## Introduction to machine learning

* Types of machine learning
  + **Unsupervised learning**
    - Finding structure in unlabeled data
  + **Supervised learning**
    - Making predictions based on labeled data
    - Predictions like regression or classification
  + **Reinforcement learning**
    - Making decisions based on past experience

## Clustering

### K-means clustering

* **K-means clustering** algorithm
  + Breaks observations into k pre-defined number of clusters
    - Step 1: select k (the number of clusters)
    - Step 2: select k = 3 distant data point at random
    - Step 3: measure distance between the 1st point and the k = 3 initial clusters
    - Step 4: assign the 1st point to the nearest cluster
    - Step 5: update cluster centers
      * Calculate the mean value for the blue cluster including the new point
    - Step 6: assign next point to closest cluster
      * Use updated cluster centers for distance calculation
    - Step 7: update cluster centers and move to next point
    - Step 8: repeat for each point
    - Step 9: assess the quality of the clustering by adding up the variation within each cluster
      * K-means keeps track of these clusters and their **total variance** and then does the whole thing over again with different starting points
    - Step 10: repeat with different starting points
  + After several iterations k-means clustering knows it has the best clustering so-far based on the smallest total variation with clusters
    - However, it does not knot if it has found the best overall. So it will perform several more iterations with different starting points
* More dimensions:
  + Pick random points and use the **Euclidean distance**
    - 2D: d = sqrt(x^2 + y^2)
    - 3D: d = sqrt(x^2 + y^2 + z^2)
  + We can just calculate the Euclidean distance along any number of dimensions and perform our k-means clustering in the same way
* K-means in R
  + Sample code:
    - # k-means algorithm with 3 centers, run 20 times
    - kmeans(x, centers = 3, nstart = 20)
  + input x is a numeric matrix, or data.frame, with one observation per row, one feature per column
  + k-means has a random component
  + run algorithm multiple times to improve odds of the best model
* Model selection
  + Recall k-means has a random component
  + Best outcome is based on total within cluster sum of squares:
    - For each cluster
      * For each observation in the cluster, determine squared distance from observation to cluster center
    - Sum all of them together
  + Running algorithm multiple times (i.e. setting nstart) helps find the global minimum total within cluster sum of squares
    - Increasing the default value of nstart is often sensible
* Determining number of clusters
  + Trial and error is not the beast approach
  + Systematically try a range of different k values and plot a **“scree plot”**
    - The elbow point is where K is the best

### Hierarchical clustering

* Hierarchical clustering
  + Number of clusters is not known ahead of time
  + Two kinds of hierarchical clustering
    - Bottom-up
    - Top-down
  + Hierarchical clustering in R
    - First we need to calculate Euclidean distance between observations
      * dist\_matrix <- dist(x)
      * our input is a distance matrix from the dist() function
      * nots: symmetrical pairwise distance matrix
    - the **hclust()** function returns a hierarchical clustering model
      * hc <- hclust(d = dist\_matrix)
    - plot the results as a **dendrogram**
      * plot(hc)
* **dendrogram**
  + three shaped structure used to interpret hierarchical clustering models
  + dendrogram plotting in R
    - draws a dendrogram
      * plot(hc)
    - add line
      * abline(h = 6, col = “red”)
    - cut by height h
      * cutree(hc, h = 6)
        + [1] 1, 1, 1, 2, 2
    - Cut into k groups
      * cutree(hc, k = 2)
        + [1] 1, 1, 1, 2, 2
* Linking clusters in hierarchical clustering
  + How is distance between clusters determined?
  + There are four main linkage methods to determine which cluster should be linked:
    - **Complete**: pairwise similarity between all observations in cluster 1 and cluster 2, and uses largest of similarities (defult)
    - **Single**: same as above but uses smallest of similarities
    - **Average**: same as above but uses average of similarities
    - **Centroid**: finds centroid of cluster 1 and centroid of cluster 2, and uses similarity between two centroids
  + Linkage in R
    - Using different hierarchical clustering methods
      * hc.complete <- hclust(d, method = “complete”)
      * hc.average <- hclust(d, method = “average”)
      * hc.single <- hclust(d, method = “single”)

## Dimensionality reduction, visualization and “structure” analysis

* **principle component analysis (PCA)**
  + PCA converts the correlation (or lac there of) among all cells into a representation we can more readily interpret (e.g., a 2D graph)
    - Cells that are highly correlated cluster together
  + Some key points:
    - The PCs (i.e., new plot axis) are ranked by their importance
      * So PC1 is more important than PC2 which in turn is more important than PC3 etc.
    - The PCs are ranked by the amount of variance in the original data that they “capture”
      * In this example PC1 “captures” 4x more of the original variance then PC2 (44/11 = 4)
  + We actually get two main things out of a typical PCA
    - The new axis (called PCs or **Eigenvectors**) and
    - **Eigenvalues** that detail the amount of variance captured by each PC
  + Another cool thing we can get out of PCA is a quantitative report on how the original variables contributed to each PC
    - In other words, which were the most important genes that lead to the observed clustering in PC-space
    - These are often called the **loadings** and we can plot them to see which are the most important genes for the observed separation as well as outputting raked lists of genes that act to discriminate the samples
* How to do PCA in R
  + How to use the **prcompt()** function to do PCA
    - Prcompt() expects the samples to be rows and genes to be columns so we need to first transpose the matrix with the t() function
      * pca <- prcomp(t(mydata), scale = TRUE)
    - see what is returned by the prcomp() function
      * attributes(pca)
      * the returned pca$x here contains the principal components (PCs) for drawing our first graph
      * use the square of pca$sdev, to calculate how much variation in the original data each PC acoounts for
  + How to draw and interpret **PCA plots**
    - Draw A basic PC1 vs. PC2 2D plot
      * plot(pca$x[,1], pca$x[,2])
  + How to determine how much **variation each principal component accounts** for and the “intrinsic dimensionality” useful for further analysis
    - variance captured per PC
      * pca.var <- pca$sdev^2
      * pca.var.per <- round(pca.var/sum(pca.var) \* 100, 1)
    - interpret the variance
      * barplot(pca.var.per, main = “Scree Plot”, xlab = “principal component”, ylab = “percent variation”)
      * from the “scree plot” it is clear that PC1 accounts for almost all of the variation in the data, which means there are big difference between the two groups that are separated along the pC1 axis
  + How to examine the **loadings** (or loading scores) to determine what variables have the largest effect on the graph and are thus important for discriminating samples
    - The prcomp() function calls loading scores $rotation
      * loading\_scores <- pca$rotation[,1]
      * gene\_scores <- abs(loading\_scores)
      * gene\_score-ranked <- sort(gene\_scores, decreasing = TRUE)
      * top\_5\_genes <- names(gene\_score\_ranked[1:5])
      * pca$rotation[top\_5\_genes, 1]
* Main PCA objectives include
  + To reduce dimensionality
  + To visualize multidimensional data
  + To choose the most useful variables (features)
  + To identify groupings of objects (e.g. genes/samples)
  + To identify outliers
* PCA recap
  + PCA is classic “multivariate statistical technique” used to reduce the dimensionality of a complex data set to a more manageable number (typically 2D or 3D)
  + For a matrix of m genes x n samples, we mean center (i.e. subtract the sample mean from each sample column), optionally rescale the values for each sample column, then calculate a new covariance matrix of size n x n
  + We finally diagonalize the covariance matrix to yield our n Eigenvectors (called principle components or PCs) and n Eigenvalues
  + The top PCs (with largest Eigenvalues) retain the essential features of the original data and represent a useful subspace for further analysis (e.g. visualization, clustering, feature extraction, outlier detection, etc.)
* Practical issues with PCA
  + Scaling the data
    - prcomp(x, center=TRUE, scale=FALSE)
    - prcomp(x, center=TRUE, scale=TRUE)
  + Missing values
    - Drop observations with missing values
    - Impute/estimate missing values