ENGR 151 Project 4

Thursday October 27th 2022

This assignment is due Thursday November 17th 2022 at 11:59PM

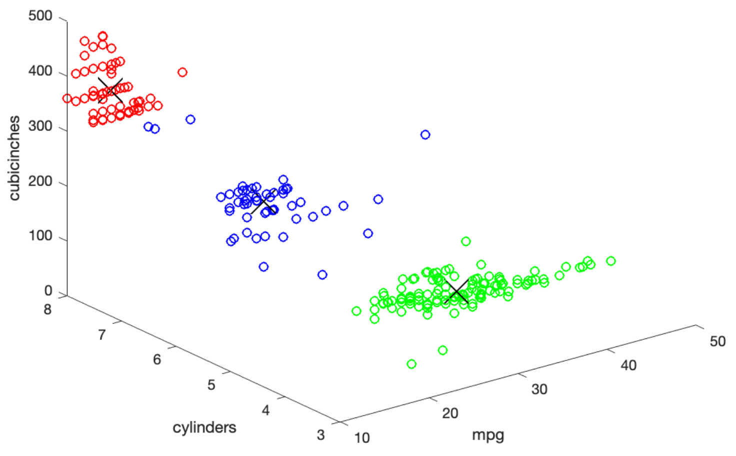

Figure 1: K-means clustering.

1 Overview

K-means clustering is an introductory *machine learning* technique where a dataset with many variables is analyzed and broadly categorized into K "clusters" of data that have similar properties. For example, cars may be clustered in terms of having similar miles-per-gallon (mpg), number of cylinders and engine size in cubic inches etc., as shown by the 3D scatter plot above. In this case, there are three visually identifiable "clusters" of data. The "centroid" of each cluster, i.e. the mean position of all the points in the cluster, is used as a way of defining the cluster, with the "distance" from each centroid being used as a way of determining which cluster a datum belongs to. Once clusters are established, by identifying the nearest cluster to new, incomplete, data, the full properties of the data set may be predicted. Such an analysis technique is useful in many areas, such as understanding customers for market research, object recognition in images, or robotics, to name but a few applications.

There are a few basic steps to define the K clusters:

1. Randomly initialize K centroids.
2. Assign *normalized* data to K clusters by finding the shortest distance to a centroid.
3. Re-compute all K centroids by finding the mean position of each cluster based on the data associated with it.
4. Repeat steps 2 and 3 until convergence.

In this project, you will have access to a data set `cars.csv`, which includes a table of information on the mpg, cylinders, engine size (cubic inches), horsepower (hp), weight, 0-60 miles-per-hour time, year and geographical origin (labelled "brand"). Using this table you should perform a K means clustering analysis to find clusters in the parameter space mpg – cylinders – cubic inches and find correlations with other properties in the table.

2 Specifics for project credit

This is a more involved code that the previous projects. It is very important to write this code in a modular way, testing each function correctly. To help you to structure your code, we include some scaffolding (two .cpp files) in the project 4 folder which contains a main function with a series of steps that need to be implemented. You are expected to define functions and call them in the main function. We also provide a header file (.hpp file) that includes the definition of a structured data type `car_data` and a few implemented functions (e.g., `read_csv`, `initialize`). You can use the structured data type for storing the data and call the functions in your .cpp files. Read and fully understand the .hpp file before starting your implementation. Please do not modify the .hpp file since it will not be submitted (your code may not compile on Autograder if you work with a modified .hpp file locally).

2.1 Part 1: Generate clusters

You should submit a C++ code to the autograder named `kmeans_clusters.cpp`, which must import a csv file named `cars.csv`. Your C++ code should do the following:

1. Read in the data from the "cars.csv" file accessible from Canvas (and provided in the autograder). To improve uniformity of code for debugging purposes, a header file named `kmeans.hpp` has been included. This provides definitions of a `car_data` structure (struct) and a function

   ```
   void read_csv(std::ifstream &fin, car_data &dat);
   ```

   for reading in data from the data set in the file `cars.csv` and placing into the structure. This can be added to your code by writing `#include "kmeans.hpp"` as a preprocessor directive. Scaffolding for `kmeans_clusters.cpp` and `kmeans_identify.cpp` is also provided on the Canvas website.

2. Normalize the numerical values of each feature to allow comparison. Normalization in this context means subtracting the minimum value and then dividing by the maximum value of that feature (*after subtraction*), for example for mpg subtracting the minimum mpg and then dividing all mpg entries by the maximum mpg (*after subtraction*). i.e. specifically for a feature X, the normalized version $\overline{X}$ is given by

$$\overline{X} = \frac{X - \min(X)}{\max(X) - \min(X)} \tag{1}$$

   Note that this expression is equivalent to the steps `X  = X - min(X)` followed by `X  = X/max(X)` in code. Normalization should result in the data for each feature having values between 0 and 1.

3. Initialize $K = 3$ centroids as three (3) randomly selected points in the data space defined by the features, i.e. each point is a vector/array of seven (7) entries representing the seven features (a seven dimensional parameter space!). Using the `kmeans.hpp` header file, a call to the function

```
void initialize(car_data &centroids);
```

   will initialize three centroids randomly to specification.

   Note the random numbers generated by different platforms (Mac, Windows, Linux) could be different, and this makes the debugging process more challenging although a correctly implemented algorithm should converge to the right solution in spite of the random numbers (initial centroids). Nevertheless, another version of the header file with fixed random numbers (generated by the Linux system on autograder) is provided. You may use `kmeans_fixRand.hpp` when debugging your code to ensure a completely consistent behavior between autograder and your local machine.

4. Run the K-means clustering algorithm as described in the previous section, assigning points to their closest centroids and then updating the centroids for the three clusters. The closest centroids should be found using the Euclidean "distance" d, i.e. the square root of the sum of the squares of $X_i(\text{data}) - X_i(\text{centroid})$, where X_i are the positions in the normalized data space (i.e. X_1, the normalized mpg, X_2, the normalized number of cylinders etc.).

$$d = \sqrt{\sum_i \left(X_i(\text{data}) - X_i(\text{centroid})\right)^2}\,.$$

   Check for convergence of the algorithm, i.e. the centroids change by less than a very small amount, i.e an $\varepsilon = 10^{-10}$.

5. Export a csv file named `cars_clustered.csv`, which should be otherwise identical to the first 7 columns of `cars.csv` but should contain an additional column at the end of

the other columns that contains the cluster number (i.e. 1-3) that you calculate, along with the header label `cluster`.

You should output a file `clusters.csv`, which contains the same headers as in the file `cars_clustered.csv`, but with the information for the 3 cluster centroids that you identify with your K-means code in the 3 rows of data, i.e. in the format:

```
mpg, cylinders, cubicinches, hp, weightlbs, time-to-60, year, cluster
12, 8, 300, 163, 4109, 12, 1972, 1
18.9, 4, 89, 69, 1925, 14, 1960, 2
12, 8, 302, 120, 3229, 11, 1923, 3
```

A function

```
 void print(std::ostream &fout, const car_data &data);
```

in the `kmeans.hpp` header file provides a method for printing to an output stream in this standardized format.

2.2 Part 2: Graph output

With this algorithm, find the $K = 3$ clusters, export the data and generate the 3D scatter plot in MATLAB for the features `mpg`, `cylinders` and `cubicinches`. Color the points using an array `colors`, with the three colors red, green and blue corresponding to the clusters that you identify.

To graph the data so that it is uniform for all submissions for grading purposes, you must use `scatter3` in the following way:

```
scatter3(dataX, dataY, dataZ, 50,colors);
hold on
scatter3(centroidX, centroidY, centroidZ, 500, 'k', 'X');
xlabel('mpg');
ylabel('cylinders');
zlabel('cubicinches');
hold off
```

where `dataX`, `dataY` and `dataZ` are the arrays for mpg, cylinders, cubicinches, and `centroidX`,`centroidY` and `centroidZ` are the centroid positions. `colors` is an array of the colors *indicating* the cluster assigned to each datum displayed. Please refer to the manual of scatter3 for setting up the array of `colors`.

You should generate a picture `clusters.jpg` from the `cars_clustered.csv` and `clusters.csv` data. Note that such a diagnostic will also be useful for seeing what your code is doing for debugging purposes. **The picture should be submitted to Canvas**.

2.3 Part 3: Identify mystery cars

Your code should then be able to take in an input vector `mystery_cars` with seven (7) features , which will be cars that are not part of the training data set, and identify them as belonging to one of the clusters (which we could call “sports cars”, “fullsize cars” and “compact cars” for example).

You should submit a C++ code to the autograder named `kmeans_identify.cpp`. This code should input from stdin a list of seven (7) features (i.e. mpg etc. which will be inputted as a whitespace delimited list) as in the order of the first 7 columns of `cars.csv` for a mystery car. Your code should identify the type of car (type 1, 2 or 3), consistent with the cluster centroids given in the file `clusters.csv` and your clustering algorithm. The code should exit with a single statement “`Mystery car is type <X>.`”, where `<X>` should be the number of the closest centroid, e.g. “`Mystery car is type 1.`”.

3 Grading

First, successful submission of a graphic as described in the specifications will provide a maximum score of 10 points. 50 points will be given for generation of correct `cars_clustered.csv` and `clusters.csv` files. 40 points will be given for correct identification of several mystery car specifications. Second, one of our graders will evaluate your submission for style and commenting, and will subtract 0-10 points from the score that autograder evaluated. In total, 100 points are possible on this project. The following is the breakdown for each part:

# pts	Description
10	Generation of correct graphic as specified above.
50	Generation of correct `cars_clustered.csv` and `clusters.csv` files.
40	Correct identification of several mystery car specifications

# pts	Description (pts will be subtracted if item is missing or insufficient)
2	Each file has name/section number/submitted date included in a header comment
2	Comments are used appropriately (e.g. major steps explained)
2	Indenting and whitespace are appropriate (including functions properly formatted)
2	Variables are given descriptive names
2	Overall program structure (e.g, using functions instead of repeated code blocks)