ClusterPCAML

November 13, 2018

1 Lecture 23: Clustering and machine learning

CBIO (CSCI) 4835/6835: Introduction to Computational Biology

1.1 Overview and Objectives

It's nice when you're able to automate a lot of your data analysis. Unfortunately, quite a bit of this analysis is "fuzzy"–it doesn't have well-defined right or wrong answers, but instead relies on sophisticated algorithmic and statistical analyses. Fortunately, a lot of these analyses have been implemented in the `scikit-learn` Python library. By the end of this lecture, you should be able to:

- Define clustering, the kinds of problems it is designed to solve, and the most popular clustering variants
- Use SciPy to perform hierarchical clustering of expression data
- Define machine learning
- Understand when to use supervised versus unsupervised learning
- Create a basic classifier

1.2 Part 1: Clustering

What is **clustering**?

Wikipedia:

```
Cluster analysis or clustering is the task of grouping a set of objects in such a way that objec
```

Generally speaking, clustering is a hard problem, so it is difficult to identify a provably optimal clustering.

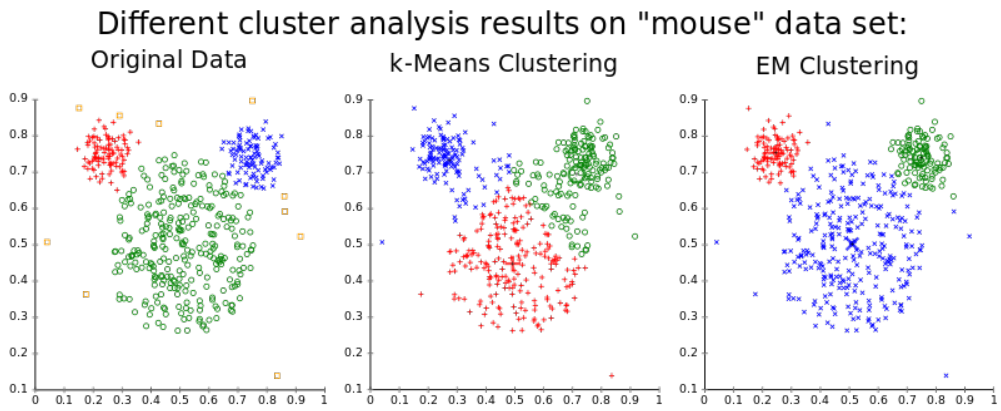
1.2.1 k-means

In *k-means clustering* we are given a set of d -dimensional vectors and we want to identify k sets S_i such that

$$\sum_{i=0}^{k} \sum_{x_j \in S_i} ||x_j - \mu_i||^2$$

is **minimized** where μ_i is the *mean* of cluster S_i. That is, all points are close as possible to the 'center' of the cluster.

Limitations



cluster

- Classical *k*-means requires that we be able to take an average of our points - *no arbitrary distance functions*.
- Must provide *k* as a parameter.
- Clustering results are very sensitive to *k*; poor choice of *k*, poor clustering results.

The general algorithm for *k*-means is as follows:

1: Choose the initial set of *k* centroids. These are essentially pseudo-datapoints that will be updated over the course of the algorithm.

2: Assign all the actual data points to one of the centroids–whichever centroid they're closest to.

3: Recompute the centroids based on the cluster assignments found in step 2.

4: Repeat until the centroids don't change very much.

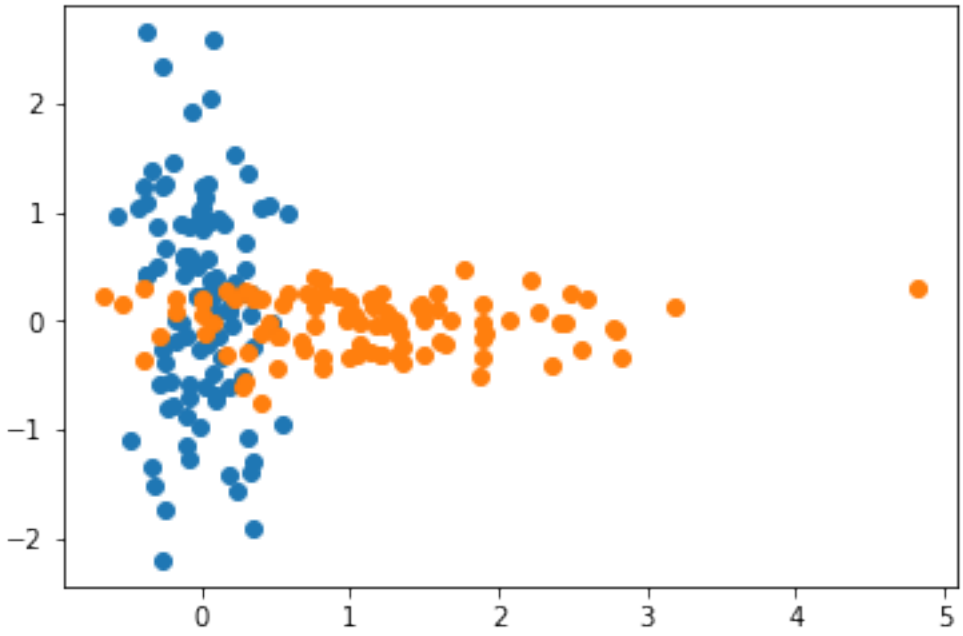
1.2.2 Visualizing *k*-means

https://www.naftaliharris.com/blog/visualizing-k-means-clustering/

1.2.3 *K*-means examples

First let's make a toy data set...

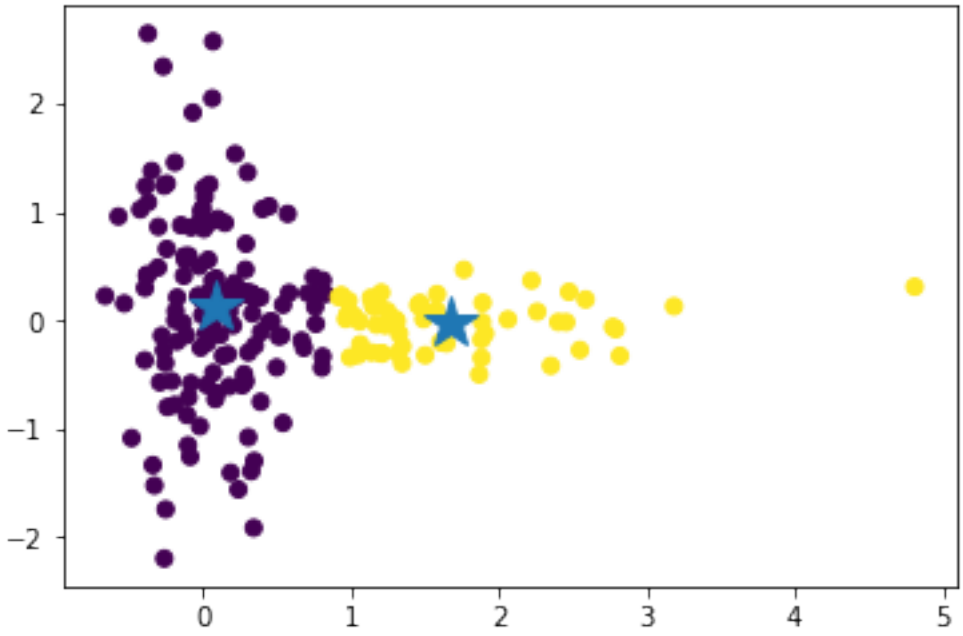
```
In [1]: %matplotlib inline
        import matplotlib.pylab as plt
        import numpy as np

In [2]: randpts1 = np.random.randn(100, 2) / (4, 1) #100 integer coordinates in the range [0:50]
        randpts2 = (np.random.randn(100, 2) + (1, 0)) / (1, 4)

        plt.plot(randpts1[:, 0], randpts1[:, 1], 'o', randpts2[:, 0], randpts2[:, 1], 'o')
        X = np.vstack((randpts1,randpts2))
```




```
In [3]: import sklearn.cluster as cluster

        kmeans = cluster.KMeans(n_clusters = 2)
        kmeans.fit(X)

        # Get the cluster assignments.
        clusters = kmeans.labels_

        # Get the centroids.
        means = kmeans.cluster_centers_
```

The means are the cluster centers

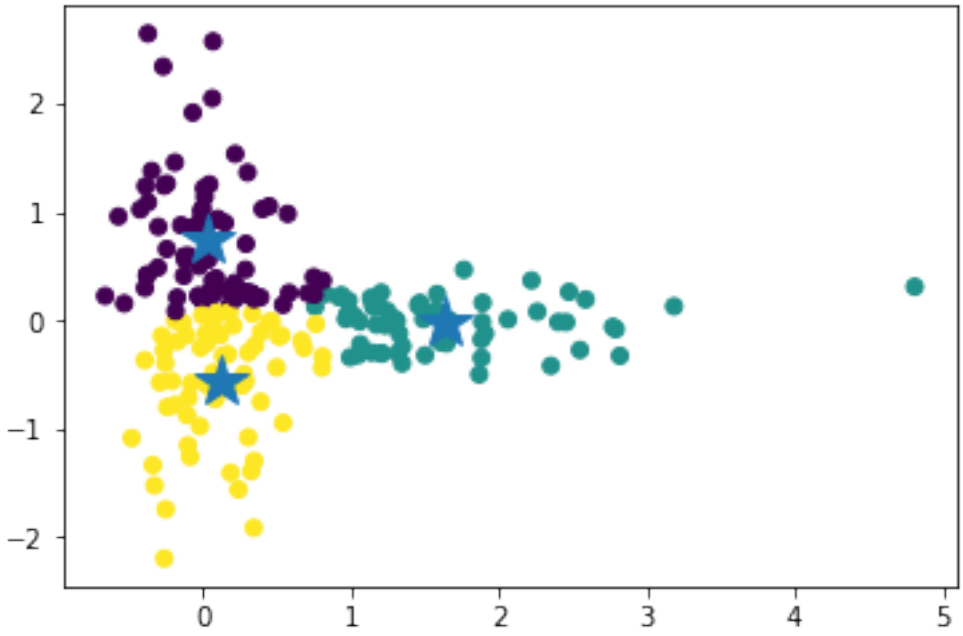
```
In [4]: plt.scatter(X[:, 0], X[:, 1], c = clusters)
        plt.plot(means[:, 0], means[:, 1], '*', ms = 20);
```



1.2.4 Changing k

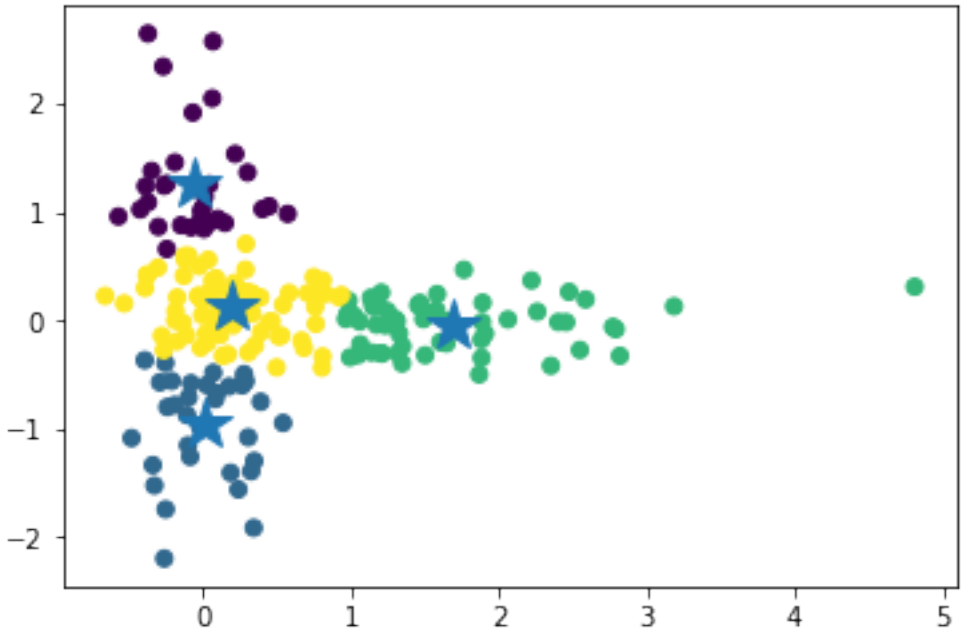
Let's look at what happens if we change from $k = 2$ to $k = 3$.

```
In [5]: kmeans = cluster.KMeans(n_clusters = 3)
        kmeans.fit(X)

        clusters, means = kmeans.labels_, kmeans.cluster_centers_

        plt.scatter(X[:, 0], X[:, 1], c = clusters)
        plt.plot(means[:, 0], means[:, 1], '*', ms = 20);
```



And for $k = 4$

```
In [6]: kmeans = cluster.KMeans(n_clusters = 4)
        kmeans.fit(X)

        clusters, means = kmeans.labels_, kmeans.cluster_centers_

        plt.scatter(X[:, 0], X[:, 1], c = clusters)
        plt.plot(means[:, 0], means[:, 1], '*', ms = 20);
```



Will K-means always find the same set of clusters?

- Yes
- No

1.2.5 Hierarchical clustering

Hierarchical clustering creates a heirarchy, where each cluster is formed from subclusters.

There are two kinds of hierarchical clustering: *agglomerative* and *divisive*.

- *Agglomerative* clustering builds the hierarchy from the bottom up: start with all data points as their own clusters, find the two clusters that are closest, combine them into a cluster, repeat.

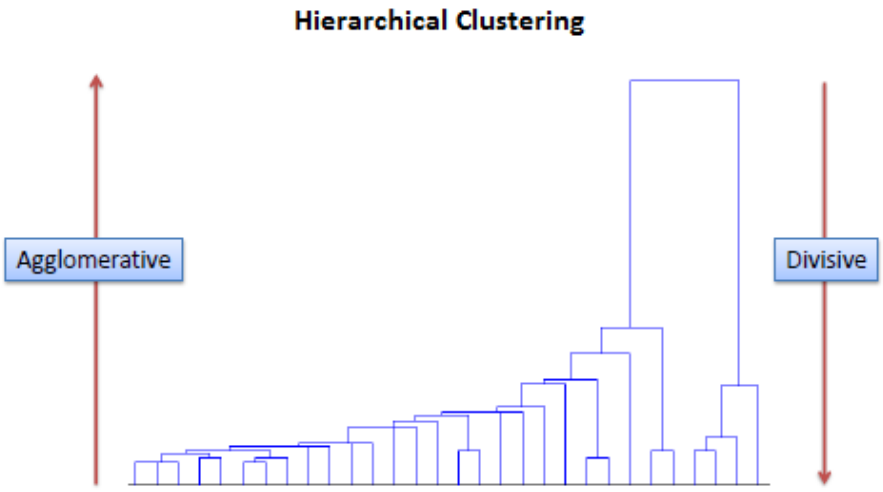
- *Divisive* clustering is the opposite: start with all data points as part of a single huge cluster, find the groups that are most different, and split them into separate clusters. Repeat.

Which do you think is easier, in practice?

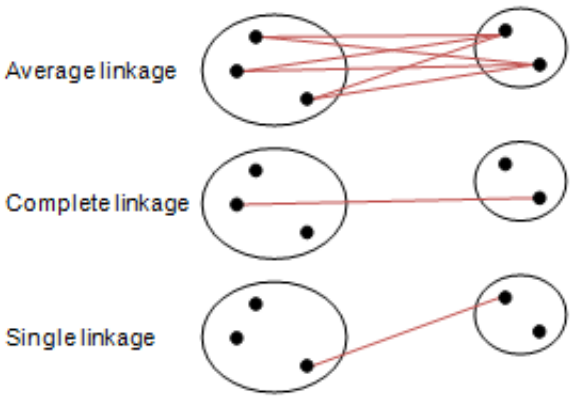
1.2.6 Agglomerative clustering

Agglomerative clustering requires there be a notion of distance between *clusters of items*, not just the items themselves (why?).

On the other hand, all you need is a distance function. You do *not* need to be able to take an average, as with k-means.



hierarchy



distances

1.2.7 Distance (Linkage) Methods

- **average**:

$$d(u,v) = \sum_{ij} \frac{d(u_i, v_j)}{|u||v|}$$

- **complete** or farthest point:

$$d(u,v) = \max(dist(u_i, v_j))$$

- **single** or nearest point:

$$d(u,v) = \min(dist(u_i, v_j))$$

1.2.8 `linkage`

`scipy.cluster.hierarchy.linkage` creates a clustering hierarchy. It takes three parameters: * *y* the data or a precalculated distance matrix * *method* the linkage method (default single) * *metric* the distance metric to use (default euclidean)

```
In [7]: import scipy.cluster.hierarchy as hclust
        Z = hclust.linkage(X)
```

- An $(n-1) \times 4$ matrix Z is returned.
- At the i^{th} iteration, clusters with indices `Z[i, 0]` and `Z[i, 1]` are combined to form cluster $n+i$.
- A cluster with an index less than n corresponds to one of the n original observations.
- The distance between clusters `Z[i, 0]` and `Z[i, 1]` is given by `Z[i, 2]`.
- The fourth value `Z[i, 3]` represents the number of original observations in the newly formed cluster.

```
In [8]: print(X.shape)
        print(Z)
        print(Z.shape)
```

```
(200, 2)
[[5.20000000e+01 8.00000000e+01 5.46104484e-03 2.00000000e+00]
 [1.11000000e+02 1.20000000e+02 5.75628114e-03 2.00000000e+00]
 [7.60000000e+01 1.99000000e+02 7.59955496e-03 2.00000000e+00]
 [5.00000000e+00 4.20000000e+01 1.12296562e-02 2.00000000e+00]
 [3.70000000e+01 9.30000000e+01 1.62556168e-02 2.00000000e+00]
 [6.90000000e+01 1.89000000e+02 1.66985682e-02 2.00000000e+00]
 [1.04000000e+02 1.29000000e+02 1.68648446e-02 2.00000000e+00]
 [2.02000000e+02 2.04000000e+02 1.76761477e-02 4.00000000e+00]
 [5.30000000e+01 9.70000000e+01 1.82930831e-02 2.00000000e+00]
 [1.27000000e+02 1.45000000e+02 1.92537400e-02 2.00000000e+00]
 [2.00000000e+00 1.25000000e+02 2.15373254e-02 2.00000000e+00]
 [1.54000000e+02 1.98000000e+02 2.37754345e-02 2.00000000e+00]
 [1.69000000e+02 1.88000000e+02 2.51246631e-02 2.00000000e+00]
 [1.64000000e+02 1.80000000e+02 2.51378410e-02 2.00000000e+00]
 [6.60000000e+01 2.08000000e+02 2.52241798e-02 3.00000000e+00]
 [1.82000000e+02 1.86000000e+02 2.53160649e-02 2.00000000e+00]
```

[1.40000000e+02 2.07000000e+02 2.54982005e-02 5.00000000e+00]
[1.33000000e+02 2.15000000e+02 2.64342910e-02 3.00000000e+00]
[1.70000000e+01 3.20000000e+01 2.68497541e-02 2.00000000e+00]
[9.60000000e+01 2.14000000e+02 2.74023100e-02 4.00000000e+00]
[3.50000000e+01 7.40000000e+01 2.74320473e-02 2.00000000e+00]
[1.66000000e+02 1.97000000e+02 2.80245139e-02 2.00000000e+00]
[6.50000000e+01 7.80000000e+01 3.06791051e-02 2.00000000e+00]
[6.80000000e+01 1.87000000e+02 3.70320570e-02 2.00000000e+00]
[2.00000000e+02 2.06000000e+02 3.92033318e-02 4.00000000e+00]
[1.84000000e+02 2.11000000e+02 4.24365499e-02 3.00000000e+00]
[1.48000000e+02 1.68000000e+02 4.29218054e-02 2.00000000e+00]
[1.09000000e+02 2.05000000e+02 4.31315099e-02 3.00000000e+00]
[1.12000000e+02 1.74000000e+02 4.40724573e-02 2.00000000e+00]
[1.81000000e+02 1.85000000e+02 4.50069137e-02 2.00000000e+00]
[2.20000000e+01 5.80000000e+01 4.63821952e-02 2.00000000e+00]
[1.52000000e+02 1.95000000e+02 4.64960718e-02 2.00000000e+00]
[1.39000000e+02 1.49000000e+02 4.91215816e-02 2.00000000e+00]
[1.67000000e+02 2.28000000e+02 4.91481129e-02 3.00000000e+00]
[2.13000000e+02 2.32000000e+02 4.99346472e-02 4.00000000e+00]
[1.00000000e+01 9.50000000e+01 5.09135189e-02 2.00000000e+00]
[8.70000000e+01 2.27000000e+02 5.19649423e-02 4.00000000e+00]
[2.16000000e+02 2.29000000e+02 5.21021928e-02 7.00000000e+00]
[1.20000000e+01 2.60000000e+01 5.23820370e-02 2.00000000e+00]
[1.19000000e+02 1.38000000e+02 5.27714079e-02 2.00000000e+00]
[1.80000000e+01 8.80000000e+01 5.37551891e-02 2.00000000e+00]
[4.70000000e+01 4.90000000e+01 5.56824071e-02 2.00000000e+00]
[2.70000000e+01 3.80000000e+01 5.62741872e-02 2.00000000e+00]
[4.30000000e+01 9.90000000e+01 5.76203041e-02 2.00000000e+00]
[1.35000000e+02 1.71000000e+02 5.88990492e-02 2.00000000e+00]
[1.24000000e+02 1.93000000e+02 5.92076799e-02 2.00000000e+00]
[3.00000000e+01 7.30000000e+01 6.06804712e-02 2.00000000e+00]
[1.22000000e+02 1.53000000e+02 6.09012377e-02 2.00000000e+00]
[8.10000000e+01 2.43000000e+02 6.18105490e-02 3.00000000e+00]
[2.17000000e+02 2.45000000e+02 6.37165207e-02 5.00000000e+00]
[1.37000000e+02 2.01000000e+02 6.51261109e-02 3.00000000e+00]
[4.00000000e+00 5.10000000e+01 6.52059129e-02 2.00000000e+00]
[1.21000000e+02 2.33000000e+02 6.83279214e-02 4.00000000e+00]
[1.62000000e+02 1.76000000e+02 6.85134492e-02 2.00000000e+00]
[1.00000000e+02 1.18000000e+02 6.90845167e-02 2.00000000e+00]
[1.90000000e+01 2.48000000e+02 6.95604991e-02 4.00000000e+00]
[2.24000000e+02 2.30000000e+02 7.04579946e-02 6.00000000e+00]
[1.55000000e+02 2.12000000e+02 7.08230849e-02 3.00000000e+00]
[2.30000000e+01 6.20000000e+01 7.09406284e-02 2.00000000e+00]
[1.31000000e+02 1.34000000e+02 7.26929488e-02 2.00000000e+00]
[1.07000000e+02 2.31000000e+02 7.46948814e-02 3.00000000e+00]
[2.20000000e+02 2.55000000e+02 7.58288027e-02 6.00000000e+00]
[1.02000000e+02 2.34000000e+02 7.59303128e-02 5.00000000e+00]
[6.10000000e+01 2.61000000e+02 7.73984617e-02 7.00000000e+00]

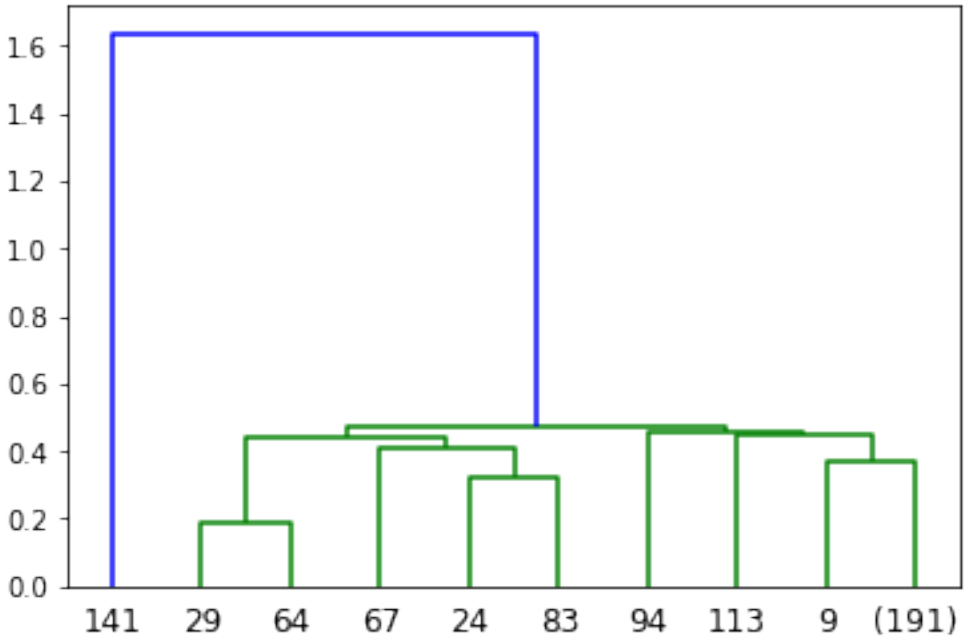
[5.00000000e+01 1.26000000e+02 7.88428385e-02 2.00000000e+00]
[2.50000000e+01 2.35000000e+02 7.97682292e-02 3.00000000e+00]
[1.90000000e+02 2.46000000e+02 8.02371819e-02 3.00000000e+00]
[1.30000000e+01 8.20000000e+01 8.08864686e-02 2.00000000e+00]
[3.90000000e+01 1.05000000e+02 8.41675288e-02 2.00000000e+00]
[6.00000000e+00 2.00000000e+01 8.42138379e-02 2.00000000e+00]
[1.10000000e+01 1.78000000e+02 8.44222730e-02 2.00000000e+00]
[3.40000000e+01 3.60000000e+01 8.50911747e-02 2.00000000e+00]
[1.60000000e+02 2.51000000e+02 8.53278659e-02 3.00000000e+00]
[1.43000000e+02 2.42000000e+02 8.58837575e-02 3.00000000e+00]
[2.37000000e+02 2.66000000e+02 8.60452886e-02 1.00000000e+01]
[2.68000000e+02 2.74000000e+02 8.67044575e-02 1.20000000e+01]
[2.80000000e+01 7.00000000e+01 8.71572648e-02 2.00000000e+00]
[2.58000000e+02 2.63000000e+02 8.77596672e-02 9.00000000e+00]
[1.79000000e+02 2.49000000e+02 8.85430333e-02 6.00000000e+00]
[3.30000000e+01 2.36000000e+02 8.89000879e-02 5.00000000e+00]
[2.44000000e+02 2.50000000e+02 9.37205817e-02 5.00000000e+00]
[2.57000000e+02 2.60000000e+02 9.51291373e-02 6.00000000e+00]
[5.60000000e+01 2.03000000e+02 9.54871891e-02 3.00000000e+00]
[1.28000000e+02 2.25000000e+02 9.56309610e-02 4.00000000e+00]
[1.58000000e+02 2.75000000e+02 9.57814345e-02 1.30000000e+01]
[1.16000000e+02 2.83000000e+02 9.66753466e-02 5.00000000e+00]
[8.60000000e+01 2.72000000e+02 9.67825389e-02 4.00000000e+00]
[0.00000000e+00 5.40000000e+01 9.70908824e-02 2.00000000e+00]
[2.22000000e+02 2.56000000e+02 9.77674402e-02 8.00000000e+00]
[2.62000000e+02 2.78000000e+02 9.79483907e-02 1.10000000e+01]
[1.92000000e+02 2.59000000e+02 9.84852882e-02 3.00000000e+00]
[2.70000000e+02 2.84000000e+02 9.85813329e-02 1.50000000e+01]
[1.15000000e+02 2.80000000e+02 9.99914900e-02 6.00000000e+00]
[4.60000000e+01 2.41000000e+02 1.00661998e-01 3.00000000e+00]
[1.14000000e+02 1.44000000e+02 1.03788610e-01 2.00000000e+00]
[2.79000000e+02 2.93000000e+02 1.03931746e-01 8.00000000e+00]
[1.75000000e+02 2.10000000e+02 1.05717098e-01 3.00000000e+00]
[2.90000000e+02 2.92000000e+02 1.06098327e-01 9.00000000e+00]
[2.65000000e+02 2.77000000e+02 1.06800099e-01 1.20000000e+01]
[4.00000000e+01 9.10000000e+01 1.07292264e-01 2.00000000e+00]
[4.50000000e+01 2.95000000e+02 1.08484097e-01 9.00000000e+00]
[1.77000000e+02 2.47000000e+02 1.08852137e-01 3.00000000e+00]
[2.19000000e+02 2.76000000e+02 1.09090825e-01 6.00000000e+00]
[2.88000000e+02 2.91000000e+02 1.09786346e-01 2.30000000e+01]
[1.01000000e+02 2.81000000e+02 1.12035261e-01 7.00000000e+00]
[1.72000000e+02 2.89000000e+02 1.14942069e-01 1.20000000e+01]
[1.94000000e+02 2.82000000e+02 1.15164543e-01 4.00000000e+00]
[1.30000000e+02 2.71000000e+02 1.15655128e-01 3.00000000e+00]
[7.10000000e+01 2.40000000e+02 1.15842832e-01 3.00000000e+00]
[2.73000000e+02 3.07000000e+02 1.16903362e-01 6.00000000e+00]
[1.32000000e+02 1.46000000e+02 1.17822282e-01 2.00000000e+00]
[2.52000000e+02 3.05000000e+02 1.17945727e-01 1.60000000e+01]

[5.70000000e+01 7.70000000e+01 1.18991653e-01 2.00000000e+00]
[2.97000000e+02 3.11000000e+02 1.19696581e-01 2.50000000e+01]
[3.09000000e+02 3.12000000e+02 1.19855372e-01 8.00000000e+00]
[2.86000000e+02 3.03000000e+02 1.20095970e-01 2.70000000e+01]
[3.04000000e+02 3.13000000e+02 1.20787565e-01 3.20000000e+01]
[1.00000000e+00 3.00000000e+02 1.21281830e-01 1.00000000e+01]
[7.20000000e+01 3.15000000e+02 1.23601742e-01 2.80000000e+01]
[7.00000000e+00 3.08000000e+02 1.24623619e-01 4.00000000e+00]
[9.20000000e+01 2.18000000e+02 1.24892863e-01 3.00000000e+00]
[3.10000000e+02 3.16000000e+02 1.25424788e-01 3.40000000e+01]
[1.61000000e+02 3.14000000e+02 1.26216088e-01 9.00000000e+00]
[2.53000000e+02 3.21000000e+02 1.26366686e-01 3.60000000e+01]
[2.09000000e+02 2.96000000e+02 1.26812148e-01 5.00000000e+00]
[2.64000000e+02 3.24000000e+02 1.29828179e-01 7.00000000e+00]
[2.67000000e+02 3.17000000e+02 1.31330691e-01 1.20000000e+01]
[1.08000000e+02 2.85000000e+02 1.31767021e-01 6.00000000e+00]
[1.03000000e+02 1.63000000e+02 1.31784837e-01 2.00000000e+00]
[3.19000000e+02 3.26000000e+02 1.32254190e-01 1.60000000e+01]
[3.18000000e+02 3.22000000e+02 1.33870019e-01 3.70000000e+01]
[9.80000000e+01 3.30000000e+02 1.34568018e-01 3.80000000e+01]
[8.00000000e+00 3.29000000e+02 1.35095719e-01 1.70000000e+01]
[9.00000000e+01 3.02000000e+02 1.35246171e-01 7.00000000e+00]
[5.90000000e+01 3.33000000e+02 1.35931215e-01 8.00000000e+00]
[3.25000000e+02 3.31000000e+02 1.36837919e-01 4.50000000e+01]
[2.23000000e+02 3.35000000e+02 1.39329340e-01 4.70000000e+01]
[6.00000000e+01 2.87000000e+02 1.39769183e-01 3.00000000e+00]
[7.90000000e+01 3.36000000e+02 1.41101153e-01 4.80000000e+01]
[1.96000000e+02 3.38000000e+02 1.41430441e-01 4.90000000e+01]
[1.65000000e+02 3.27000000e+02 1.41821983e-01 7.00000000e+00]
[1.40000000e+01 2.38000000e+02 1.43472308e-01 3.00000000e+00]
[2.54000000e+02 3.39000000e+02 1.44521027e-01 5.10000000e+01]
[6.30000000e+01 3.20000000e+02 1.46267450e-01 4.00000000e+00]
[1.17000000e+02 1.70000000e+02 1.46414858e-01 2.00000000e+00]
[2.94000000e+02 3.42000000e+02 1.49234965e-01 5.30000000e+01]
[2.69000000e+02 3.43000000e+02 1.50309694e-01 6.00000000e+00]
[1.06000000e+02 1.59000000e+02 1.53042405e-01 2.00000000e+00]
[2.10000000e+01 2.98000000e+02 1.54187051e-01 1.30000000e+01]
[1.47000000e+02 2.26000000e+02 1.57906342e-01 3.00000000e+00]
[3.34000000e+02 3.45000000e+02 1.58061785e-01 6.10000000e+01]
[1.91000000e+02 3.23000000e+02 1.59022668e-01 3.70000000e+01]
[4.40000000e+01 3.46000000e+02 1.61010390e-01 7.00000000e+00]
[1.50000000e+01 3.37000000e+02 1.62822433e-01 4.00000000e+00]
[3.50000000e+02 3.51000000e+02 1.67545187e-01 9.80000000e+01]
[3.32000000e+02 3.54000000e+02 1.69315358e-01 1.15000000e+02]
[3.01000000e+02 3.47000000e+02 1.69828811e-01 5.00000000e+00]
[1.51000000e+02 2.39000000e+02 1.71440445e-01 3.00000000e+00]
[3.49000000e+02 3.55000000e+02 1.73524788e-01 1.18000000e+02]
[4.10000000e+01 3.52000000e+02 1.75175132e-01 8.00000000e+00]

```
 [1.83000000e+02 3.56000000e+02 1.76226270e-01 6.00000000e+00]
 [1.36000000e+02 3.60000000e+02 1.78237388e-01 7.00000000e+00]
 [3.40000000e+02 3.58000000e+02 1.80784229e-01 1.25000000e+02]
 [3.06000000e+02 3.62000000e+02 1.81506868e-01 1.29000000e+02]
 [2.90000000e+01 6.40000000e+01 1.86049578e-01 2.00000000e+00]
 [4.80000000e+01 8.90000000e+01 1.88331420e-01 2.00000000e+00]
 [1.60000000e+01 8.40000000e+01 1.90397523e-01 2.00000000e+00]
 [2.99000000e+02 3.63000000e+02 1.93369558e-01 1.31000000e+02]
 [1.57000000e+02 3.67000000e+02 1.96146161e-01 1.32000000e+02]
 [3.44000000e+02 3.68000000e+02 2.00197552e-01 1.34000000e+02]
 [3.48000000e+02 3.59000000e+02 2.01632667e-01 2.10000000e+01]
 [3.57000000e+02 3.61000000e+02 2.10655000e-01 1.00000000e+01]
 [3.69000000e+02 3.70000000e+02 2.12689848e-01 1.55000000e+02]
 [3.71000000e+02 3.72000000e+02 2.18545132e-01 1.65000000e+02]
 [3.10000000e+01 3.53000000e+02 2.22136075e-01 5.00000000e+00]
 [1.10000000e+02 3.73000000e+02 2.28186331e-01 1.66000000e+02]
 [7.50000000e+01 3.65000000e+02 2.32665220e-01 3.00000000e+00]
 [5.50000000e+01 3.75000000e+02 2.38367056e-01 1.67000000e+02]
 [1.23000000e+02 1.42000000e+02 2.44587202e-01 2.00000000e+00]
 [8.50000000e+01 3.77000000e+02 2.47230287e-01 1.68000000e+02]
 [1.56000000e+02 2.21000000e+02 2.47713864e-01 3.00000000e+00]
 [3.28000000e+02 3.79000000e+02 2.48280109e-01 1.70000000e+02]
 [3.76000000e+02 3.81000000e+02 2.63914288e-01 1.73000000e+02]
 [3.74000000e+02 3.82000000e+02 2.70041931e-01 1.78000000e+02]
 [3.78000000e+02 3.83000000e+02 2.72915814e-01 1.80000000e+02]
 [1.50000000e+02 3.84000000e+02 2.75340289e-01 1.81000000e+02]
 [3.80000000e+02 3.85000000e+02 2.75672891e-01 1.84000000e+02]
 [3.66000000e+02 3.86000000e+02 2.80256045e-01 1.86000000e+02]
 [3.41000000e+02 3.87000000e+02 2.83980403e-01 1.89000000e+02]
 [3.00000000e+00 3.88000000e+02 2.87758359e-01 1.90000000e+02]
 [1.73000000e+02 3.89000000e+02 2.88985823e-01 1.91000000e+02]
 [2.40000000e+01 8.30000000e+01 3.21498946e-01 2.00000000e+00]
 [9.00000000e+00 3.90000000e+02 3.69645127e-01 1.92000000e+02]
 [6.70000000e+01 3.91000000e+02 4.08793535e-01 3.00000000e+00]
 [3.64000000e+02 3.93000000e+02 4.42974340e-01 5.00000000e+00]
 [1.13000000e+02 3.92000000e+02 4.47154332e-01 1.93000000e+02]
 [9.40000000e+01 3.95000000e+02 4.54364339e-01 1.94000000e+02]
 [3.94000000e+02 3.96000000e+02 4.75879988e-01 1.99000000e+02]
 [1.41000000e+02 3.97000000e+02 1.63626027e+00 2.00000000e+02]]
(199, 4)
```

1.2.9 Dendograms

```
In [9]: #show first 4 levels
        hclust.dendrogram(Z, p = 4, truncate_mode = 'level');
```



1.2.10 `fcluster`: extracting clusters from a hierarchy

`fcluster` takes a linkage matrix and returns a cluster assignment. It takes a threshold value and a string specifying what method to use to form the cluster.

```
In [10]: help(hclust.fcluster)
```

```
Help on function fcluster in module scipy.cluster.hierarchy:

fcluster(Z, t, criterion='inconsistent', depth=2, R=None, monocrit=None)
    Form flat clusters from the hierarchical clustering defined by
    the given linkage matrix.

    Parameters
    ----------
    Z : ndarray
        The hierarchical clustering encoded with the matrix returned
        by the `linkage` function.
    t : float
        The threshold to apply when forming flat clusters.
    criterion : str, optional
        The criterion to use in forming flat clusters. This can
        be any of the following values:

          ``inconsistent`` :
              If a cluster node and all its
```

```
            descendants have an inconsistent value less than or equal
            to `t` then all its leaf descendants belong to the
            same flat cluster. When no non-singleton cluster meets
            this criterion, every node is assigned to its own
            cluster. (Default)

        ``distance`` :
            Forms flat clusters so that the original
            observations in each flat cluster have no greater a
            cophenetic distance than `t`.

        ``maxclust`` :
            Finds a minimum threshold ``r`` so that
            the cophenetic distance between any two original
            observations in the same flat cluster is no more than
            ``r`` and no more than `t` flat clusters are formed.

        ``monocrit`` :
            Forms a flat cluster from a cluster node c
            with index i when ``monocrit[j] <= t``.

            For example, to threshold on the maximum mean distance
            as computed in the inconsistency matrix R with a
            threshold of 0.8 do::

                MR = maxRstat(Z, R, 3)
                cluster(Z, t=0.8, criterion='monocrit', monocrit=MR)

        ``maxclust_monocrit`` :
            Forms a flat cluster from a
            non-singleton cluster node ``c`` when ``monocrit[i] <=
            r`` for all cluster indices ``i`` below and including
            ``c``. ``r`` is minimized such that no more than ``t``
            flat clusters are formed. monocrit must be
            monotonic. For example, to minimize the threshold t on
            maximum inconsistency values so that no more than 3 flat
            clusters are formed, do::

                MI = maxinconsts(Z, R)
                cluster(Z, t=3, criterion='maxclust_monocrit', monocrit=MI)

    depth : int, optional
        The maximum depth to perform the inconsistency calculation.
        It has no meaning for the other criteria. Default is 2.
    R : ndarray, optional
        The inconsistency matrix to use for the 'inconsistent'
        criterion. This matrix is computed if not provided.
    monocrit : ndarray, optional
```

```
        An array of length n-1. `monocrit[i]` is the
        statistics upon which non-singleton i is thresholded. The
        monocrit vector must be monotonic, i.e. given a node c with
        index i, for all node indices j corresponding to nodes
        below c, ``monocrit[i] >= monocrit[j]``.

    Returns
    -------
    fcluster : ndarray
        An array of length ``n``. ``T[i]`` is the flat cluster number to
        which original observation ``i`` belongs.
```

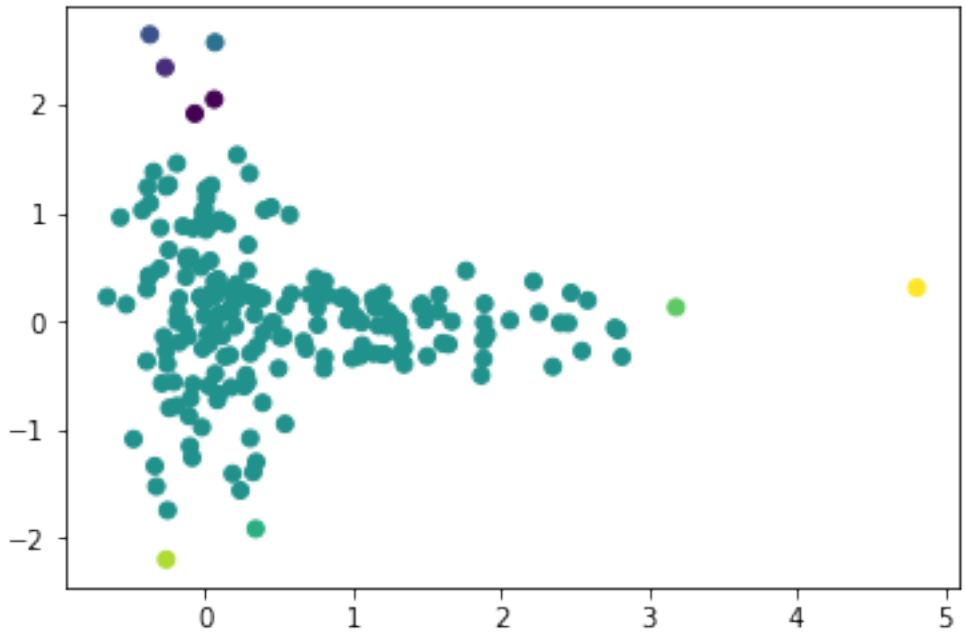
1.2.11 Flatten based on distance threshold

```
In [11]: clusters = hclust.fcluster(Z, 0.3, 'distance')
         len(set(clusters))
```

```
Out[11]: 9
```

```
In [12]: plt.scatter(X[:, 0], X[:, 1], c = clusters)
```

```
Out[12]: <matplotlib.collections.PathCollection at 0x105471e48>
```



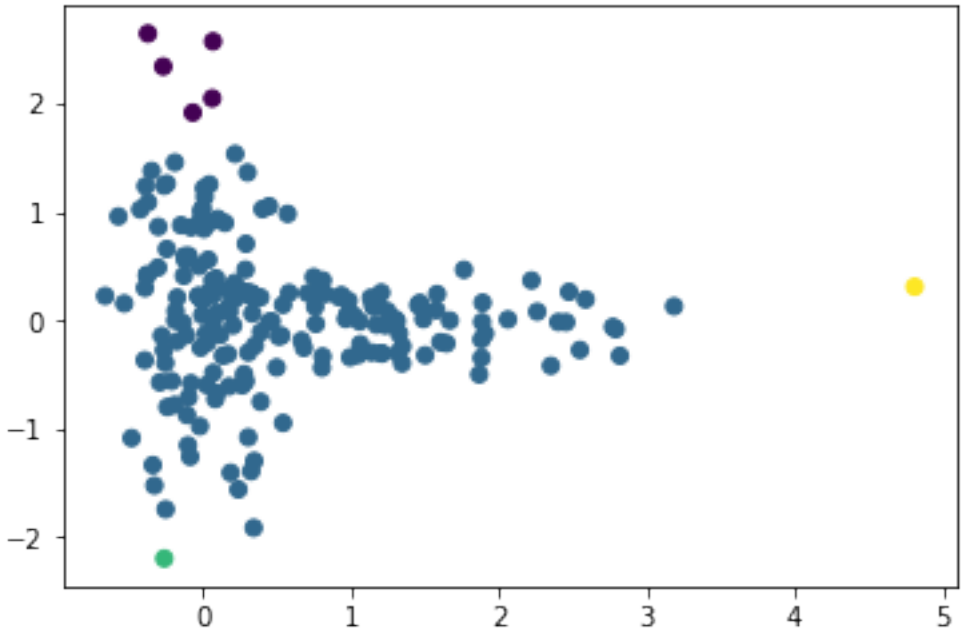
1.2.12 Flatten based on number of clusters

```
In [13]: clusters = hclust.fcluster(Z, 4, 'maxclust')
         len(set(clusters))
```

```
Out[13]: 4
```

```
In [14]: plt.scatter(X[:, 0], X[:, 1], c = clusters)
```

```
Out[14]: <matplotlib.collections.PathCollection at 0x11067ac50>
```

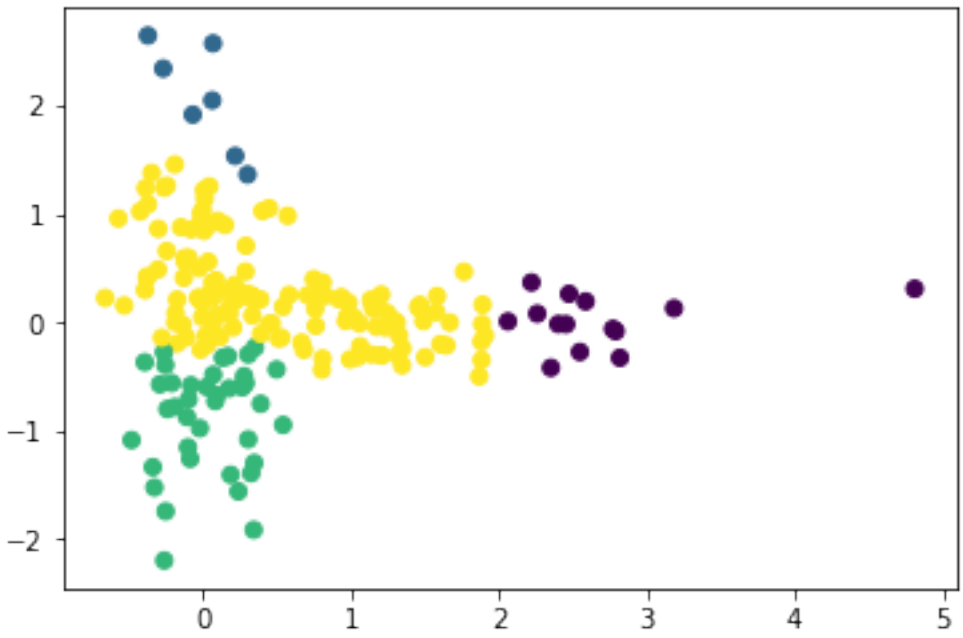


1.2.13 `fclusterdata`

`fclusterdata` does both linkage and fcluster in one step. Let's try out different linkage methods.

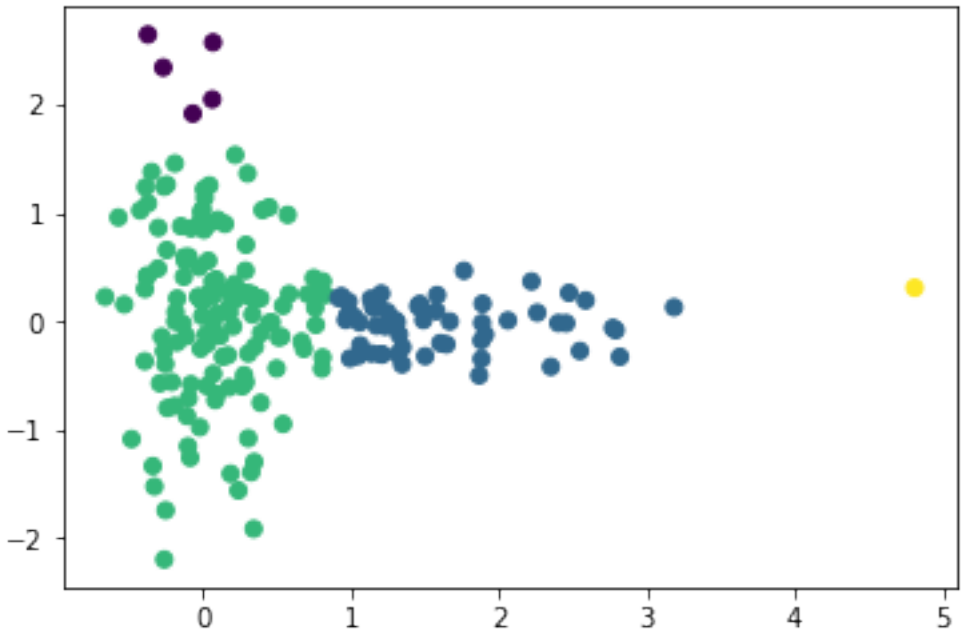
```
In [15]: clusters = hclust.fclusterdata(X, 4, 'maxclust', method = 'complete')
         plt.scatter(X[:, 0], X[:, 1], c = clusters)
```

```
Out[15]: <matplotlib.collections.PathCollection at 0x1a1700c5f8>
```




```
In [16]: clusters = hclust.fclusterdata(X, 4, 'maxclust', method = 'average')
         plt.scatter(X[:, 0], X[:, 1], c = clusters)
```

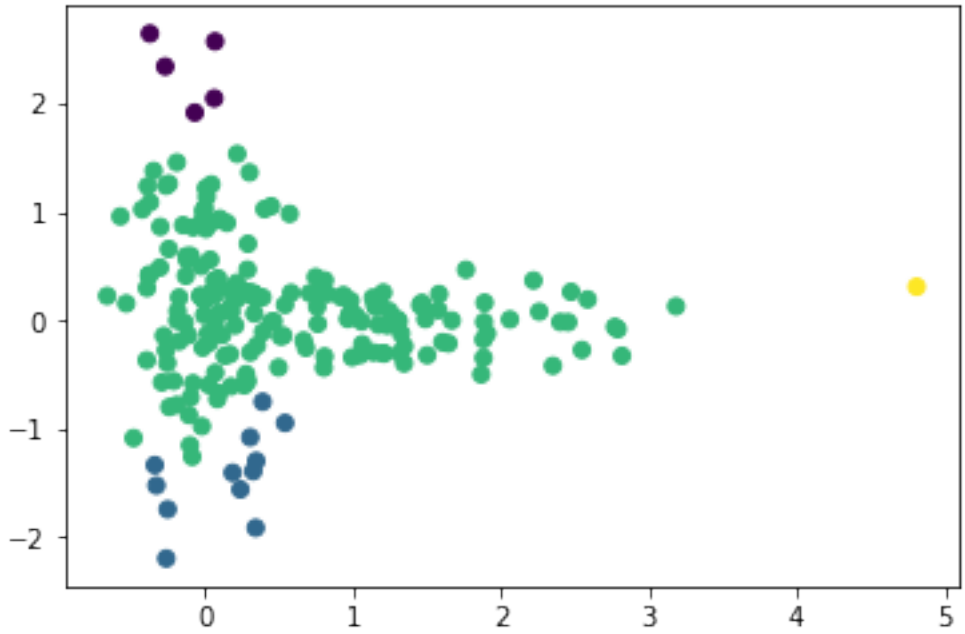
```
Out[16]: <matplotlib.collections.PathCollection at 0x11070fdd8>
```



You can even use a non-Euclidean metric.

```
In [17]: clusters = hclust.fclusterdata(X, 4, 'maxclust', method = 'average', metric = 'cityblo
         plt.scatter(X[:, 0], X[:, 1], c = clusters)
```

```
Out[17]: <matplotlib.collections.PathCollection at 0x1a17270630>
```



1.2.14 `leaves_list`

A hierarchical cluster imposes an order on the leaves. You can retrieve this ordering from the linkage matrix with `leaves_list`

```
In [18]: hclust.leaves_list(Z)
```

```
Out[18]: array([141,  29,  64,  67,  24,  83,  94, 113,   9, 173,   3,  14,  12,
                 26,  16,  84, 156, 166, 197, 150, 123, 142,  31,  15,  60,   0,
                 54,  75,  48,  89, 103, 163,  85,  55, 110, 151, 119, 138, 136,
                183, 177, 122, 153, 106, 159, 117, 170, 157,  40,  91, 194,  56,
                  5,  42, 165, 108, 116, 128, 184, 154, 198, 147, 148, 168,   8,
                  7,  71,  18,  88,  13,  82,   1,  45,  33,  87, 109,  69, 189,
                 46,  47,  49,  59,  90,  96,  66,  53,  97,  28,  70, 114, 144,
                100, 118, 196,  79,  68, 187,  50, 126, 127, 145, 175,   2, 125,
                 98,  72,  86, 160,   4,  51,  65,  78,  52,  80, 104, 129,  22,
                 58,  11, 178, 158,  39, 105, 140,  76, 199,  37,  93, 181, 185,
                190,  30,  73, 161, 143,  27,  38, 130,  34,  36,  57,  77, 191,
                162, 176, 132, 146, 101, 155, 169, 188, 107, 152, 195, 192, 131,
```

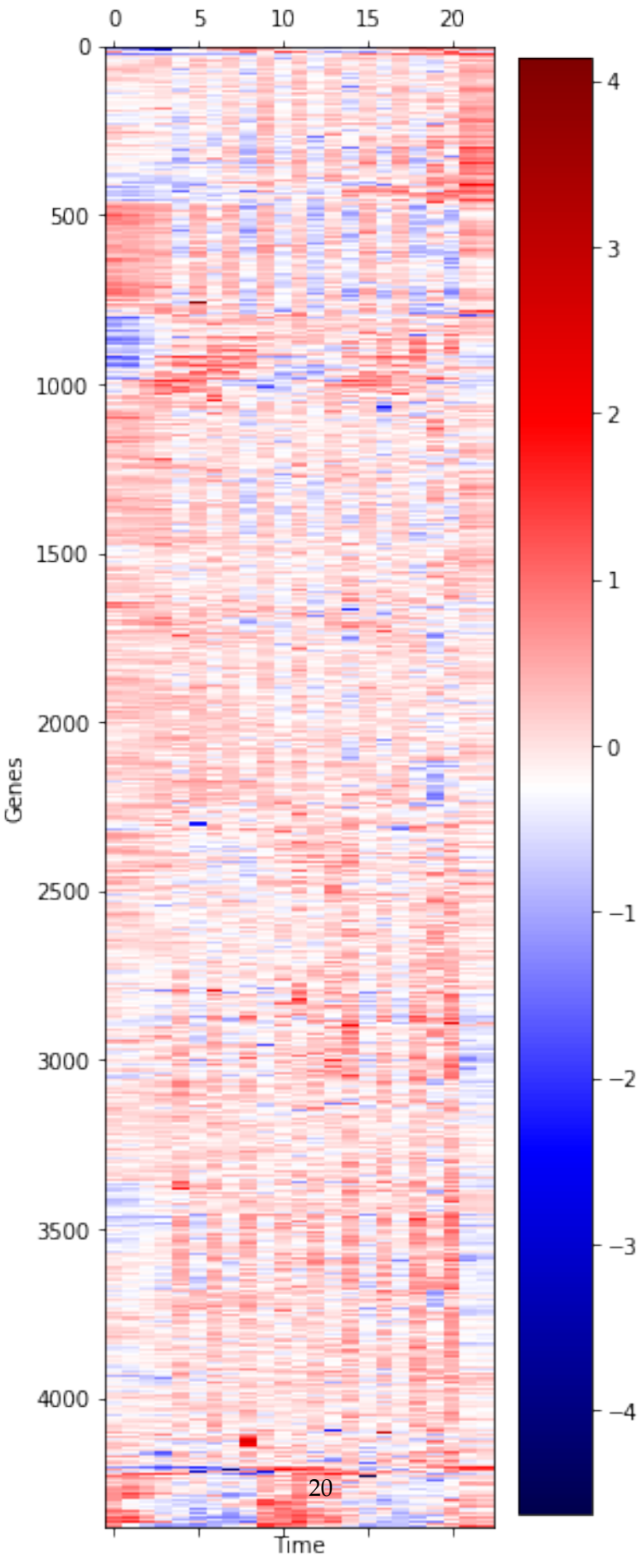
```
134, 115, 135, 171, 137, 111, 120, 121, 167, 112, 174, 172, 102,
164, 180, 139, 149, 179, 133, 182, 186, 124, 193,  21,  25,  10,
 95,  23,  62,  61,  35,  74,  19,  81,  43,  99,  41,  44,   6,
 20,  63,  92,  17,  32], dtype=int32)
```

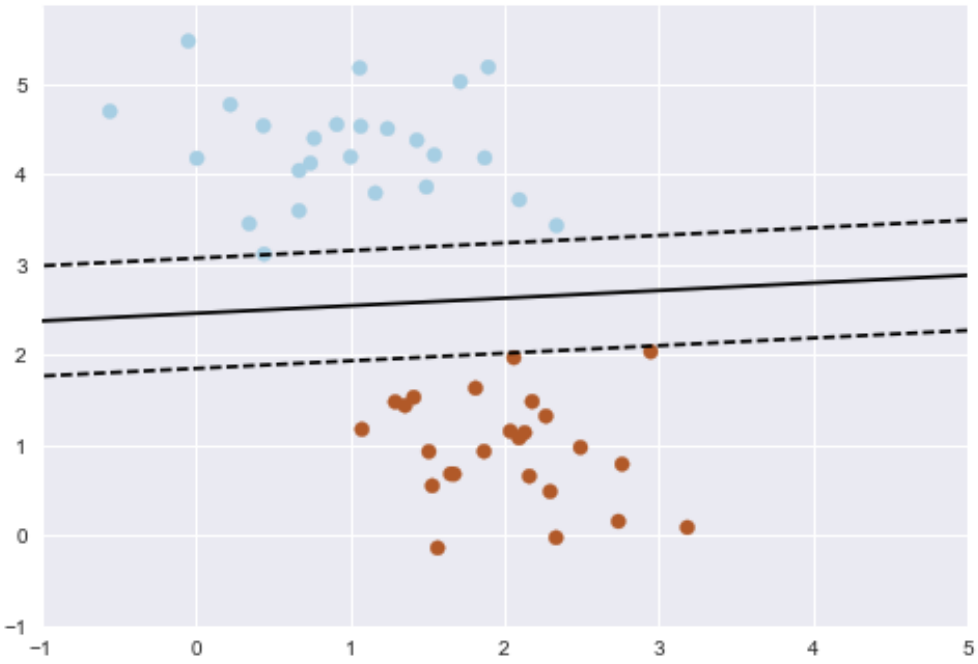
1.2.15 Clustering expression data

- Download https://eds-uga.github.io/cbio4835-fa18/ClusterPCAML/Spellman.csv
- read the expression data into a numpy array
- cluster it with the default parameters
- retrieve the leaves
- reorder the orginal data according the the leaf order
- display the result as a heatmap

In [19]:
```
from matplotlib.pylab import cm
data = np.genfromtxt('ClusterPCAML/Spellman.csv',skip_header=1,delimiter=',')[:,1:]
Z = hclust.linkage(data,method='complete')
leaves = hclust.leaves_list(Z)
ordered = data[leaves]
plt.matshow(ordered, aspect = 0.02, cmap=cm.seismic);
plt.xlabel("Time")
plt.ylabel("Genes")
plt.colorbar()
```

Out[19]:
```
<matplotlib.colorbar.Colorbar at 0x1a171b9c88>
```


0
5
10
15
20
0
500
1000
1500
2000
2500
3000
3500
4000
Genes
Time
4
3
2
1
0
−1
−2
−3
−4



decision-boundary

1.3 Part 2: Machine Learning

What is machine learning?

Wikipedia:

```
Machine learning is a subfield of computer science that evolved from the study of pattern recogn
```

Dr. David Koes, University of Pittsburgh:

```
Creating useful and/or predictive computational models from data.
```

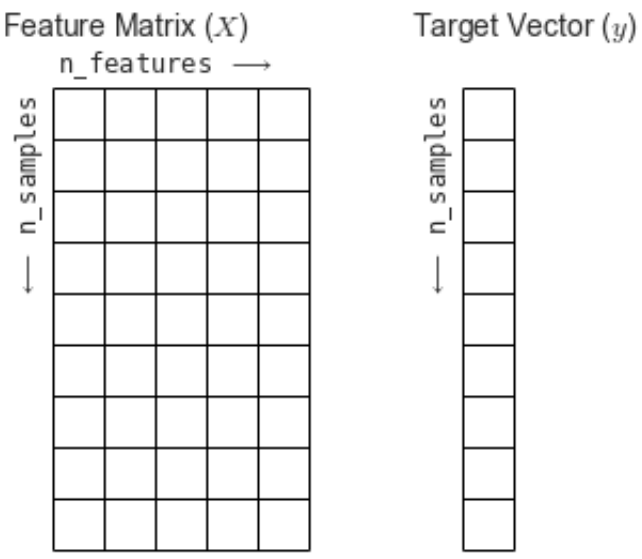
Machine Learning can be considered a subfield of **Artificial Intelligence**.

These algorithms can be seen as building blocks to make computers learn to behave more intelligently by somehow **generalizing** rather that just storing and retrieving data items like a database system would do.

Here's an overly simple example:

This may seem like a trivial task, but it is a simple version of a very important concept. By drawing this separating line, we have learned a model which can **generalize** to new data.

If you were to drop another point onto the plane which is unlabeled, this algorithm could now **predict** whether it's a blue or a red point.



data-layout

1.3.1 `scikit-learn`

Scikit-Learn is a Python package designed to give access to **well-known** machine learning algorithms within Python code, through a **clean, well-thought-out API**. It has been built by hundreds of contributors from around the world, and is used across industry and academia.

Scikit-Learn is built upon Python's NumPy (Numerical Python) and SciPy (Scientific Python) libraries, which enable efficient in-core numerical and scientific computation within Python.

1.3.2 Representing Data

Machine learning is about creating models from data, and `scikit-learn` uses a particular layout for building its models.

Most machine learning algorithms implemented in scikit-learn expect data to be stored in a **two-dimensional array or matrix**, typically as `numpy` arrays. The shape of the array is expected to be `[n_samples, n_features]`

1.3.3 Example: the Iris dataset

As an example of a simple dataset, we're going to take a look at the iris data (stored in `scikit-learn`, along with dozens of other sample datasets).

The data consists of measurements of three different species of irises.

There are three species of iris in the dataset, which we can picture here:

Iris Setosa

Iris Versicolor

Iris Virginica

The iris data consists of the following:

- Features in the Iris dataset:

setosa

versicolor

virginica

1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm

- Target classes to predict:

1. Iris Setosa
2. Iris Versicolour
3. Iris Virginica

Accessing the iris data in `scikit-learn` is pretty easy:

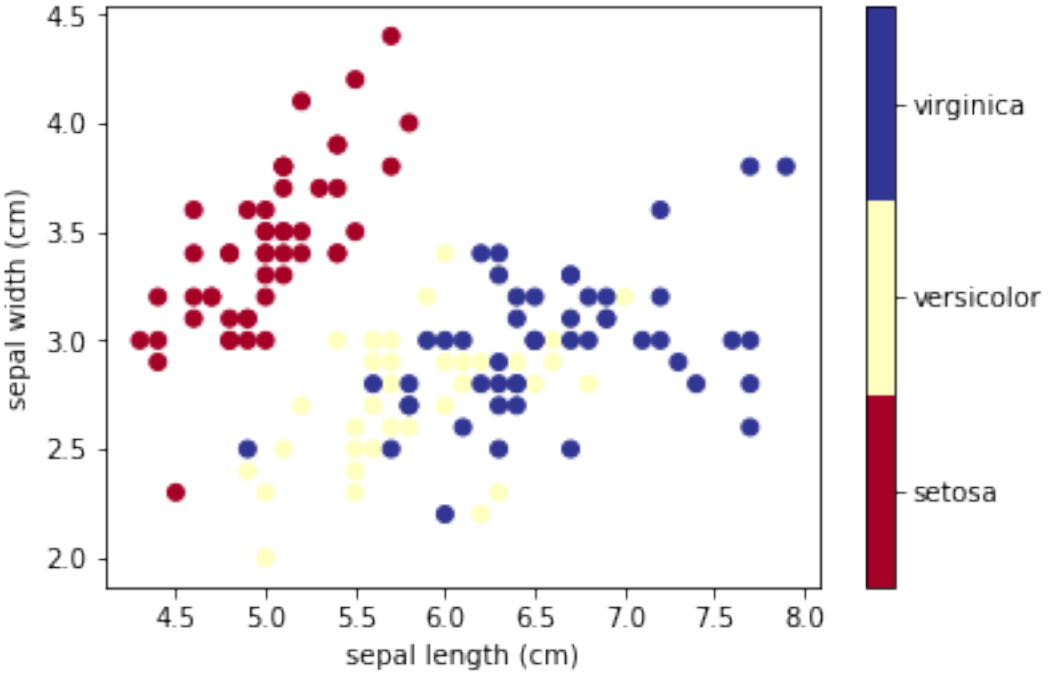
```
In [20]: from sklearn.datasets import load_iris
         iris = load_iris()  # "iris" is a DICTIONARY
```

We can examine this data more closely, using the properties of the various Python objects storing it:

```
In [21]: iris.keys()

Out[21]: dict_keys(['data', 'target', 'target_names', 'DESCR', 'feature_names', 'filename'])

In [22]: n_samples, n_features = iris["data"].shape
         print((n_samples, n_features))
         print(iris["data"][0]) # the 0th data point

(150, 4)
[5.1 3.5 1.4 0.2]


In [23]: print(iris["data"].shape)
         print(iris["target"].shape)

(150, 4)
(150,)


In [24]: print(iris["target"])

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 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 1 1 1 1 1 1 1 1 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]


In [25]: print(iris["target_names"])

['setosa' 'versicolor' 'virginica']
```

This data is four dimensional, but we can visualize two of the dimensions at a time using a simple scatter-plot:

```
In [26]: x_index = 0
         y_index = 1

         # this formatter will label the colorbar with the correct target names
         formatter = plt.FuncFormatter(lambda i, *args: iris.target_names[int(i)])

         plt.scatter(iris.data[:, x_index], iris.data[:, y_index],
                     c=iris.target, cmap=plt.cm.get_cmap('RdYlBu', 3))
         plt.colorbar(ticks=[0, 1, 2], format=formatter)
         plt.clim(-0.5, 2.5)
         plt.xlabel(iris.feature_names[x_index])
         plt.ylabel(iris.feature_names[y_index]);
```



1.3.4 Unsupervised Learning

Imagine we didn't have the target data (i.e., iris species) available, and wanted to *cluster* the data into similar groups.

In this case, we're discovering an underlying structure in the data **without any *a priori* knowledge of categories or "ground truth".** Hence, *unsupervised*.

Other forms of unsupervised learning include:

- dimensionality reduction, such as principal components analysis (PCA)

model

- expectation-maximization (k-means is a variant)
- self-organizing maps
- other clustering algorithms

1.3.5 Supervised learning

This is what we do in order to build models from data that **DO include some form of "ground truth" labels**.

The data consists of a set of examples X where each example has a corresponding label y.

Our assumption is that the label is a function of the data; our goal is to learn that function that maps X to y as accurately as possible: $y = f(X)$

1.3.6 Classification versus Regression

There are two types of supervised learning.

Classification maps an example to a discrete labels. This answers questions such as - Will it rain tomorrow? - Is the protein overexpressed? - Do the cells die when a drug is added?

Regression maps an example to a continuous number. This answers questions such as - How much rain will there be tomorrow? - What is the expression level of the protein? - What percent of the cells will die when the drug is added?

1.3.7 Flowchart for choosing your methods

This is a flow chart created by scikit-learn super-contributor Andreas Mueller which gives a nice summary of which algorithms to choose in various situations. Keep it around as a handy reference!

1.3.8 Back to iris

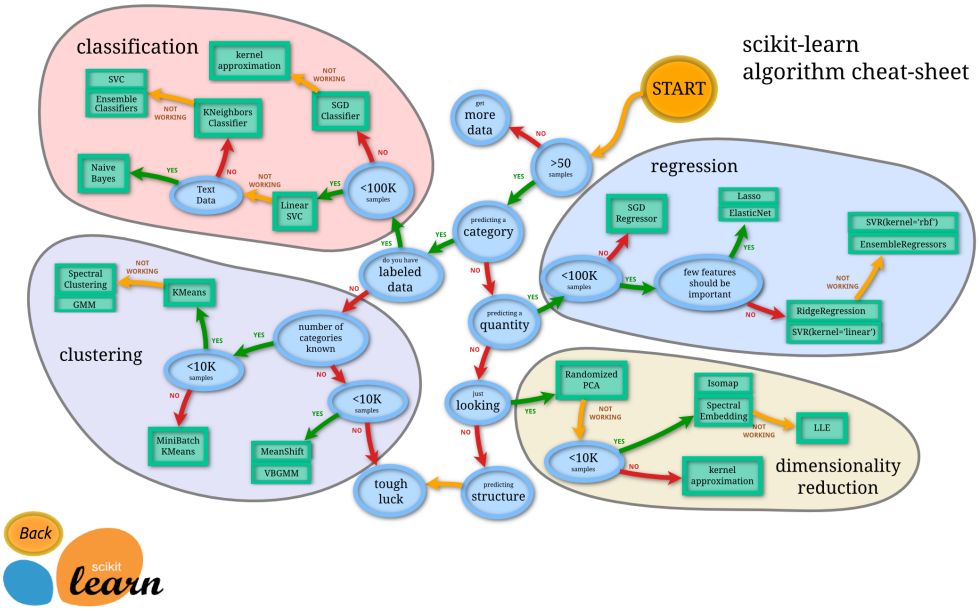
Given we have ground truth information (iris species), are we performing *supervised* or *unsupervised* learning on the iris data?

What if we didn't have grouth truth information? What kind of algorithms could we use?

What kind of supervised learning are we performing?

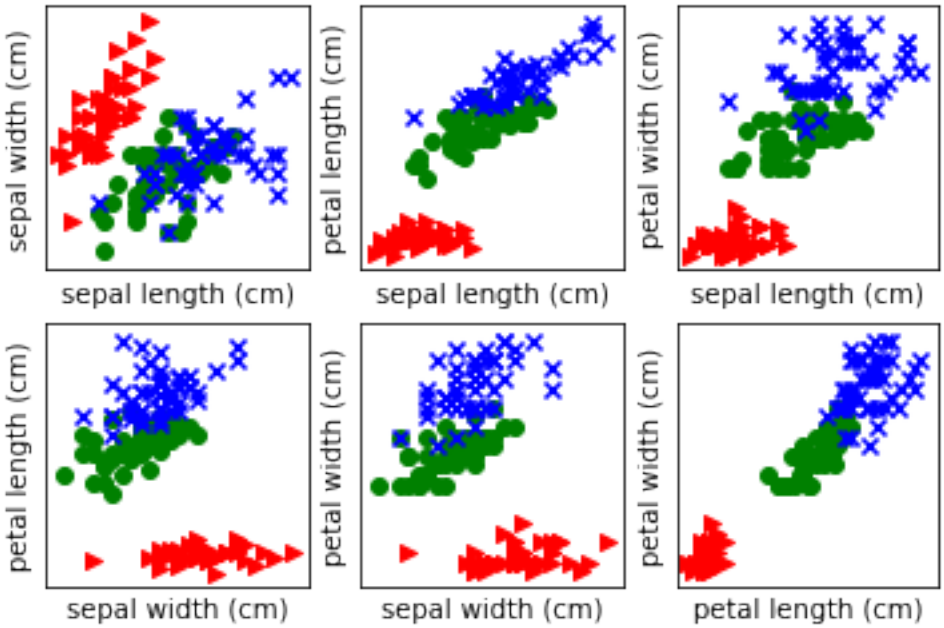
The iris data are 4-dimensional; can't exactly visualize that directly. We can, however, plot each pair of features in turn (this is called "getting a feel for the data").

```
In [27]: import itertools

         features = iris.data
         feature_names = iris.feature_names
         classes = iris.target
         feature_combos = itertools.combinations([0, 1, 2, 3], 2)
```



ml-map

```
for i, (x1, x2) in enumerate(feature_combos):
    fig = plt.subplot(2, 3, i + 1)
    fig.set_xticks([])
    fig.set_yticks([])
    for t, marker, c in zip(range(3), ">ox", "rgb"):
        plt.scatter(features[classes == t, x1],
                    features[classes == t, x2],
                    marker = marker, c = c)
    plt.xlabel(feature_names[x1])
    plt.ylabel(feature_names[x2])
```



1.3.9 Features

The features, X, are what make each example distinct. Ideally they contain enough information to predict y. In the case of the iris dataset, the features have to do with petal and sepal measurements.

The pairwise feature plots give us some good intuition for the data. For example:

- The first plot, sepal width vs sepal length, gives us a really good separation of the Setosa (red triangles) from the other two, but a poor separation between Versicolor and Virginica
- In fact, this is a common theme across most of the subplots–we can easily pick two dimensions and get a good separation of Setosa from the others, but separating Versicolor and Virginica may be more difficult
- The best pairings for separating Versicolor and Virginica may be either petal length vs sepal width, or petal width vs sepal width.

Still, for any given pair of features, we can't get a *perfect* classification rule.
But that's ok!

1.3.10 K-nearest neighbors (KNN)

Another algorithm with a k in the name–this time, a supervised algorithm–is the simplest classifier you can design.

It asks, for any given data point x: **What are the labels of the k data points closest to x?**

It then performs a majority vote, based on those labels. The winning label is assigned to the new data point x. The end!

Let's try it on the iris dataset.

```
In [28]: from sklearn import neighbors

         X = iris["data"][:, :2]  # For ease of interpretation
         y = iris["target"]

         # create the model
         knn = neighbors.KNeighborsClassifier(n_neighbors = 5)

         # fit the model
         knn.fit(X, y)
```

```
Out[28]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                    weights='uniform')
```

That's it. We've trained our classifier! Now let's try to predict a new, previously-unobserved iris:

```
In [29]: # What kind of iris has 3cm x 5cm sepal?
         # call the "predict" method:
         result = knn.predict([[3, 5],])

         print(iris.target_names[result])
```

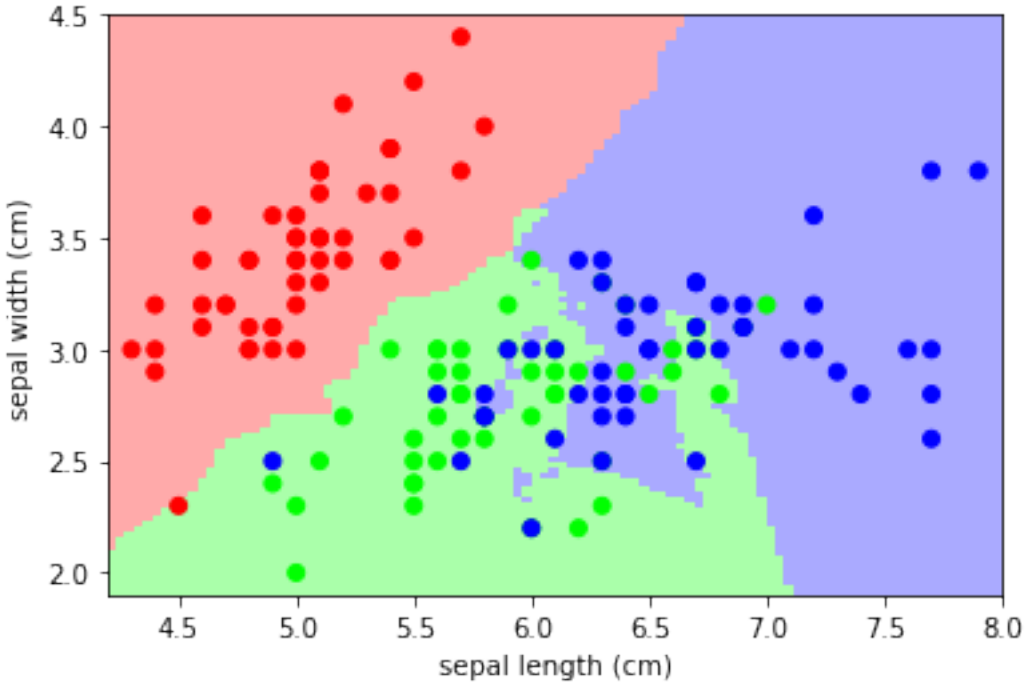
```
['setosa']
```

Our completely-made-up new mystery iris is classified as a setosa!
We can visualize what this algorithm looks like:

```
In [30]: from matplotlib.colors import ListedColormap
         cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
         cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

         ### IGNORE THIS ###
         x_min, x_max = X[:, 0].min() - .1, X[:, 0].max() + .1
         y_min, y_max = X[:, 1].min() - .1, X[:, 1].max() + .1
         xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                              np.linspace(y_min, y_max, 100))
         Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])

         # Put the result into a color plot
         Z = Z.reshape(xx.shape)
         plt.figure()
         plt.pcolormesh(xx, yy, Z, cmap = cmap_light)

         # Plot also the training points
         plt.scatter(X[:, 0], X[:, 1], c = y, cmap = cmap_bold)
         plt.xlabel('sepal length (cm)')
```

```
        plt.ylabel('sepal width (cm)')
        plt.axis('tight')
```

```
Out[30]: (4.2, 8.0, 1.9, 4.5)
```



There are plenty of more sophisticated classifiers: - support vector machines - neural networks - decision trees - ...

Furthermore, you will often combine multiple methods into a machine learning *pipeline*. For example, rather than work directly with the 4D iris data, perhaps you'll run PCA on it first to reduce it to 2 dimensions, then classify it.

Machine learning is more of an art than a science; it takes a lot of "playing with the data" to find the right combination of model and pipeline that incorporates the right assumptions about the data and can generalize well.

`scikit-learn` makes it relatively easy to get started: you don't have to know how the algorithms are implemented underneath, but that knowledge does help in fine-tuning your pipelines.

1.4 Administrivia

- How is Assignment 6 going? Due **Thursday!**
- How are projects going? Talks are **next week**, and write-ups are due **Thursday, December 6 by 11:59pm**.
- We'll go over material for the final exam-lab next week.
- Last lecture of the semester on Thursday!

1.5 Additional Resources

1. Richert, Willi and Pedro Coelho, Luis. *Building Machine Learning Systems with Python*. 2013. ISBN-13: 978-1782161400
2. ACM Machine Learning Workshop https://github.com/eds-uga/acm-ml-workshop-2017/ (based on Jake VanderPlas' scikit-learn tutorial)