TASK

Machine Learning IV

Visit our website

Introduction

WELCOME TO THE FOURTH MACHINE LEARNING TASK!

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.

Get in touch

Connect for support

Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

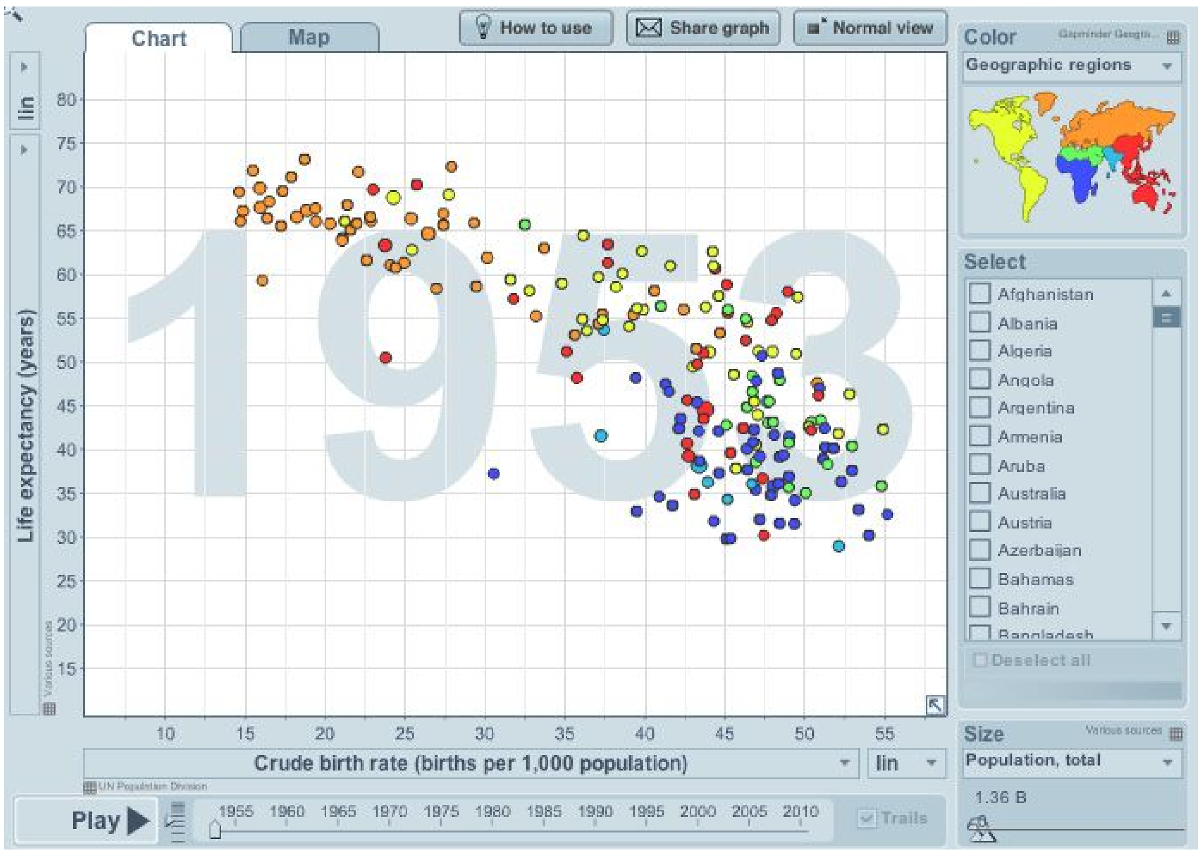
The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

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 — i.e. 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 **Gap Minder**):



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 lets 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.

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**. "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:

- The centre of each cluster is the mean of all the data points that belong to it (hence the name "k-means"). We call this the "centroid" of a cluster.
- Each data point belongs to the cluster with the nearest centre point.

These two assumptions are actually sufficient to describe the entire algorithm. All the k-means algorithm does is iterate between the 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**" and "**Chebyshev distance**", but the one we will be using here will be the familiar "**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) is just $\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$. In Python, each data point can be stored as a list with two elements, the coordinates (x, y). Thus, you can 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 of any sort, you simply add all the observations together and then divide by the number of observations you added together. In the k-means algorithm, we need to find the mean coordinate point (x,y). This means we need to calculate the two-dimensional mean (which is called two-dimensional because we are working in a coordinate plane with two axes, x and y). Calculating the two-dimensional mean is no different from calculating any other mean. 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.

SPECIFICATIONS OF K-MEANS

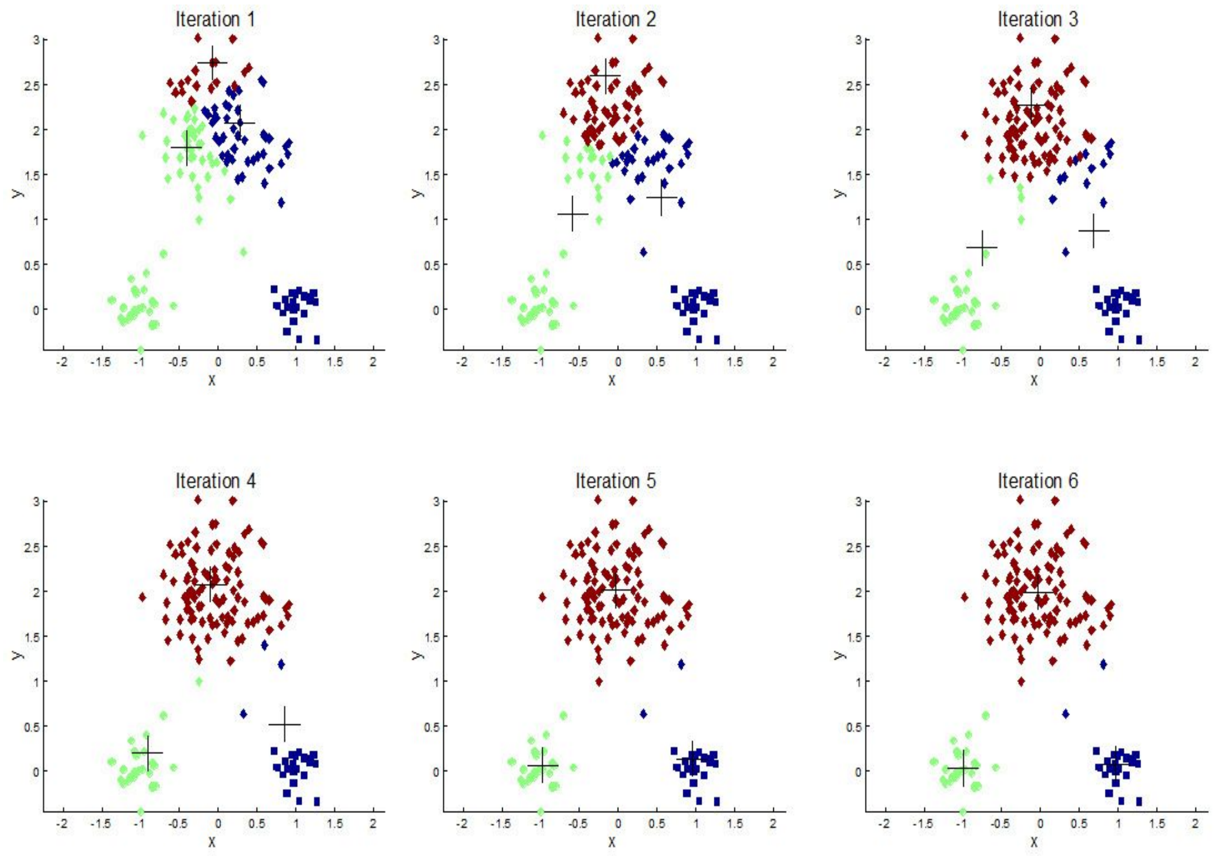
We are now ready to outline the algorithm:

- Initialise a mean for each cluster as the cluster centre (the centroid), 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 following steps a pre-specified number of times:

- Assign each point to the nearest cluster. This will require that you compute the distance of each point to each of the current three mean points and select the one with the smallest distance.
- Compute the new cluster centre (mean) 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 between these two steps 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 that continuing to run the algorithm doesn’t make this value change, i.e. the value has ‘stabilised’ — we’ll go into more detail on this a little later.



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

CONVERGENCE

After reading the above, you may have wondered how many iterations you need to use on other data sets. This depends on the data, and the number of clusters chosen. The correct number of iterations will allow the algorithm to converge. To understand this, we should talk more about convergence and "optimisation". Optimisation, 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 centroids in this case). The k-means algorithm is actually solving an optimisation problem, where each iteration minimises the value of an objective function. In this case, the objective function is the sum of squared distances between each point and the cluster centre 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.

Compulsory Task 1

Your task is to implement the k-means algorithm and run it on the provided data using the following steps as guidance:

- Open the file **kmeans.py**, in which comments have been included to help guide you. Take a look at the data provided, which consists of the two variables (life expectancy, birth rate) measured for each country, and note that there is one dataset for 2008 and one from 1953. This data is in csv file format, 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.
- To assist you in implementing the k-means algorithm, consider writing functions that can do the following:
    - Find the Euclidean distance between two points, each of which has an x and a y coordinate. A reminder that this is given by the formula $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$.
    - Find the distance to each of the centroids from each of the points. Ie. If you have 15 points and 3 centroids, you need to calculate the distance from each of the 15 points to each of the 3 centroids.
    - Compute the two-dimensional mean. 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. Thus, to compute the two-dimensional mean (ie. the mean coordinate point (x, y)), calculate the mean of the x values and the mean of the y values of all the points.

- Read in the data from either of the provided csv files. Remember that BirthRate is the 'x' value and LifeExpectancy is the 'y' value. Hint: after reading in the data from the csv file, the data should be in a form where you should easily be able to plot all the data as a scatter plot to get an idea of what the data distribution looks like.
- Visualise the data as a scatter plot, with the points in each cluster shown in a different colour.

- After doing this, implement the k-means algorithm using the steps described in the 'Specifications of k-means' section above. You should let the user decide how many clusters (k) there should be (although it might make sense to begin with just two clusters, and generalize later. I.e. k=2). The algorithm will need to run for a user-specified number of iterations, though to begin you can hard-code this in with the value 6 and update this later. You do not need to monitor the algorithm for convergence at this point.

- Once the algorithm has run, it should also 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

- You will use kmeans.py in the following two tasks, so ensure that it is completed and working properly before moving onto the next task.

Compulsory Task 2

Follow these steps:

- In your main algorithm iteration loop, for each data point, calculate the squared distance between each point and the centroid to which it belongs, and sum all of these squared distances.

- Print out this sum once on 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. The last file, 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 different clusters. Run your script on the different datasets and ensure that you can see this.

Compulsory Task 3

Follow these steps:

- Create a text file 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 BirthRates 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 text file.

Rate us

Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved or think we’ve done a good job?

Click here to share your thoughts anonymously.