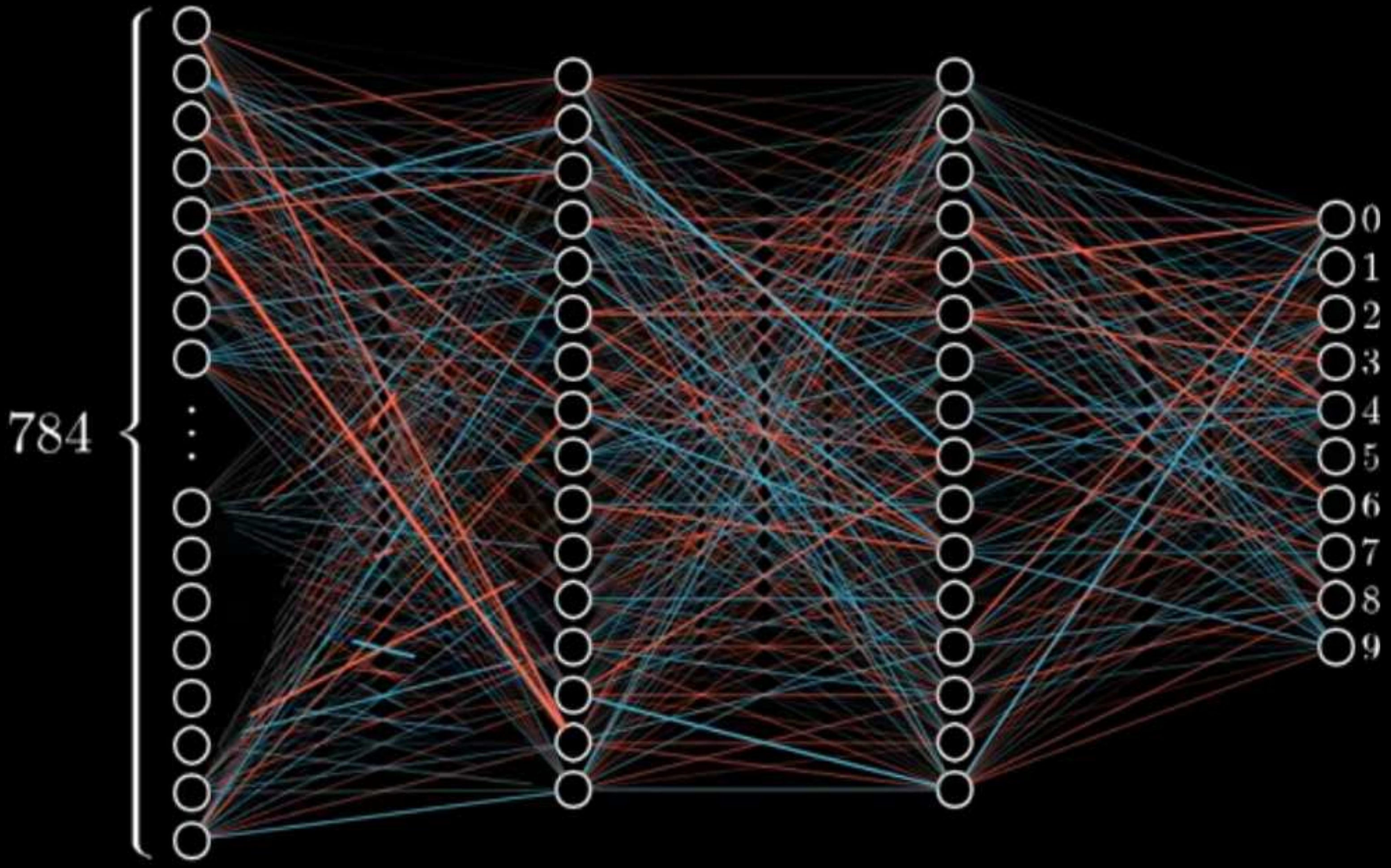
- There are two very common forms which are really like CDFs.
- *Sigmoid* (logit), $F(x) = [1 + \exp(-x)]^{-1}$.
- *Rectified Linear Unit* (ReLU), $F(x) = \max(0, x)$.
- As with anything in this realm there are many alternatives in use.



Neural Networks Development

- Phases: training, testing, validating.
- The training data must have outcome labels: $\mathbf{y}$.
- The testing data hides the label and is used for prediction and comparison to training data labels: $\mathbf{y} - \hat{\mathbf{y}}$.
- There are many forms of assessment for neural networks but they basically compare known outcomes to predicted outcomes.
- Basic neural networks are no longer in practical use but are always the best starting point for explaining more complicated learning procedures.

Deep Learning Illustration



From Chollet, Francois. Deep learning with Python. Simon and Schuster, 2021.

Neural Networks and Deep Learning

- For a long time researchers struggled to find a way to train neural nets, without success. Then in 1986, David Rumelhart, Geoffrey Hinton and Ronald Williams published the paper: "Learning Internal Representations by Error Propagation" introducing the backpropagation training algorithm, which is still the standard today.
- Their idea is based on Gradient Descent using an efficient technique for computing the gradients automatically.
- In two passes through the network (one forward, one backward), the backpropagation algorithm computes the gradient of the network's error with regards to every single model parameter.
- It finds out how each connection weight and each bias term should be tweaked in order to minimize the error.
- With these gradients, it performs a regular Gradient Descent step, and the whole process is repeated until the network converges to the solution.

Deep Learning Algorithmic Forward Pass Details

- Process mini-batch one at a time going through the full training set multiples where each pass is called an epoch.
- These mini-batches are then passed to the input layer of the network, which then sends it to the first hidden layer.
- The initial weights of the hidden layers are randomly assigned.
- Then compute the output of all the neurons in this layer for every mini-batch instance.
- This result is then passed to the first hidden layer.
- Then repeat this process at this next layer, and so on until the last layer is reached: the output layer.
- All of these intermediate results are recorded for the backward pass.
- This completes the forward pass.
- Now calculate the network error: $\mathbf{y} - \hat{\mathbf{y}}$.

The Chain Rule

- This provides a means of differentiating nested functions of the form $f \circ g = f(g(x))$.
- The case of $g(x)^{-1} = (4x^3 - 2x)^{-1}$ fits this categorization because the inner function is $g(x) = 4x^3 - 2x$ and the outer function is $f(u) = u^{-1}$.
- Typically u is used as a placeholder here to make the point that there is a distinct subfunction.
- To correctly differentiate such a nested function, we have to account for the actual *order* of the nesting relationship, done by:
$$\frac{d}{dx} f(g(x)) = f'(g(x))g'(x),$$
provided that $f(x)$ and $g(x)$ are both differentiable functions.
- We can also express this in the other standard notation: if $y = f(u)$ and $u = g(x)$ are both differentiable functions, then
$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx},$$
which may better show the point of the operation.
- If we think about this in purely fractional terms, it is clear that du cancels out of the right-hand side, making the equality obvious.

The Chain Rule

- Let us use this new tool to calculate the function $g(x)^{-1}$ from above ($g(x) = 4x^3 - 2x$):

$$\begin{aligned}
\frac{d}{dx} g(x)^{-1} &= (-1)(4x^3 - 2x)^{-2} \times \frac{d}{dx}(4x^3 - 2x) \\
&= \frac{-(12x^2 - 2)}{(4x^3 - 2x)^2} \\
&= \frac{-6x^2 + 1}{8x^6 - 8x^4 + 2x^2}.
\end{aligned}$$

Deep Learning Algorithmic Backward Pass Details

- Starting with $\mathbf{y} - \hat{\mathbf{y}}$ for the smaller number of nodes in the output layer measure these error contributions backward for each connection in the layer below using the chain running backward until the input layer is reached.

- The reverse pass efficiently measures the error gradients across every connection weight in the layer below by propogating the error gradient backward through the networ..

- Thus the last step performs a Gradient Descent step to adjust all of the connection weights in the network using all of the individual error gradients just calculated.

- Now the Deep Learning network is fully trained and can be applied to different test data.

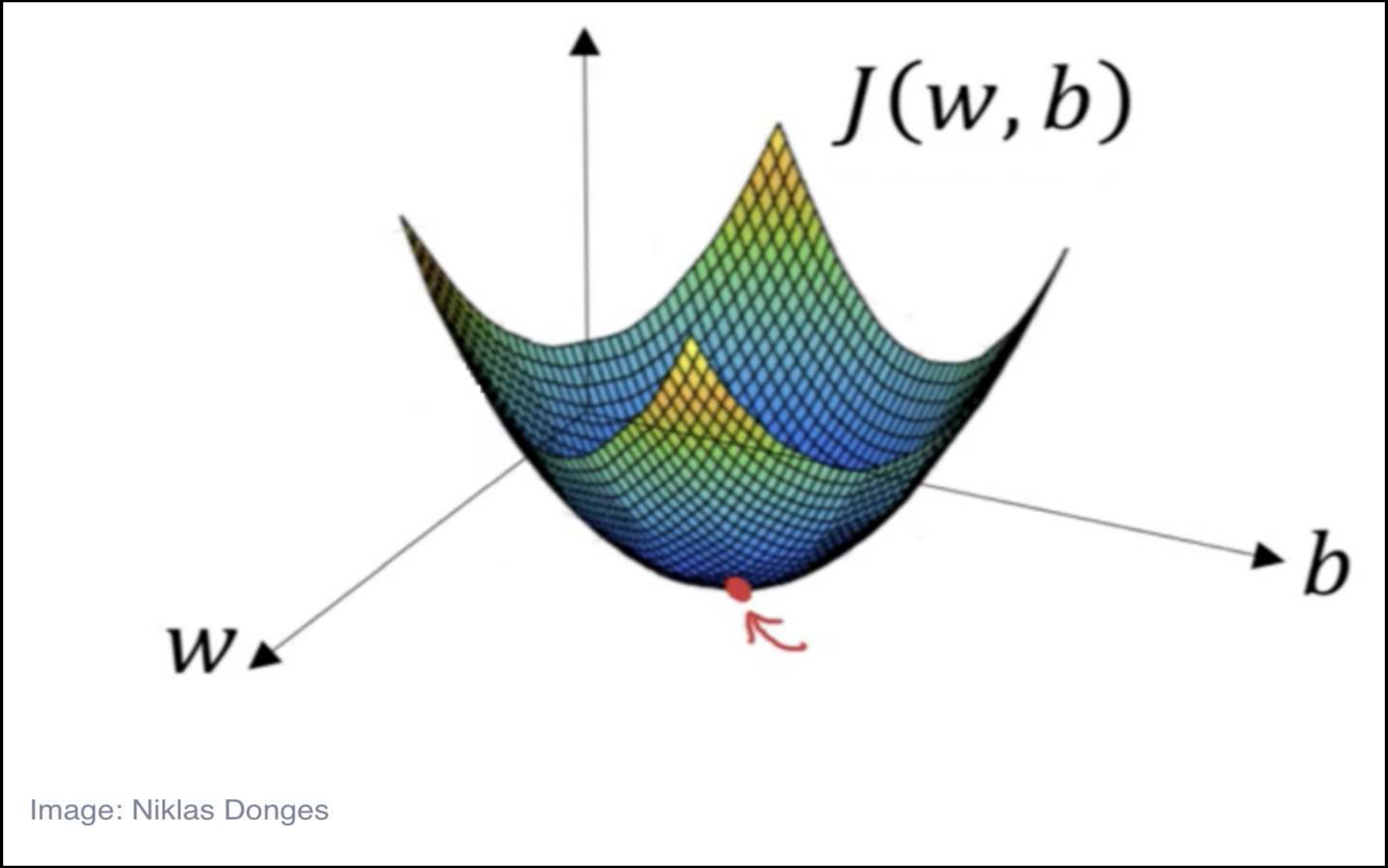
Gradient Descent for Machine Learning

- A gradient is a derivative that measures how much an (output) function changes for small changes in an input.
- In the deep learning sense it measure the change in all the weights for a small change in an error.
- Since it is a derivative, higher values mean that the slope is steeper and lower values mean that the slope is flatter.
- Higher slopes mean that the model can learn faster and zero slopes mean that no learning takes place.
- We want to get to the minimum (of errors) in a multidimensional sense, which is to minimize the function $J(w, b)$, where w is a set of weights and b is a location, then a Gradient Step looks like:

  $$b = a - \gamma \nabla f(a),$$

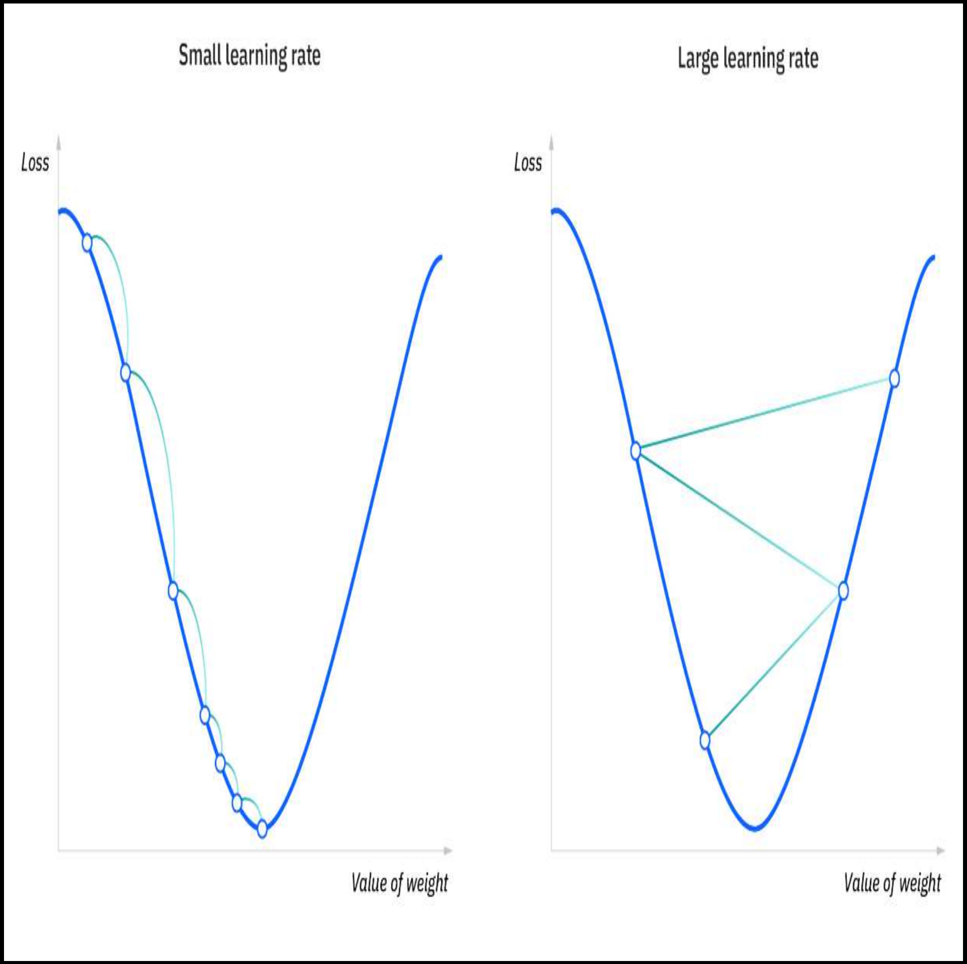
  where b is the next location, a is the current location, γ is the learning rate or step size, and $\nabla f(a)$ is the direction of the step.

Gradient Descent Illustration



Learning Rate

- This is the size of the steps that are taken to reach the function minimum.
- It is usually a small value that is evaluated and updated based on the behavior of the cost function.
- A large value results in larger steps but risk overshooting the minimum.
- A small value is more precise, but lowers the efficiency since it takes more cycles to reach the minimum.



Source: IBM.

Final Thoughts on Machine Learning

- There are usually a lot of decisions to be made that are contextual to the data and the problem:
  - size of the training set versus the test set within the same data set or outside of it.
  - tuning parameters
  - validation measures such as categorical prediction comparisons, distance measures, and cross-validation.
- Leakage: the modeling process does not have access to the training anymore at testing time.
- This an incredibly fast moving area of study with contributions from computer scientists, statisticians, mathematicians, and others.