Quantitative Applications for Data Analysis: unsupervised learning

Erik Spence

SciNet HPC Consortium

26 March 2024

SciNet compute • calcul CANADA

Today's slides

Today's slides can be found here. Go to the "Quantitative Applications for Data Analysis" page, under Lectures, "Unsupervised learning".

`https://scinet.courses/1346`

SciNet compute • calcul CANADA

Today's class

Today we're going to explore some unsupervised learning algorithms:

- Factor Analysis,
- Principle Component Analysis,
- Clustering algorithms.

These are algorithms that don't require the target (label). As a result, they are "unsupervised", they have nothing to guide them.

Ask questions!

SciNet compute • calcul CANADA

Curse of dimensionality

The "curse of dimensionality" is a generic claim which refers to the difficulty in properly fitting or modelling data in high-dimensional spaces.

- Each feature in your data set is another dimension.
- Each dimension gives more space for your solution to live in.
- The more space there is, the harder it can be to find the solution, or build a meaningful model.
- "Dimensionality reduction", or "feature selection" is the act of either
  - ignoring data which is obviously not important, or
  - modifying the data to put it into a form which has fewer dimensions.

We will examine Factor Analysis and Principle Component Analysis (PCA), which are both dimensionality-reduction techniques.

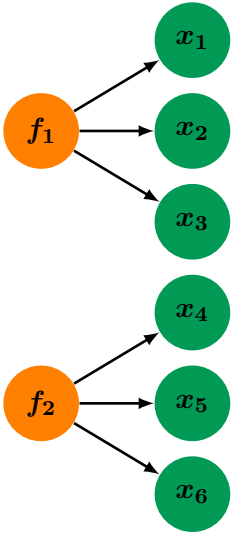
SciNet compute • calcul CANADA

Factor analysis

Imagine you've got a bunch of data, each with 6 features, x_1, x_2, ..., x_6. You've observed that there are correlations between some of the features.

It's possible that there might be some underlying, unobserved, 'factors', say f_1 and f_2, which are responsible for certain features, which is why some features are correlated to each other.

It would be useful to represent the data through these factors (also called "latent variables"), rather than the original features, since there are presumably fewer of them, and they are the actual cause of the feature's values.

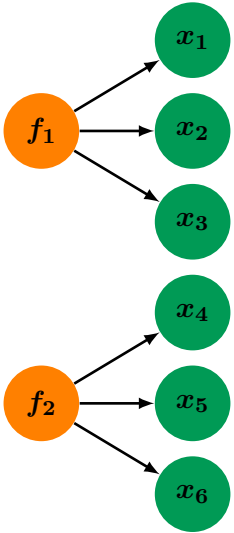


Factor analysis, continued

The goal of factor analysis (sometimes called "exploratory factor analysis") is to determine linear relationships between the factors and the features. Assuming there are only 2 factors, these relationships would take the form

$$x_1 = \beta_{10} + \beta_{11} f_1 + \beta_{12} f_2 + \epsilon_1$$
$$x_2 = \beta_{20} + \beta_{21} f_1 + \beta_{22} f_2 + \epsilon_2$$
$$\vdots$$
$$x_6 = \beta_{60} + \beta_{61} f_1 + \beta_{62} f_2 + \epsilon_6$$

Where the ϵ terms represent noise, and the β factors are known as "loadings".



Factor analysis, continued more

To calculate the relationships on the previous slide, we will make a few assumptions:

- The noise terms, ϵ, are independent, and have a mean of zero.
- The unobservable factors are independent of each other, have a mean of zero and a variance of 1 (the factors have been 'standardized').

The calculation of the loadings turns out to just be a whole lot of algebra, and correlations. We won't delve into the derivation details here.

We generally don't stop there. The calculation can be refined.

- The calculation of the factors is not unique (there are many combinations of loadings which will give the same answer).
- As such we can "rotate" the answer such that some of the loadings are large, and others are small. This makes the interpretation of the factors easier.

SciNet compute • calcul CANADA

Factor analysis, example

sklearn includes a FactorAnalysis model, but it's missing important functionality. I use the factor_analyzer package instead.

Let us do an example. We first confirm that the features have enough correlation. There are tests available to see if factor analysis is appropriate:

- Kaiser-Meyer-Olkin,
- Bartlett's test.

A KMO test value >0.6 is required.

```
In [1]: import factor_analyzer as fa
In [2]: import sklearn.datasets as skd
In [3]:
In [3]: data = skd.load_breast_cancer()
In [4]:
In [4]: fa.calculate_kmo(data.data)
Out[4]:
(array([0.834635, 0.643526, 0.853340, 0.864032, 0.814716,
0.879397, 0.891928, 0.900277, 0.825102, 0.831813,
0.834121, 0.484589, 0.842907, 0.851998, 0.644278,
0.871297, 0.825479, 0.835166, 0.583218, 0.811496,
0.823087, 0.603297, 0.884937, 0.820445, 0.753160,
0.851277, 0.902225, 0.891052, 0.690747, 0.812339]),
0.8322253094685496)
In [5]:
```

Factor analysis, example, continued

The tenth data point is most strongly depenent on the fourth factor.

Rotation of the data is done automatically.

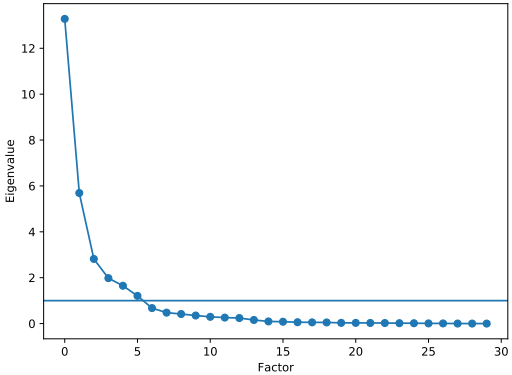
```
In [5]:
In [5]: model = fa.FactorAnalyzer(n_factors = 8)
In [6]:
In [6]: new_x = model.fit_transform(data.data)
In [7]:
In [7]: data.data.shape, new_x.shape
Out[7]: ((569, 30), (569, 8))
In [8]:
In [8]: new_x[10, :]
Out[8]:
array([ 0.16629477, -0.7870072 , -0.68653714, 1.55710662,
-0.98180443, -0.2216149 , 0.01244827, 0.06496257])
In [9]:
```

Factor analysis, example, continued

We plot the eigenvalues which result from the analysis to determine how many factors we actually need to keep.

Anything less than 1 is not important, so we should keep the first six factors.

```
In [9]:
In [9]: import matplotlib.pyplot as plt
In [10]:
In [10]: plt.plot(model.get_eigenvalues()[0], 'o-')
In [11]: plt.axhline(1)
In [12]:
```



Factor analysis, final notes

By representing the data using factors, we can now reduce the number of features. We then feed this representation of the data into our favourite machine learning algorithm.

Some things to be aware of:

- Generally, before performing factor analysis, you should perform an "adequacy test", which determines if the data can be factored:
  - Bartlett's test,
  - Kaiser-Meyer-Olkin test.
- There are different rotation algorithms available, which may affect the final results.

Note that factor analysis is sometimes controversial, since the final result is not unique, and thus the interpretation of the factors is subjective.

SciNet compute • calcul CANADA

Principle component analysis

Principle Component Analysis (PCA), seems similar to Factor Analysis on the surface, but it is different in important ways. Like Factor Analysis, it is a technique that allows dimensionality reduction in problems with purely continuous features.

It ignores the data targets; it merely imagines the data as points in a p-dimensional space.

- PCA performs a spatial transformation to the data space,
- the transformation rotates and scales the data space into directions defined by the variance in the data.
- Singular Value Decomposition is used to perform these steps.

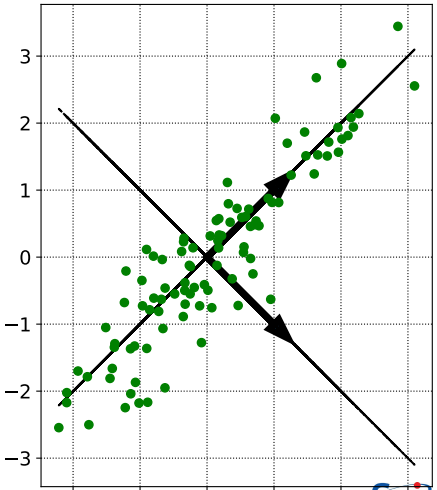
In essence (in linear-algebra-speak), PCA simply projects the data onto a new basis set, where the important features are clearer.

SciNet compute • calcul CANADA

PCA, continued

Principle component analysis picks out the directions the data takes that contain the most variance.

- The direction in which the variance is highest is rotated to point along the first axes (the first principal component). Next along the second axis, *etc.*
- Increasingly higher dimensions are flatter and flatter, as they have less variance.
- The dimensions which have very little variance contain very little information, and can be discarded.



PCA, example

We can perform PCA on the wine data set.

Unsurprisingly, PCA is built into sklearn. Fitting the data takes a single line of code.

Once built, the PCA object contains a tonne of information about the fit, including the principle components themselves.

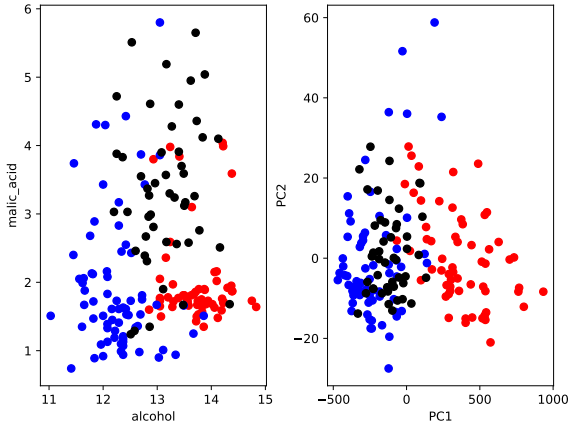
```
In [12]:
In [12]: import matplotlib.pyplot as plt
In [13]: import sklearn.decomposition as skde
In [14]:
In [14]: wine = skd.load_wine()
In [15]: x = wine.data
In [16]: y = wine.target
In [17]:
In [17]: pca = skde.PCA()
In [18]: pca = pca.fit(x)
In [19]:
In [19]: pca.n_components_
Out[19]: 13
In [20]:
```

PCA, example, continued

```
In [20]:
In [20]: x_new = pca.transform(x)
In [21]:
In [21]: plt.subplot(1, 2, 1)
In [22]: plt.scatter(x[:, 0], x[:, 1], c = y)
In [23]: plt.subplot(1,2,2)
In [24]: plt.scatter(x_new[:, 0], x_new[:, 1],
    ...:                 c = y)
In [25]:
```

Use the "transform" command to project the data onto the new basis set.

Note how the clusters of the data are more-easily distinguished.



PCA, continued more

Of what use is it?

- Once you have projected the data onto its principle components, you can determine which components are most important.
- Those dimensions which are least important can be discarded, resulting in a dimensionality reduction.
- Note that PCA will automatically center the data. No pre-centreing and scaling needed, actually it would interfere with things.
- Once projected onto the new space, clustering is sometimes a useful next step. PCA can sometimes separate clusters that are otherwise difficult to detect.

But how do we decide which components to keep? What is a good criteria? If we don't throw away any dimensions we're no better off than we were before.

SciNet compute • calcul CANADA

Explained variance ratio

The quality of the PCA, and which components are worth keeping, is indicated in the "explained_variance_ratio_" variable.

- This indicates that 99.8% of the variance in the data set lies along the first principle component.
- 0.2% along the second, etc.
- We should decide ahead of time how many components are needed to explain, say, 95% of the variance.

```
In [25]:
In [25]: pca.explained_variance_ratio_[0:4]
Out[25]: array([9.98091230e-01, 1.73591562e-03,
                9.49589576e-05, 5.02173562e-05])
In [26]:
In [26]: import numpy as np
In [27]:
In [27]: np.cumsum(pca.explained_variance_ratio_[0:4])
Out[27]: array([0.99809123, 0.99982715,
                0.99992211, 0.99997232])
In [28]:
```

99.98% of the variance is explained by the first two principle components.

Explained variance ratio, continued

It's easier to just specify ahead of time that we want to only keep the principle components which explain at least 95% of the variance.

Specifying the "n_components" flag will cause the function to cut the number of components for you.

In this example we specify 99.9% so that there is more than a single component. This is an unnecessarily-high threshold for real cases.

```
In [28]:
In [28]: pca2 = skde.PCA(n_components = 0.999)
In [29]:
In [29]: pca2 = pca2.fit(x)
In [30]:
In [30]: pca2.n_components_
Out[30]: 2
In [31]:
In [31]: x_new2 = pca2.transform(x)
In [32]:
In [32]: x.shape
Out[32]: (178, 13)
In [33]: x_new2.shape
Out[33]: (178, 2)
In [34]:
```

PCA, summary

Some further notes about PCA:

- PCA doesn't drop features; rather, it generates combinations of all features in order of how significantly they vary.
- One generally keeps most of the information from all features, but expressed in a number of combinations $k < p$.
- However, especially in situations with a large number of dimensions, the least significant principal components can often be profitably ignored, as there is very little variation in those directions.
- Once projected onto the new space, clustering is sometimes a useful next step. PCA can sometimes separate clusters that are otherwise difficult to detect.

Note there are other types of dimensionality reduction as well: Locally Linear Embedding (LLE), Linear Discriminant Analysis (LDA), and others.

SciNet compute • calcul CANADA

Clustering

Let's switch to a different sort of classification approach: clustering.

- This is a type unsupervised learning.
- It's unsupervised because there are no targets (labels) used.

This can show up in all sorts of applications:

- Finding patterns in properties of galaxies.
- Determine proteins with similar interaction types.
- Market segmentation.
- "Customers who buy X often buy...".

There are two main clustering approaches you'll run into: k-means and agglomerative (hierarchical) clustering.

SciNet compute • calcul CANADA

Clustering, continued

The reason for using algorithms to find clusters in the data is because

- It's difficult to find clusters in high-dimensional data (since you can't visualize it all at once).
- You might want to summarize a large number of observations into fewer, similar clusters.

Obviously, we haven't defined what we mean by "similar" or "cluster" yet.

- A "cluster" is a group of data points which are centered around some central, average point.
- The "similarity" between points is determined by some measure of "distance" between them, in the p dimensional space in which they live.
- In continuous spaces the distance can be Euclidean, or some other measure of distance (L1 norm).
- In ordinal spaces (bag-of-words counts, for example) you can use the "cosine similarity"

$$\cos\theta = \frac{\mathbf{A}\cdot\mathbf{B}}{|\mathbf{A}||\mathbf{B}|}$$

K-means clustering

K-means clustering is a geometric clustering algorithm which finds roughly-spherical blobs of clusters amongst the data. The algorithm is straightforward. Starting with k initial cluster centres:

- Assign each data point to the nearest centre.
- Recalculate the center of each cluster, based on its members.
- Move the centres to the new locations.
- Repeat until converged (the centres stop moving).

The value of k must be specified before starting.

SciNet compute • calcul CANADA

K-means clustering, example

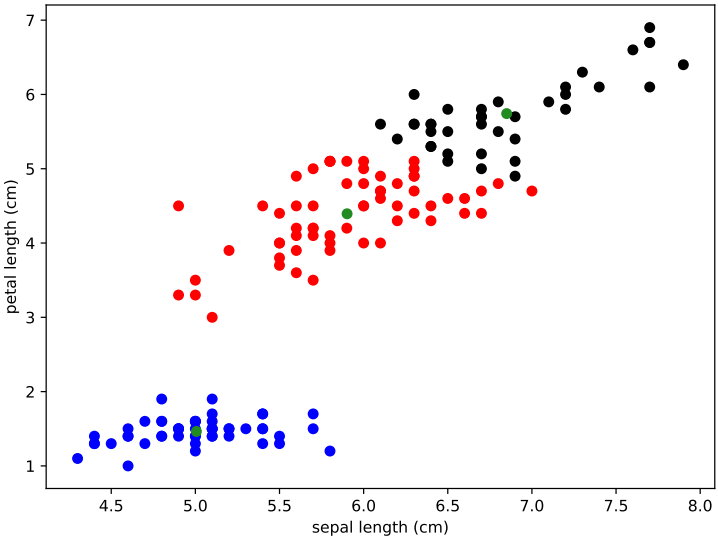
As you might expect, K-means is built into sklearn.

It's as easy to use as you might hope.

Once the model is trained, you can get the centers of the clusters, and the predicted labels, using the "cluster_centers_" and "labels_" model entries.

```
In [34]: import sklearn.cluster as skc
In [35]:
In [35]: data = skd.load_iris()
In [36]: x = data.data
In [37]: y = data.target
In [38]:
In [38]: model = skc.KMeans(n_clusters = 3)
In [39]: model = model.fit(x)
In [40]:
In [40]: plt.scatter(x[:,0], x[:,2], c = model.labels_)
In [41]:
In [41]: for i in range(3):
...:         plt.scatter(model.cluster_centers_[i][0],
...:                     model.cluster_centers_[i][2],
...:                     c = 'ForestGreen')
In [42]:
```

K-means clustering, example, continued



K-means clustering, continued

K-means has both strengths and weaknesses.

- You need to know what value of k to use.
- Random initialization of the centres can go badly wrong.
- For this to be robust, you need to repeat many times.
- This is usually done automatically by sklearn's KMeans, and the best result is returned.
- k-means has a tendency to make equally-populated clusters, which can lead to incorrect results.

For this to work consistently, we need a way to measure the quality of the model.

SciNet compute • calcul CANADA

K-means clustering, quality measures

A few measures of error have been developed for K-means.

- We'd like to minimize the within-cluster sum of squares, where $\boldsymbol{\mu_i}$ is the centre of the $\boldsymbol{i}$th cluster.

$$\mathbf{WCSS} = \sum_{\boldsymbol{i}}^{\boldsymbol{k}} \sum_{\boldsymbol{j} \in \boldsymbol{S_i}} |\boldsymbol{x_j} - \boldsymbol{\mu_i}|^{\mathbf{2}}$$

- We'd like to maximize the between-cluster sum of squares.

$$\mathbf{ICSS} = \sum_{\boldsymbol{i}}^{\boldsymbol{n}} \sum_{\boldsymbol{j}}^{\boldsymbol{n}} \boldsymbol{\delta}(\boldsymbol{S_i}, \boldsymbol{S_j}) \, |\boldsymbol{x_i} - \boldsymbol{x_j}|^{\mathbf{2}}$$

These are output by standard $\boldsymbol{k}$-means algorithms.

K-means and cross-validation

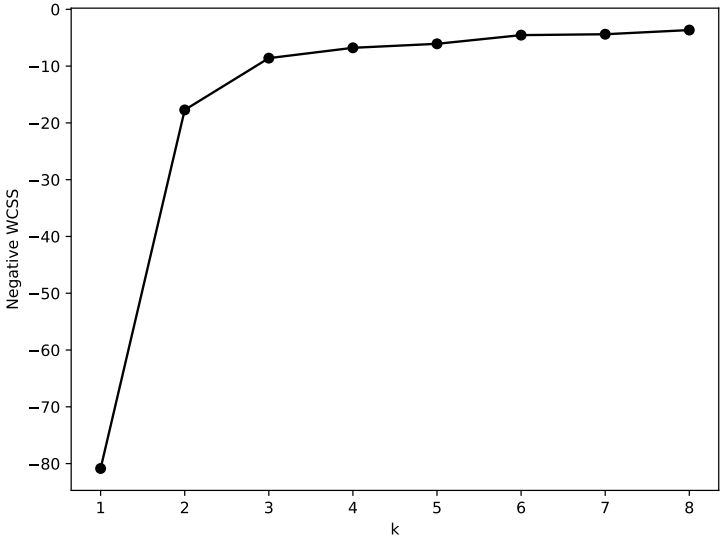
How do we pick k? You guessed it!

- Create a KMeans object.
- Use the cross_val_score function to perform the cross-validation for you.
- The function returns the scores for each k fold.
- Examine the scores to find the best k value.

```
In [42]: import sklearn.model_selection as skms
In [43]:
In [43]: kvalues = range(1, 9)
In [44]: scores = np.zeros(len(kvalues))
In [45]:
In [45]: for i, k in enumerate(kvalues):
...:         model = skc.KMeans(n_clusters = k)
...:         scores[i] = np.mean(skms.cross_val_score(model,
                                 x, cv = 10))
In [46]:
In [46]: plt.plot(kvalues, scores, 'ko-')
In [47]:
```

Unlike other algorithms, the accuracy of k-means does not 'turn over', meaning start to get worse with increasing k.

K-means and cross-validation, continued



Agglomerative clustering

K-means uses a geometric approach to clustering. Agglomerative (hierarchical) clustering works point-by-point:

- All data points start in their own cluster.
- At each iteration, the two "best matching" are joined into the same cluster.
- Repeat until there is only one cluster left.

This builds a tree of connections. This tree then needs to be pruned to distinguish the clusters. To do this we still need some sort of distance metric, and a linkage criteria, which specifies the dissimilarity of the clusters.

- k-means-like: what is the distance between the centres of the clusters which have been built thus far?
- single linkage: what is the minimum distance between any two points in two clusters?
- mean linkage: what is the mean distance between all points in two clusters?

Agglomerative clustering, example

As you might expect, agglomerative (hierarchical) clustering is built into sklearn.

Let's use a different data set to test this: swiss roll.

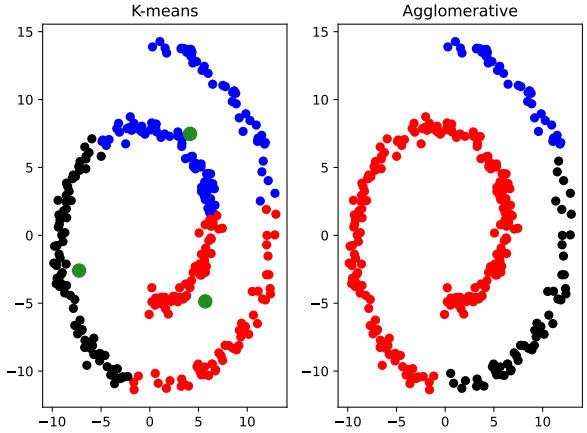
The connectivity of the points is determined using the kneighbors_graph command. This creates a graph between the points, based on some metric.

```
In [47]: import sklearn.neighbors as skn
In [48]: x, y = skd.make_swiss_roll(300, noise = 0.4)
In [49]: x.shape
Out[49]: (300, 3)
In [50]:
In [50]: x = np.c_[x[:, 0], x[:, 2]]
In [51]:
In [51]: x.shape
Out[51]: (300, 2)
In [52]:
In [52]: model = skc.AgglomerativeClustering(n_clusters = 3,
             connectivity = skn.kneighbors_graph(x, 30))
In [53]: model = model.fit(x)
In [54]:
In [54]: plt.scatter(x[:,0], x[:,1], c = model.labels_)
In [55]:
```

K-Means versus agglomerative clustering

K-means and agglomerative clustering have very different behaviours.

- K-means only cares about distances "as the crow flies".
- Agglomerative cares about distances between individual data points.
- K-means requires the number of clusters up front.
- Agglomerative gives you an entire tree of relations.



SciNet compute • calcul CANADA

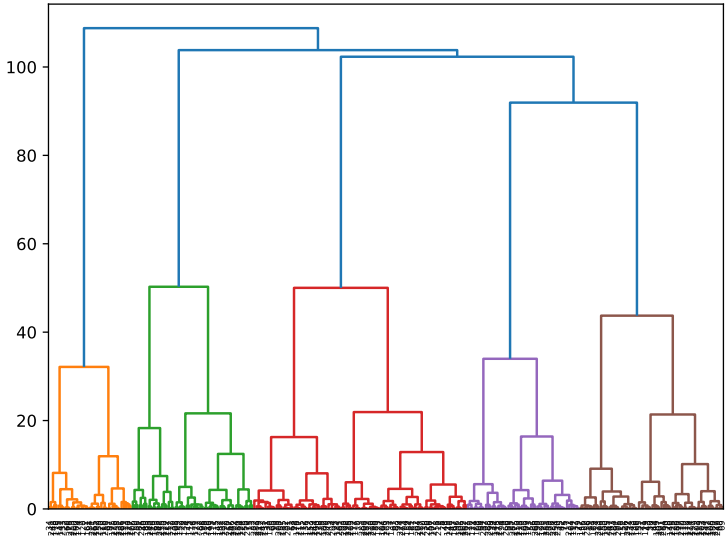
Agglomerative clustering: dendrogram code

```
# dendro_linkage.py
import numpy as np

def build_link_array(model):
  counts = np.zeros(model.children_.shape[0])
  n = len(model.labels_)
  for i, merge in enumerate(model.children_):
    node_count = 0
    for child_idx in merge:
      if child_idx < n: node_count += 1 # leaf node
      else: node_count += counts[child_idx - n]
    counts[i] = node_count
  linkage_array = np.column_stack([model.children_,
        model.distances_, counts]).astype(float)

  return linkage_array
```

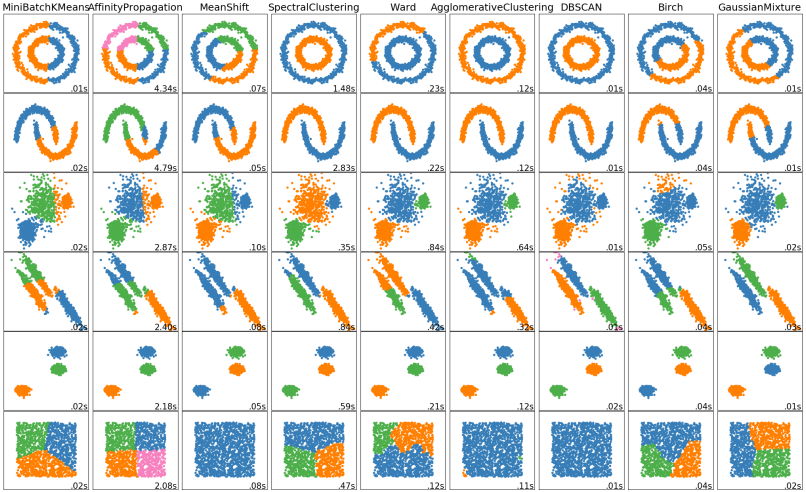
```
In [55]: import scipy.cluster.hierarchy as sch
In [56]: import dendro_linkage as dl
In [57]:
In [57]: link_array = dl.build_link_array(model)
In [58]:
In [58]: sch.dendrogram(link_array)
In [59]: plt.show()
In [60]:
```

The scipy.cluster.hierarchy has the code to build the linkage array and make the dendrogram plot, but does not allow you to use an arbitrary linkage function, as we did on the previous slide, so we need to code it ourselves.

Agglomerative clustering: dendrogram



Scikit-learn clustering algorithms



http://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_comparison.html

Summary

Some things to remember:

- Factor Analysis, PCA and clustering require numeric data.
- Factor Analysis recasts the data as linear combinations of factors.
- PCA projects the data onto a new basis set, based on the variance in the data.
- PCA accounts for as much variance in the data as possible. Factor analysis accounts for as much covariance in the data as possible.
- By discarding the dimensions with minimal variance the dimensionality of the problem can be reduced.
- PCA results are also often used as the input to clustering algorithms, since the major sources of variance have already been isolated.
- Clustering algorithms group data into 'clusters' of common attributes.
- K-means find clusters by finding the centres of clusters of data. K-means requires k to be specified.

SciNet compute • calcul CANADA