K-Means and Star Types

Eckel, TJHSST AI2, Spring 2023

Background & Explanation

In this assignment, you'll learn the *k*-means algorithm. This is a simple example of an **unsupervised learning** algorithm in machine learning; that is, this algorithm attempts to find regularities or patterns without checking its learning against previously determined right answers. For the rest of fourth quarter, we'll be focusing on **supervised learning** algorithms that have a specified training set that includes desired outputs. But for now: *k*-means.

Specifically, *k*-means is an unsupervised learning algorithm designed to look at a lot of data and find meaningful groups. There's an article here - https://blogs.oracle.com/datascience/introduction-to-k-means-clustering - which goes into more formal mathematical detail, for those of you who are curious, and also outlines the use of this algorithm on a data set of delivery drivers. The article is unfortunately missing its images now but is still comprehensible. (Also: fun fact, it's written by a friend of mine from college, Andrea Trevino. She's now the data science team lead for Riot Games, the company that makes League of Legends!)

In Andrea's example, the algorithm is able to find groups representing rural drivers, urban drivers, and drivers that speed / drivers that drive safely just from some raw data. This is the basic idea: *k*-means is used to find meaningful clusters. In this case, "rural drivers who speed" is a meaningful cluster. You can imagine a chain store using this process to classify customers into groups for advertising purposes, for instance, or a medical study trying to group subjects into predictive subgroups.

We'll demonstrate the strengths and weaknesses with *k*-means by applying it to a data set of different stars and seeing what clusters it finds, and how similar those are to the actual classifications of real stars. Then, in the next assignment, we will use *k*-means to reduce an image down to a small number of individual colors, something similar to the "Posterize" button from PixLab in Foundations of Computer Science at TJ but *much* more effective.

In this case, we'll use a data set of various stars (taken from https://www.kaggle.com/deepu1109/star-dataset) and each data point will consist of its measures of surface temperature, luminosity (relative to the sun), radius (relative to the sun), and absolute visual magnitude. (Well, almost; there's a note about reading in this data set later on in the assignment that you need to read before you implement anything here!)

Before you start this assignment, **visit that Kaggle link** and click "view more". In addition to a description of the data set itself, which is worth reading, you should take a look at the diagrams that show the different types of stars as classified by astronomers so you can see what we're aiming to find.

We'll be looking at the "star type" in this assignment; note the categories as described on the Kaggle page:

- Brown Dwarf -> Star Type = 0
- Red Dwarf -> Star Type = 1
- White Dwarf-> Star Type = 2
- Main Sequence -> Star Type = 3
- Supergiant -> Star Type = 4
- Hypergiant -> Star Type = 5

The goal here is to let the *k*-means algorithm take these stars and group them naturally, without knowing their categories, and then look at the categories it creates and see how well they match the real ones.

Don't start coding yet; you want to understand this algorithm completely before you start.

Algorithm

So: we want to find the best natural groups of data points in our data set. We'll call each data point a vector, in the same sense as a tuple – a collection of values. In the case of this data set, each data point is a 4-tuple, or a 4d vector.

The *k*-means algorithm goes like this:

1) Specify a value of **K**.
2) Choose **K** *distinct* random elements of the set. (In our case, the specific vectors representing **K** random elements from the data set, ensuring that all **K** of the selections represent different specific stars.) Call these the *means*, though we haven't averaged anything yet.
3) Associate each mean with a list of stars. Loop over every single star and assign it to the list belonging to the mean that is closest to that star's data. We'll do this by finding the squared error between the star and each mean and assigning it to the mean that results in the smallest squared error. You **may not** use numpy or scipy to do this; this is just the distance formula on multiple dimensions. If you don't know what I mean by this ask me – don't google a package you don't need. (Note also that in this particular assignment we'll use the logs instead of the values for some variables; more on this below.)
4) Take each list of stars generated in step 3 separately. Find the actual mean of these stars. That is, form a new vector with the average value of each star's surface temperature in this group, then the average of luminosity, etc. Intuitively, you can see that – at least at first – these values will be quite different from our initial "means" that we randomly guessed in step 2.
5) Repeat steps 3 and 4 over and over. Since the values of the means have changed, stars that used to belong to one mean's group will now shift to a different one. Therefore, as step 4 repeats, the means will change again… and the points will move again… etc. Keep repeating until this becomes stable; that is, until no star changes group / no mean changes value. (This means you'll need to keep track of the groups and how many stars move in or out during each round.)
6) When the process resolves, you've found the **K** specific means that minimize the squared error! (Well, pretty close; the algorithm isn't mathematically guaranteed to converge to optimality. In practice it is quite robust, but may produce a variety of near-optimal answers.) Each mean is paired with a group of data points; these are the natural groups that the algorithm has found.

But wait: before you implement this, we need to discuss a problem.

A problem with this data set / *k*-means in general

If you look at the data explorer summary on Kaggle, you'll see that the first three measurements in the data set vary exponentially, with a majority of data points in a small range and then a long tail as the values increase. Intuitively, you might see a problem here already. In the *k*-means algorithm, we measure the "closeness" of one data point to each of the means using squared error. But: with exponential variation, a single mean near the bottom will be closest to the vast majority of data points. This will make for unequal groups.

This is a general weakness of *k*-means – it only applies when your variables vary approximately linearly, and the groups you want to make are all about the same size. In our case, the groups are about the same size – the data set has done this for us, each final category has the same number of stars – but the exponential variation will mess up the algorithm.

Here's how we'll address this problem. When you read in the data set and create each data point's vector, instead of being:

<Temperature, Luminosity, Radius, Absolute Visual Magnitude>

We'll do this instead, using Python's log command from the math library (the default base e works just fine):

<log(Temperature), log(Luminosity), log(Radius), Absolute Visual Magnitude>

By taking log of each of the values that varies exponentially, we turn them into values that vary linearly.

Implementation

So to be specific, your code needs to:

1) Read in the data set. If you've never worked with a .csv file before, it's not anything too complex to deal with; read it like a text file, and each line will contain all the values separated by commas. Call .strip() as you always would and then call .split() on a comma character to divide the line into appropriate substrings. (Students keep asking me about the pandas package, which must be what comes up if you google this; my grading script's Python install for sure does NOT have pandas on it so do not use that!)
2) Create a data structure that stores each data point's numerical vector (using logs for three of the values as explained on the previous page) and its star type (which won't be considered as the algorithm runs, but which we'll want to look up later to see how well it did). Discard the star color and spectral class variables. **(NOTE: we are NOT using the LAST column, the spectral class, to define our categories. DISCARD the last two columns; we're using the numerical 0-5 value for "star type"! If you have questions about this, see the Kaggle page for the data set.)** Also note: **you do not need pandas to read in this data file**, and solutions which use pandas will be rejected (too easy to cheat). A csv file is just a text file with values separated by commas; call split on "," and you're good to go.
3) Considering each data point's 4-value vector, run the k-means algorithm as described before. There are 6 star types, so setting k to 6 is a logical first step. When the algorithm ends, you should have 6 mean vectors, each associated with a list / set of stars.
4) Print out each mean followed by all of the stars associated with that mean **and the star type of each one**. (Once again: NOT the spectral class, which is a letter. Print the STAR TYPE, which should be a number from 0 to 5.)

Do several runs of your code; perhaps try with 5 or 7 means.

Discuss Your Results

If you'd like, you can work together with another student to answer these questions; compare the output each of you gets and verify they both seem about the same, then answer these questions together. If you do work with a partner, be sure **each** of you messages me your answers **along with a note about who you worked with**.

Send me a Mattermost message with the following answers:

1) How close does *k*-means get to identifying the six separate types of stars?
2) What variation do you see across multiple runs of your code? Does the algorithm find exactly the same groups every time? If you tried 5 or 7 means, what difference did that make?
3) Look back at the Kaggle page, and look at the stellar diagrams given under "Dataset Info". If you imagine six means moving around these images and gathering stars close to them as groups, which type of star seems least likely to make a successful cluster? Does this match what you saw in your runs (ie, is that type of star more frequently distributed among different *k*-means clusters?)
4) After reading the original info page that described using this algorithm on truck driver data as well as the experience you have here, briefly discuss the strengths and weaknesses of this algorithm as you currently understand them.

Specification for *k*-Means: Stars

Once you've messaged me on Mattermost, read the specification below. Make sure your code follows it, and submit **a single python script** to the link on the course website.

This assignment is **complete** if:

- You follow the instructions on the submission form to format your submission properly.
- Your code runs as described above, completing a k-means process with 6 means and outputting each mean, the stars in that cluster, and the type of each star. Make sure your output is clearly formatted so I can see where each group ends. The grader directory will have the star_data.csv file in it.
- Runtime should be very fast – single digit number of seconds.
- You also must complete the task on the previous page and message me to receive credit.