Unsupervised clustering:

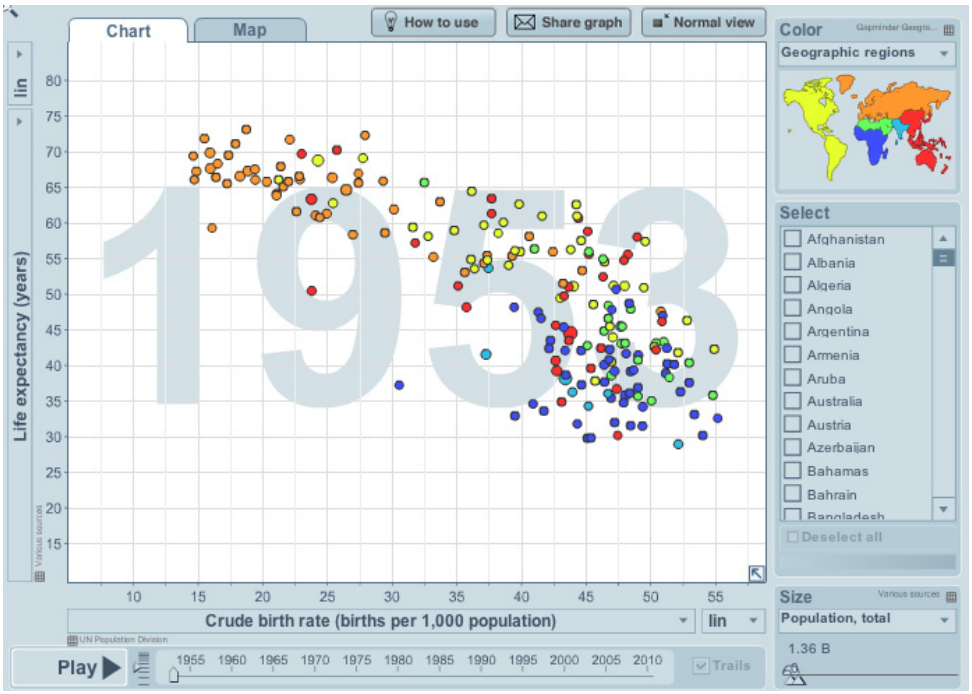
What is machine learning?:

Machine learning is all about learning structure in data. Learning is vital in many types of AI and even in fields outside of this. Machine learning algorithms can be used in pathfinding robots to allow them to make predictions about the world and decisions on how to act.

Clustering as a machine learning algorithm:

"Unsupervised clustering" is a method for interpreting the structure of a dataset. Clustering is a machine learning algorithm used in everything from Computer Vision (trying to find out which pixels in an image are part of a certain object in the picture - ie a cow on a landscape) to Bioinformatics (Trying to find the shortest distance between some closely related species to attempt to reconstruct the historical evolution of the animals).

Imagine we had a dataset that was made up of a large number of data points. When performing unsupervised clustering, we seek to assign each point of data to a cluster, learning where the clusters are, and which data points belong to which clusters. As a concrete example, imagine we had measured two variables for each of a large number of countries, like this (data from http://www.gapminder.org/world):



The two variables we have measured are Birth Rate (the number of children born per 1000 people each year), and Life Expectancy (how long someone lives on average after being born). Measuring these two variables let each country be placed on a two dimensional plane, creating a graph known as a "scatter plot" which you may have seen before. Further, imagine you were working for the United Nations, and that the UN wants to give different aid packages to countries with different birth rates and life expectancies. You have been asked to group these countries into 3 groups (or "clusters"), where countries with similar Life Expectancies and Birth Rates are grouped together. You could attempt to draw lines on the scatter plot, separating the clusters by eyeball, but this kind of manual clustering is not very scientific or fair, and becomes much harder when you have a larger number of variables, so you cannot simply plot the data.

The k-means clustering algorithm:

So how can we automate this kind of task? We can use a popular machine learning algorithm: The "k-means" clustering algorithm (see http://en.wikipedia.org/wiki/K-means_clustering). "k" in "k-means" stands for the number of clusters you wish to find. In our example, k is equal to 3, as requested by the UN. "Mean" just refers to a simple average.

The key assumptions behind the k-means algorithm:

1) The center of each cluster is the mean of all the data points that belong to it (hence the name "k-means").

2) Each data point belongs to the cluster with the nearest center point.

These two assumptions are actually sufficient to describe the entire algorithm. All the k-means algorithm does is iterate two steps, each trying to satisfy one of these conditions!

Before spelling out the entire algorithm, we first need to discuss the notion of "distance". In machine learning, there are a number of different distance metrics that can be used for these kinds of algorithms. Simple examples you might not have heard of are "Manhattan distance" (http://en.wikipedia.org/wiki/Taxicab_geometry) and "Chebyshev distance" (http://en.wikipedia.org/wiki/Chebyshev_distance), but the one we will be using here will be the familiar "Euclidean distance" (http://en.wikipedia.org/wiki/Euclidean_distance) you have probably seen in high school mathematics. It is generally important to bear in mind that the notion of "distance" extends beyond just Euclidean distance, even though we will stick to Euclidean distance for this exercise. In two dimensions, the Euclidean distance between the points $[x_i,y_i]$ and $[x_j,y_j]$ just $\sqrt{[x_j-x_i]^2 + [y_j-y_i]^2}$. In Python, each data point will be stored as a list with two elements (x,y), so you should make a function that takes two data points and returns the distance between them.

To compute the mean (or average) of a number of observations, you simply add all the observations together, and then divide by the number of observations you added together. The two dimensional mean is no different. You have a number of x and y values, so to compute the mean $[x,y]$ point of a number of points, you simply compute the mean of all the x values, and the mean of all the y values.

Specification of k-means:

We are now ready to outline the algorithm:
0) Initialize a mean for each cluster, by randomly picking points from the data and using these points as the starting values for the means. You can use the "sample()" function in python (after importing "random") to do this.
Repeat the code in brackets a pre-specified number of times:
{
1) Assign each point to the nearest cluster
2) Compute the means for each cluster as the mean for all the points that belong to it
}

It really is that simple. It might not be obvious, but iterating steps 1 and 2 causes the means of each cluster to rapidly converge upon a stable set of values, which correspond to something called a "local optimum". To converge means that the means get to a certain value and continuing to run the algorithm doesn't make this value change, ie the value has 'stabilised' - we'll go into more detail on this a little later.

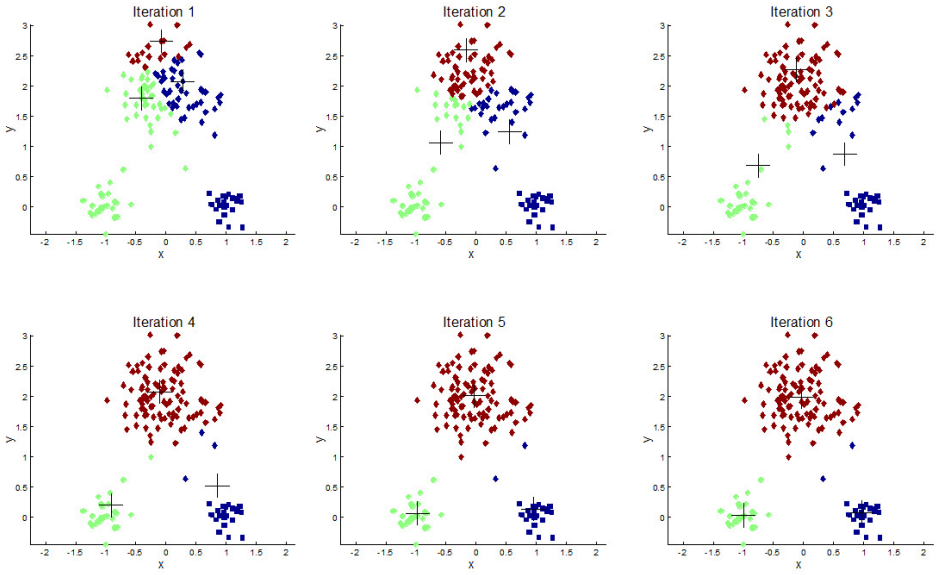


Figure 2 above shows 6 iterations (runs) of the k-means algorithm, where the points are coloured by which cluster they belong to. In iteration 1, the centers were all initialized randomly at points within a single large cluster at the top, producing a very poor clustering of the points. Subsequent iterations drag the cluster centers out as the new cluster centers are computed each iteration, and by the 6th iteration, a sensible clustering has been found.

Exercise 1: Implement the k-means algorithm:

Your task is to implement the k-means algorithm, and run it on the provided data. Open the file 'kmeans.py' and fill in the missing code to implement the algorithm - comments have been placed to guide you. Also take a look at the data provided which consists of the two variables (life expectancy, birth rate) measured for each country, and there is one dataset for 2008 and one from 1953. This data is in a csv file, which can be opened just like a text file. Open each file and take a look at how the data is laid out. Your algorithm should work on either data set.

You should let the user decide how many clusters there should be (although, if you are unsure, it might make sense to begin with just two clusters, and generalize later). The algorithm will need to run for a user-specified number of iterations, and output:

1.) The number of countries belonging to each cluster
2.) The list of countries belonging to each cluster
3.) The mean Life Expectancy and Birth Rate for each cluster

How many iterations do you need to use? Well, that depends on the data, and the number of clusters chosen. How can we tell if the algorithm has converged? To get into this, we should talk more about convergence and "optimization". Optimization, very broadly, is taking some measure of goodness, and making it as good as possible. The measure of goodness is called an "objective function", and the value of this objective function depends on both the data, and on the parameters (which are just the cluster means, in this case). The k-means algorithm is actually solving an optimization problem, where each iteration minimizes the value of an objective function. In this case, the objective function is the sum of squared distances between each point and the cluster center that each point belongs to. "Convergence" refers to the fact that the changes in the value of the objective function get smaller and smaller, until eventually the value appears to be the same from one iteration to the next, and we can say that the algorithm has converged.

Exercise 2: Monitoring convergence:

In your main iteration loop, for each data point, calculate the squared distance between each point and the cluster mean to which it belongs, and sum all of these squared distances. Print out this sum once each iteration, and you can watch the objective function converge. Make sure you are picking enough iterations to allow the algorithm to converge. If the value of the objective function gets worse, then you either have a bug in your k-means algorithm, or a bug in your objective function calculation (or both!). Thus, explicitly calculating the objective function also serves to test your code. We call this a "sanity check".

There are three provided datasets: data1953.csv, data2008.csv, and dataBoth.csv. The first two contain 197 countries, with Life Expectancy and Birth Rate measured for each country, but the first has these measurements taken from 1953, and the second from 2008. dataBoth.csv contains both the 2008 values and the 1953 values, but pretending that the countries from different years are different countries. Thus, we can see, for example, that (1953)Zimbabwe falls in the same cluster as (2008)Zimbabwe, but that (1953)Botswana and (2008)Botswana fall into difference clusters.

Exercise 3: Interpreting the data:

Create a textfile called 'interpretation.txt' and fill in your answers to the questions below.
Run k-means using 3 clusters on the 1953 and 2008 datasets separately. What do you see? Take note of how the clusters change from 1953 to 2008. You will need to pay attention not only to which countries are in clusters together, but also to the Life Expectancy and Birth Rates for those clusters. Next, run the algorithm with 4 clusters on dataBoth.csv. Note any observations in your text file. Which countries are moving up clusters? How does "(2008)South Africa" compare to "(1953)United States"? Are there any 2008 countries that are in a cluster that is made up mostly of 1953 countries? Try and explain why. Are there any 1953 countries

that are in a cluster that is made up of mostly 2008 countries? Try and explain why in your textfile.

Optional task:

So we know that each iteration of the k-means algorithm improves an objective function, the total within-cluster squared distance, which is a measure of how good the clustering is. But each time you run it, the initialization is randomly selected. What happens if you run it (until convergence) a number of different times? Do you always get the same clustering? It turns out that you don't. This is because, while each iteration improves the objective function, it isn't guaranteed to find the "globally" best clustering. This is why we say that the algorithm finds a local maximum. What can we do about this?

Optional exercise 4: Attempt to find the global optimum:

While it is hard to find the optimum with certainty ("NP-hard", in fact. See http://en.wikipedia.org/wiki/NP-hard), we can at least improve upon our current setup. Each time you run the k-means algorithm until convergence, you start from a different place, and may find a different local optimum. However, the algorithm runs to convergence extremely quickly. One simple but commonly used trick in machine learning for algorithms that run quickly but only find local optima is to rerun the algorithm a large number of times, starting from a random point each time. When doing this, you will need to store the best solution, using the objective function to judge which run gave the best solution. Code up this "multiple restart k-means", using 100 distinct runs from random starting locations, outputting the best solution of the 100 runs. Redo exercise 3 using this, and see if any of your conclusions change!

Sometimes datasets will have more than two variables. We refer to the number of variables measured for each point as the "dimensionality" of the dataset. The data we are working with is 2-dimensional. As such, it can be plotted on a plane. When a 3rd dimension is included, you need a 3 dimensional plot. When the number of dimensions grows greater than 3, plotting the data becomes very difficult, and there is a whole field called "data visualization" that seeks good ways to plot high-dimensional data. k-means clustering, however, doesn't get any harder in higher dimensions. Instead of calculating the two dimensional Euclidean distance, we calculate a higher dimensional version of it, and we take the mean in more dimensions as well.

Extra credit exercise 5: Higher dimensions:

First, go and find or make a dataset with more than 3 dimensions. You could try and get data from www.gapminder.org, or from any other source. Then implement k-means in higher dimensions, and run the algorithm, and interpret the results.

Extra credit exercise 6: Plotting the algorithm:

Back to 2 dimensions, attempt to plot the points in a similar manner to figure 2. Each point must be coloured according to which cluster it belongs to, and the locations of the cluster centers must be displayed in some easy to see way. You will need to install matplotlib (or some other python plotting library) to achieve this. Don't be shy to google for help when you get stuck plotting something. And don't forget to read the manual!

Written by Benjamin Murrell and edited by Riaz Moola.