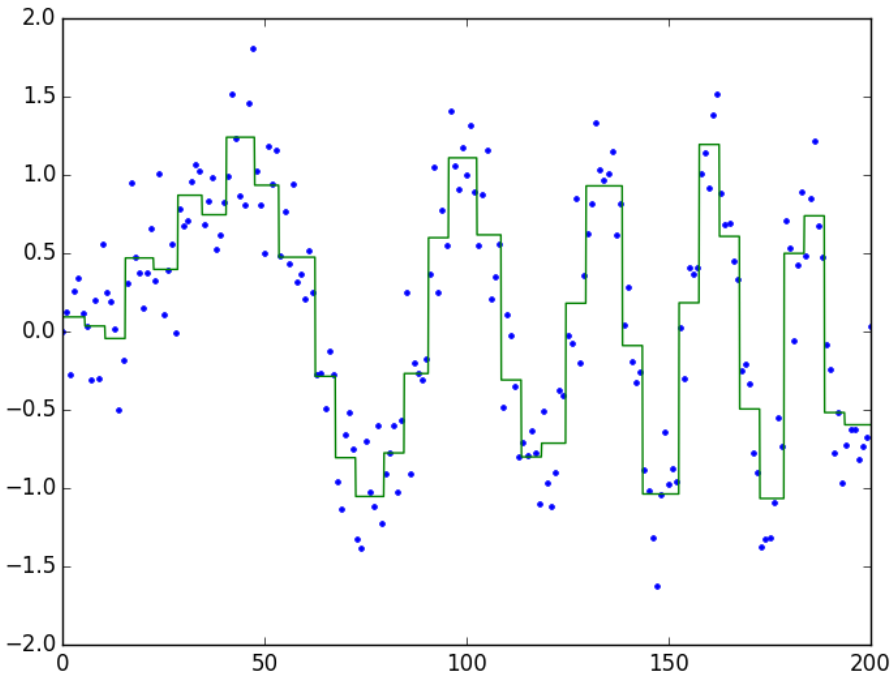
Classification and Regression Trees

- We can think of a series of a decisions as a recursive partition of the feature space as a *tree*
- Tree based models are simple to understand yet powerful
- Also interact well with *ensemble* learning methods

Trees

- Partition the feature space recursively using inequalities on the feature space
- *Classification*: The different regions correspond to different classes
- *Regression*: Approximate the value of the function in the region by a single value



Regression Model

- Our tree partitions the feature space into N regions, $R_1, R_2, \ldots, R_N$
- Our model consists of choosing the region boundaries and assigning values to the c_m

$$\hat{f}(X) = \sum_{m=1}^{m=N} c_m I[(x_1, x_2) \in R_m] \tag{1}$$

White-box Model

- It is easy to understand what the algorithm is doing by inspecting the tree that is produced
- Reproduces some of how an 'expert' thinks about a problem in his or her field: applying a sequence of conditionals and drawing some conclusion

Growing a Regression Tree

- Assume we have a two-dimensional problem and that data consists of L input pairs (x_i, y_i)
- Response is modeled by a sum of indicator functions
- Criterion is to minimize the sum of squares

$$f(X) = \sum_{m=1}^{m=N} c_m I[x \in R_m] \tag{2}$$

$$J = \sum (y_i - f(x_i))^2 \tag{3}$$

Growing a Regression Tree cont.

- For a given region R_m the best value of c_m is the average of y_i for all $x_i \in R_m$
- Finding the best possible partition is in general computationally infeasible so a *greedy* approach is used

$$c_m = E[y_i] \forall i \{ i : x_i \in R_m \} \tag{4}$$

Splitting Criterion

- For a given region R_m the best value of c_m is the average of y_i for all $x_i \in R_m$
- Finding the best possible partition is in general computationally infeasible so a *greedy* approach is used

$$\min_{j,s} \left[\min_{c_1} \sum_{R_1} (y_i - c_1)^2 + \min_{c_2} \sum_{R_2} (y_i - c_2)^2 \right] \tag{5}$$

$$R_1\{x : x(j) \leq s\} \tag{6}$$

$$R_2\{x : x(j) > s\} \tag{7}$$

Splitting Criterion

- For any feature the best split point s can be found very quickly for a given feature
- Therefore it is also easy to scan over all of the features and determine the best split point
- A given iteration outputs a pair: (feature to split over, the split point)

Tree Parameters

- Large tree might overfit the data (number of nodes)
- One option to control the complexity by only splitting if the error improves greater than some threshold value
- Other options available in learning packages include setting the maximum depth of the tree or the minimum number of nodes in a given leaf

Splitting Criterion for Classification

- For classification we split based upon a different error criterion, the mis-classification rate in each node

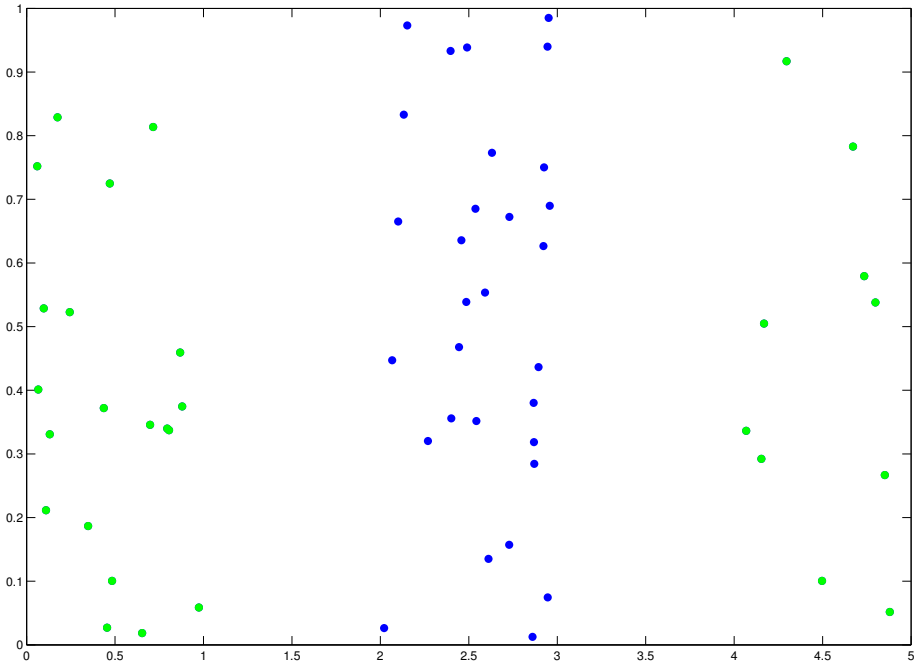
$$\frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) \tag{8}$$

Algorithm for Growing a Tree

- Split the nodes in a *greedy* fashion
- Step 1: Begin with root node containing all the training examples
- Step 2: For each feature find the split minimizes node impurity or error in the two child nodes and pick minimum over all features and all splits
- Step 3: Stop splitting when criterion is met, or apply Step 2 recursively

Simple Example

- Two dimensional case where data fits very neatly into rectangular regions
- Obviously only the first feature (horizontal axis) is important here
- Assume the three regions contain respectively 20, 30, and 10 examples



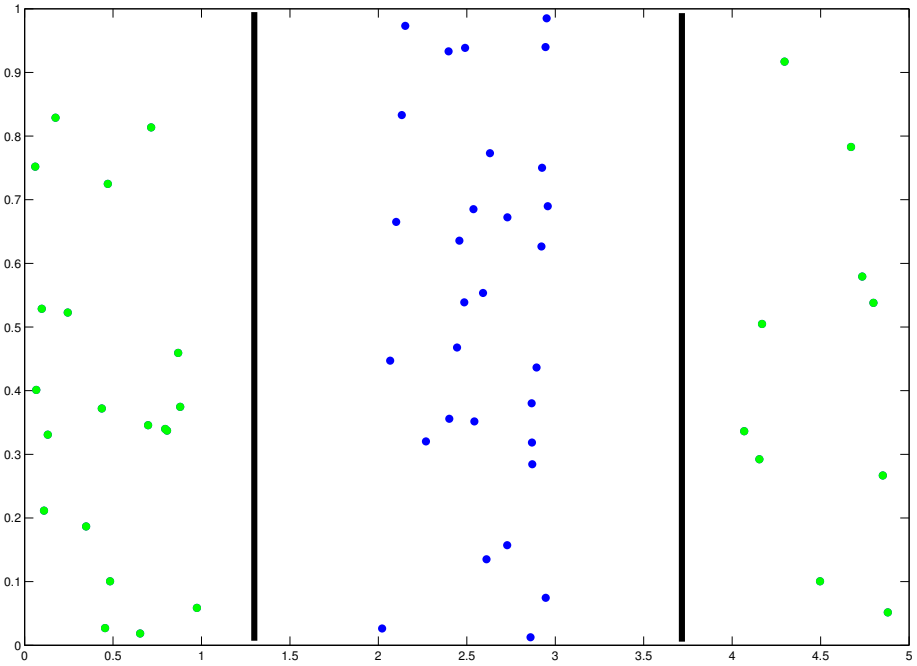
Node Error

- Classification rule: assign to each region a class by majority voting
- In previous example we had $N = 60$ training examples with 30 samples from each class
- What is the node error of the root node?

$$\text{Node Error} = 1 - p_i \tag{9}$$

Choosing a split

- We can see by inspection the potential splits in this data set, what are they?
- How can we choose amongst the potential splits, for instance if we are maximizing based upon node-error. (20, 30, 10)



Choosing a split cont.

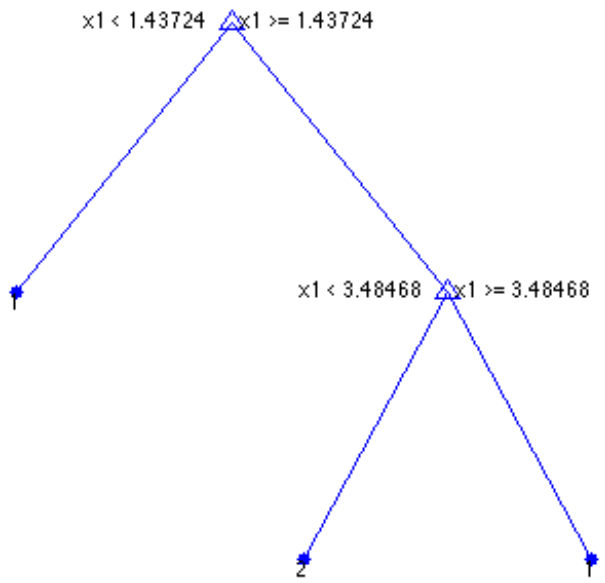
- Following the algorithm, we choose the split that minimizes the sum of the impurities in the children node

$$0 + (1 - \frac{30}{40}) = 0.25 \tag{10}$$

$$0 + (1 - \frac{30}{50}) = 0.4 \tag{11}$$

Simple Example

- We can easily visualize the structure of the classifier and understand exactly how the class labels are being assigned



Other Measures of Impurity

- We can grow the decision tree based upon a variety of impurity measures
- Gini Diversity index or cross entropy
- Zero when node is pure

$$1 - \sum_i p_i^2 \tag{12}$$

Bootstrap

- Tool for resampling a dataset
- A *bootstrap sample* is a sequence of data points (x_i, y_i) are drawn uniformly and with replacement
- Decent approximation of the answer we would get if we had more data to sample
- When the bootstrap sample is the same size as the original dataset approximately 63 % of the samples are contained

Ensemble Methods in Machine Learning

- *Goal*: To combine the predictions of many (potentially weak) estimators to improve robustness compared to any single estimator
- Averaging: Build predictors independently and average their results
- Two primary approaches here are called *bagging* and *boosting*

Bagging

- Meta-algorithm for machine learning: stands for *bootstrap aggregating*
- Averages the predictions from an ensemble of classifiers each trained on an independent bootstrap sample
- The goal is to inject diversity into the learning process but also prevent overfitting with the randomized bootstrap samples

Bagging by Majority Rule

- Meta-algorithm for machine learning: stands for *bootstrap aggregating*
- Take the most commonly predicted class from all of the classifiers

$$f_{bag}(x) = \text{argmax(k)} \sum_{b=1}^{B} I[f_b(x) = k] \tag{13}$$

Bagging in Python

- Given a classification object in python we can automatically create a bagged classifier using the syntax `BaggingClassifier(<... Learning Algorithm ... >)`
- For example if we want to create a bagged decision tree classifier we could write

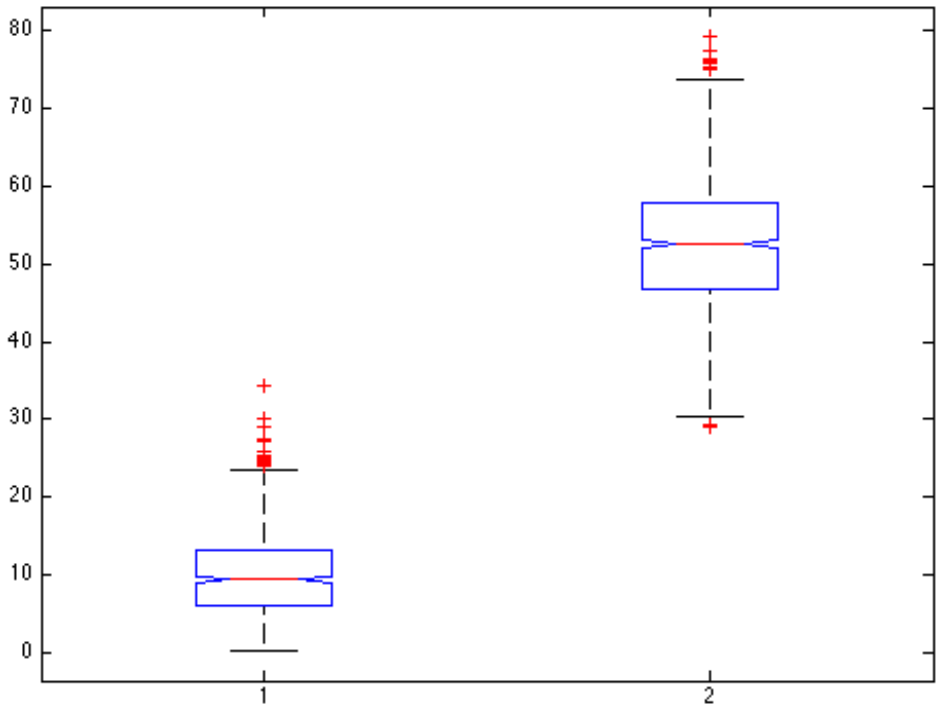
```
clf = BaggingClassifier(DecisionTreeClassifier(max_depth=2))
clf.fit(X,y)
```

Data Visualization

- Designing graphics for the communication of information
- Ranges from simple tables and plots to complex 3d renderings
- What are some examples?

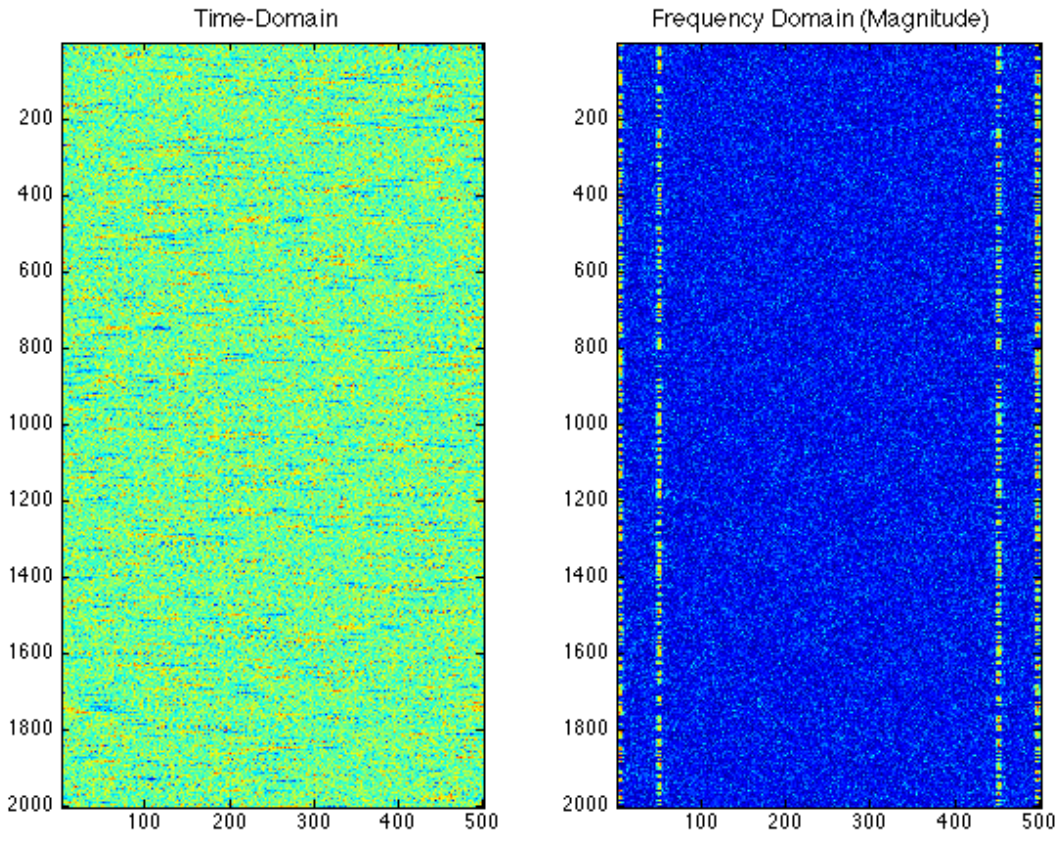
Box-plot

- Visualization of the central tendency and dispersion of a dataset, and how it differs among groups



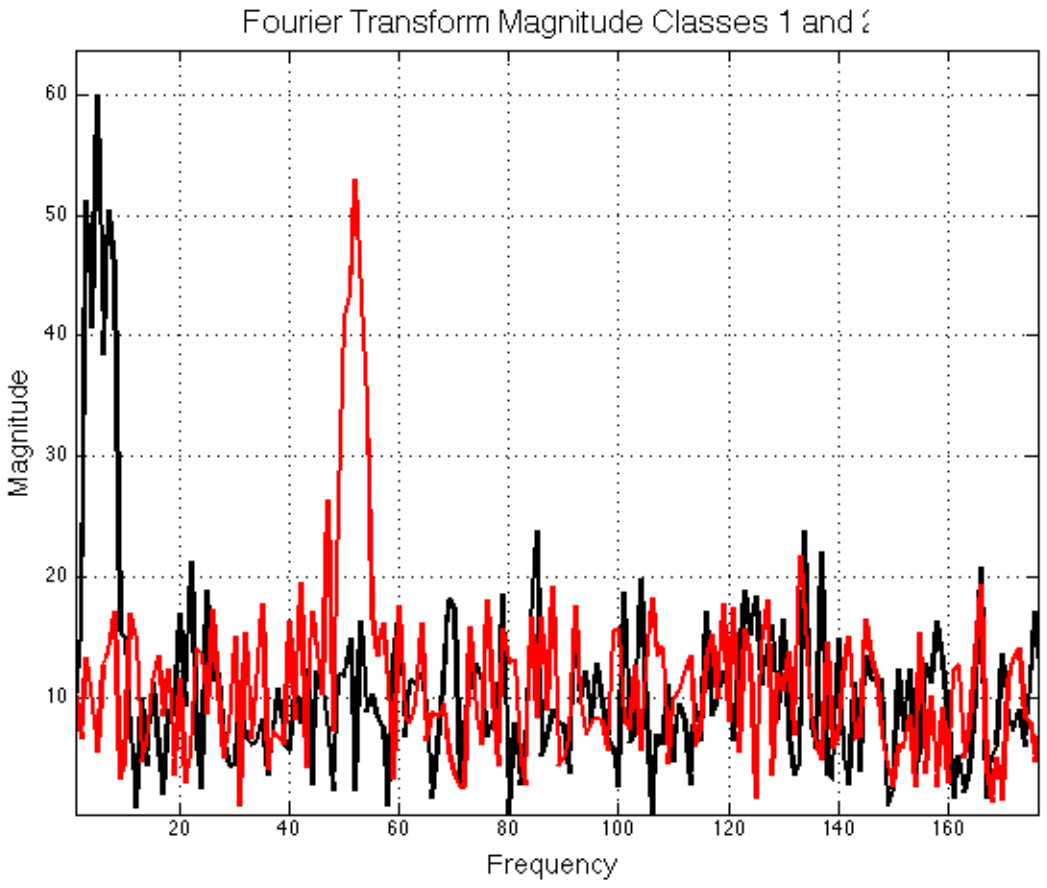
Heatmap

- Effectively communicated how our data changed as we moved from the time-domain to the frequency-domain



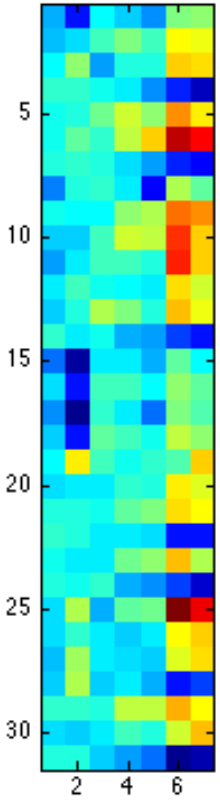
Simple Line-Plot

- Same data shown in a typical line-plot



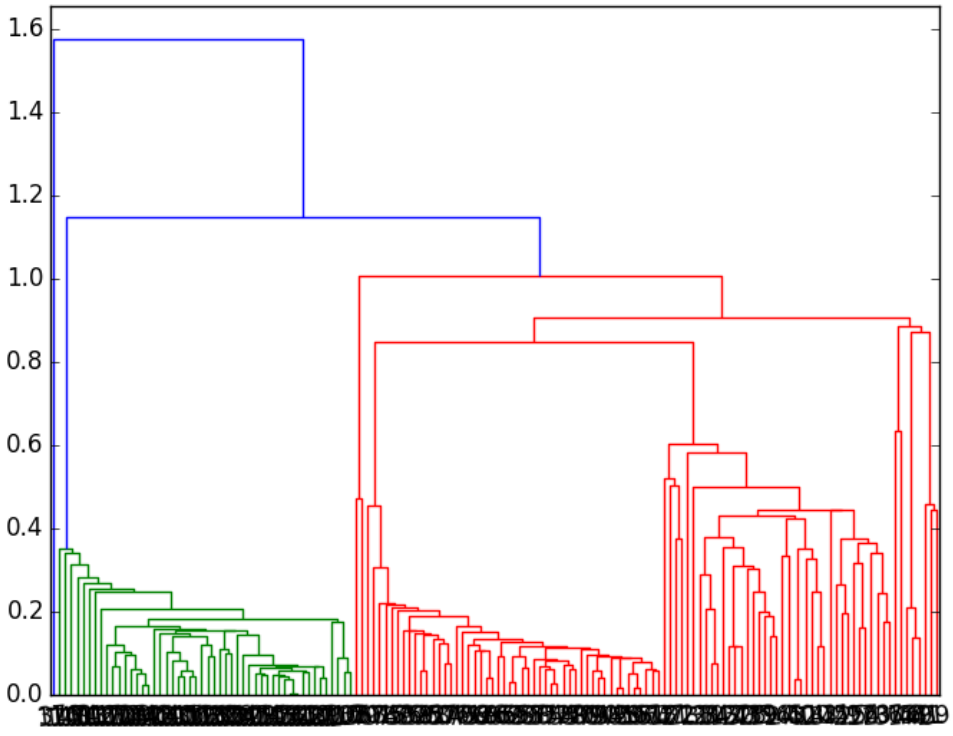
Heatmaps for 2D Arrays

- Way to visualize a two-dimensional array as an image
- Render the array as an image
- Color and intensity in block (i, j) is some function of the value in the array A_{ij}
- small values $\rightarrow$ blue, large values $\rightarrow$ red
- What manipulations can we make to this data to find the underlying structure?



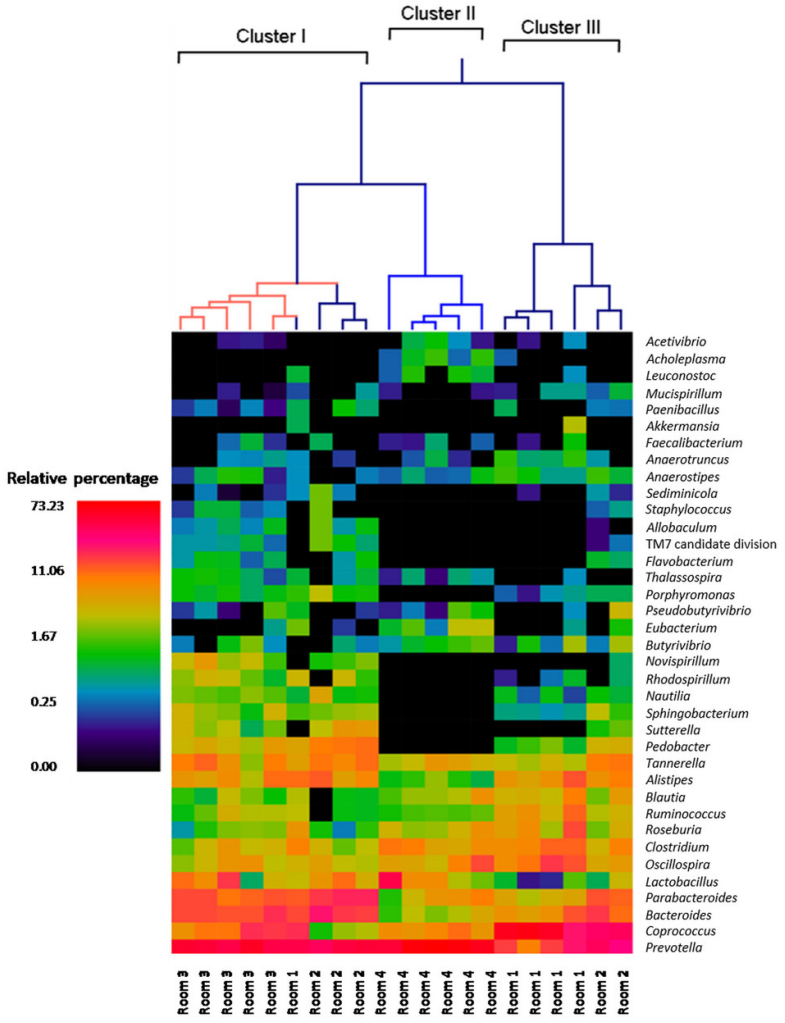
Clustering and Dendrograms

- A dendrogram is a visualization of the hierarchical clustering process



Heatmaps in Medicine

- Heatmaps, together with clustering, can be an effective visualization tool
- Used often in medical data
- Figure from Rogers, G. B., et al. "Functional divergence in gastrointestinal microbiota in physically-separated genetically identical mice." Scientific reports 4 (2014).
- This is an example that looked at the gut flora of mice living in controlled conditions



Transforming our Data

- Recall that there are two primary dimensions to consider in a data analysis problem: the number of data-instances and the number of features
- Equivalently the *row dimension* and *column dimension* of the design matrix
- In many ways the ordering of the instances or features is arbitrary
- For the purposes of visualization we can permute both the columns and rows at will

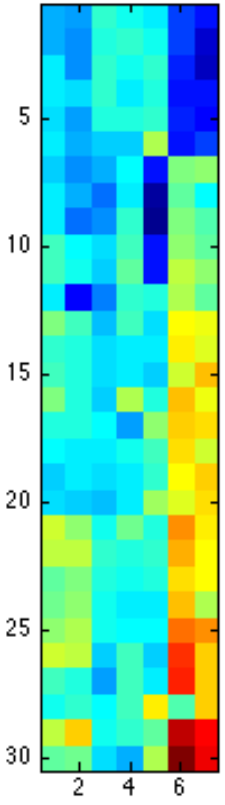
Permutations of our Design Matrix

- Mathematically we have the following

$$\hat{A}_{ij} = A_{p(i),q(j)} \tag{14}$$

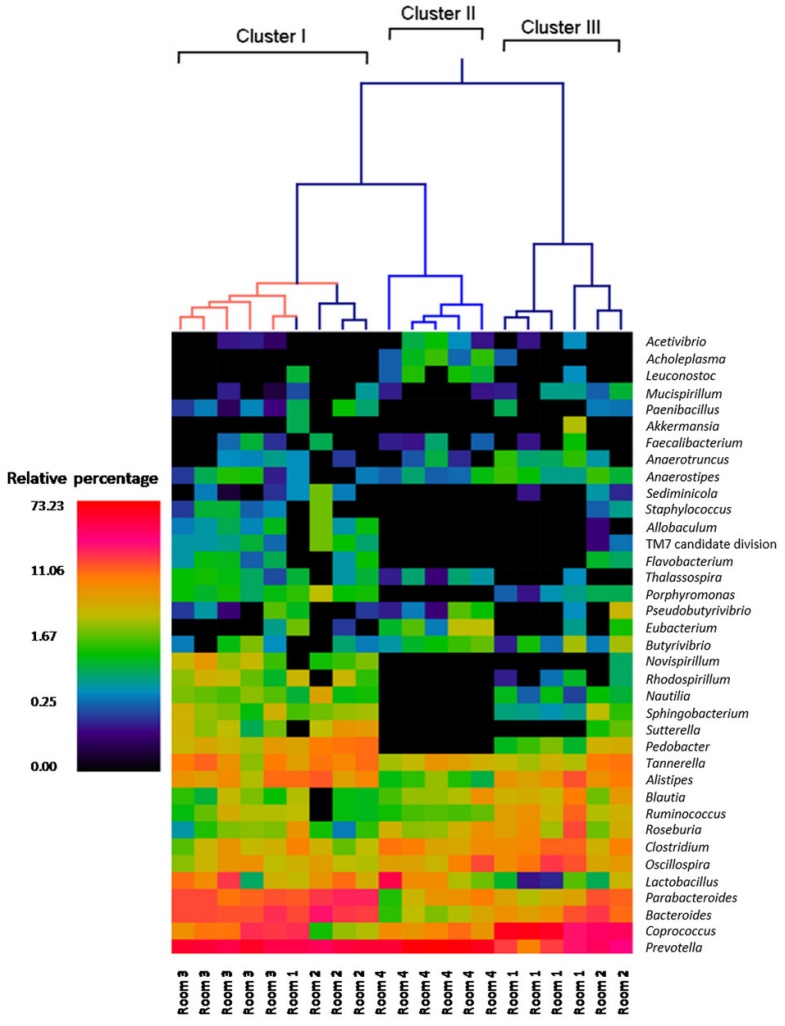
$$p = [3, 1, 6, 10, \ldots] \tag{15}$$

- Goal is to find some permutations in both the row and column dimensions that puts like things with like



Heatmaps for 2D Arrays

- Look at the example again and what conclusions should we draw from this experiment?
- Important to observe that the dendrogram effectively defines a permutation of the features



Heatmaps and Clustering in Practice

- In most cases a function or module that computes a dendrogram will return a data structure allowing you to recover the permutations
- In Python we used `scipy.cluster.hierarchy.dendrogram` to find dendrograms
- Additionally `plt.pcolor()` can be used to render a two-dimensional array

Returning a dendrogram

- Use the syntax below to obtain a data structure that contains the dendrogram
- The permutation can be found from `x=dendr['ivl']` (need an additional step, using `map` and `int` to convert this list to numerical values
- With slicing syntax the permutation can be applied as follows `A=A[perm,:]`

```
distm = pdist(vl)
Z = linkage(distm,method='average')
dendr=dendrogram(Z,no_plot='True')
```

MapReduce

- Cluster and distributed processing used in Hadoop
- Idea is to map the input data to a collection of *key-value* pairs and then perform some reduction operation over all pairs with same key
- `map()` function processes a local chunk of data and produces a sequence of k-v pairs
- `reduce()` function merges the output generated by `map()` into the desired result

List Processing in Python

- In Python we have three built-in functions for processing streams `filter`,`map`, and `reduce`
- Filter is like applying a *predicate* to each element of a sequence and only returning the results that are true
- Map applies a given function to each element of a sequence
- Reduce returns a single value by repeatedly applying a binary function to all the elements of a sequence

Reduce Examples

- The example below simply shows how to sum up all the elements of a sequence of numbers using reduce
- What are the properties of the function (addition in this case) that makes this procedure sensible?

```
from functools import reduce
def add(x,y): return x+y
...
reduce(add, [1 3 5 7 9 11])
...
a = [1,2,3,4,5]
reduce(lambda a,b: a+b,a)
...
```

List Comprehensions

- Different syntax for creating new lists from other lists
- Filter out based on a predicate, map a function to all elements

```
a = [1,2,3,4,5]
[(lambda x: x*x*2)(x) for x in a if x > 2]
S = ['bob', 'joe', 'Charlie', 'annette']
[len(s) for s in S]
[s for s in S if len(s) > 3]
```