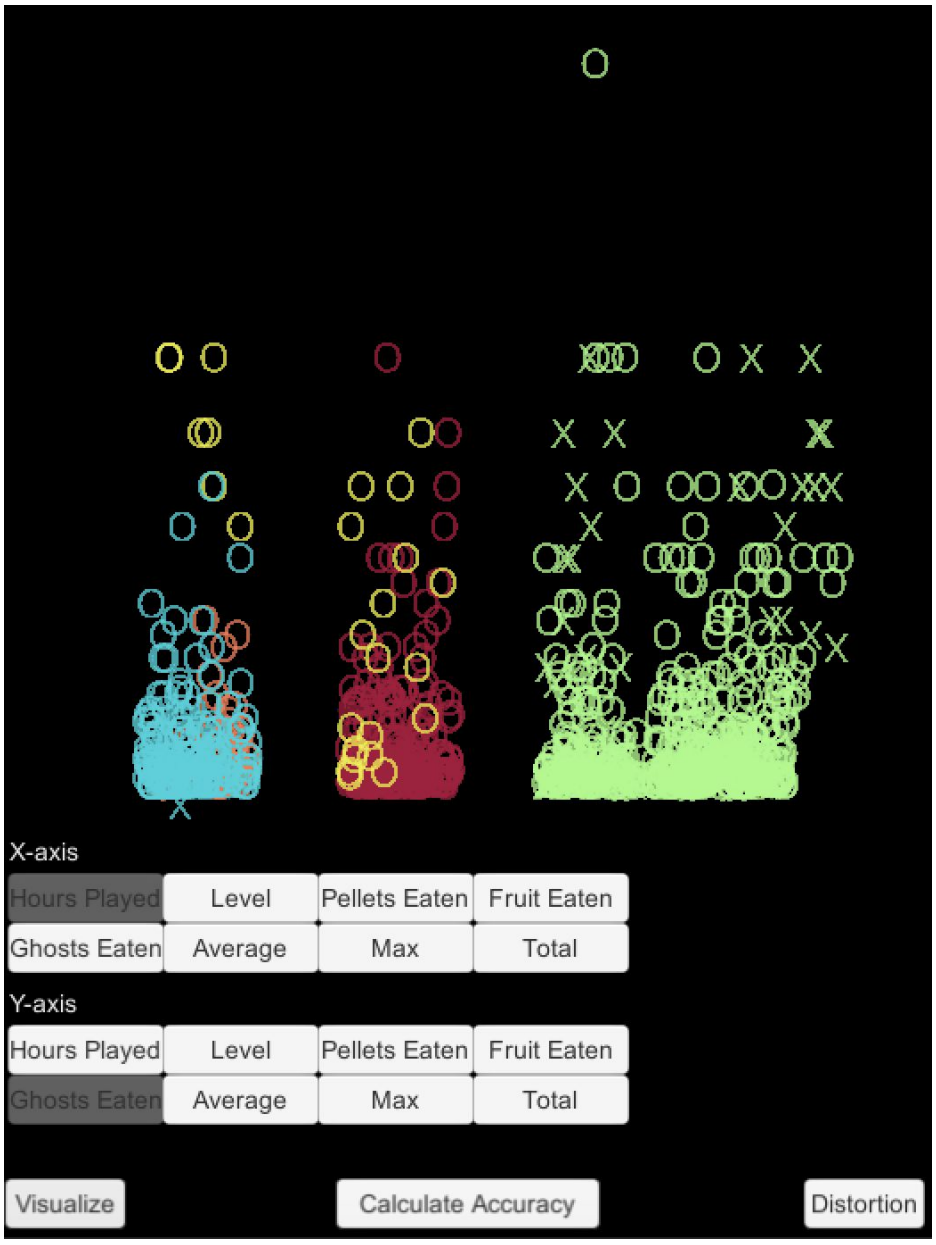
Assignment 4: Clustering

The purpose of this assignment is to make you familiar with some basic approaches of data science (model free player modeling) for games. Clustering (specifically K means and K medoids clustering) are common ways to identify player types and player strategies in games using real player data.



In this assignment, you will be provided with a great deal of synthetic player data. You will implement and apply either K means or K medoids clustering to this data, which will include

implementing a distance function and finding the correct value of K. For extra credit you can attempt to predict churn, whether or not a player will stop playing the game.

What you need to know

There are two scripts you will interact with for this assignment, and then one script you may interact with for the extra credit.

KClusterer

KClusterer handles the clustering behavior. You can choose whether to implement K means or K medoids.

Member variables:

- K: The number of clusters to create. **You will be responsible for setting this to the correct value.**
- MAX_ATTEMPTS: A variable you may (but do not have to) use to specify the maximum number of times your code should attempt clustering.
- threshold: A variable you may (but do not have to) use to specify how close two sets of centers must be to be considered identical. You will need to alter this when you change the distance function.

Member functions:

- Dictionary<Datapoint, List<Datapoint>> Cluster(Datapoint[] datapoints): The basic cluster function that you will need to fully implement for this assignment. Currently, it takes in an array of Datapoints and returns a random set of clusters as a dictionary of the center DataPoints to a list of the points clustered to that center. **You will need to alter it so that it correctly implements K means or K medoids clustering.**
- float Distance(Datapoint a, Datapoint b): The basic distance function, returns the distance between two DataPoint inputs (a and b). **You will need to alter it as well so that it better reflects differences between DataPoints.**
- float DifferenceBetweenCenters(Datapoint[] centers1, Datapoint[] centers2): Takes in two sets of DataPoint arrays, representing two sets of centers. Uses the Distance function above to calculate the summed distance between the most similar center points. This will be helpful in determining if your clustering approach has completed.
- Datapoint GetMedian(Datapoint[] datapoints): Takes in an array of DataPoint values, meant to represent a cluster, and returns the median point. You will use this if you implement K medoids clustering.
- Datapoint GetAverage(Datapoint[] datapoints): Takes in an array of DataPoint values, meant to represent a cluster, and returns the average point. You will use this if you implement K means clustering.

Datapoint.cs

This script holds the Datapoint class, which holds the summarizing statistics for a single player.

Member public variables:

- UserID: public getter for the user's id string.
- HoursPlayed: An int for how long the player has played the game.
- Level: An int representing the level of the player.
- PelletsEaten: An int representing the sum of all the pellets eaten by the player.
- FruitEaten: An int representing the sum of all fruit eaten by the player.
- GhostsEaten: An int representing the total number of ghosts eaten by the player.
- AvgScore: A float representing the average score the player achieves per game.
- MaxScore: An int representing the max score the player ever achieved.
- TotalScore: An int representing a sum of the total score achieved by the player.
- Churned: A boolean representing whether or not the player quit playing.

Member functions:

- Datapoint:There are two constructors, one that passes in the churned value from the data and one that does not. You will not need to use either constructor in your clustering implementation.
- SetPredictedChurn(bool _churned): Sets the passed in value to the Churned variable. Only used for the extra credit.
- bool Equals(object obj): Determines if two Datapoint objects are equal.
- int GetHashCode(): Gets hashcode for this Datapoint
- string ToString(): Returns a string representation of the Datapoint, useful for debugging.
- float GetValueByString(string variableName): Returns the associated float for the given name. You will not need to use this function in your clustering implementation.

The extra credit script is ChurnPredictor.cs. It has a single function "AssignPredictedChurn". The goal of the function is to predict whether or not a given player will churn (stop playing the game).

This function takes as input a set of Datapoint values(verificationDatapoints) and the clustered training data (clustersByCenters). For the extra credit your job is to correctly predict the churned value for each Datapoint in verificationDatapoints and set it with "SetPredictedChurn".

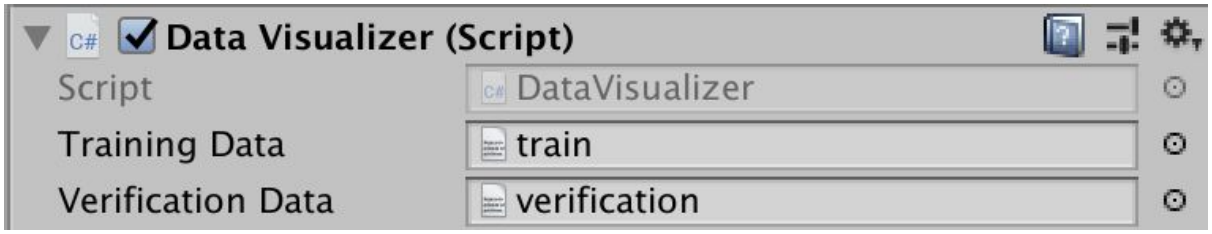
Instructions

By default the project will create three random clusters, which you can visualize in the Data Visualization scene by clicking two of the variable names followed by the Visualize button (on the bottom left). The bottom right of the screen button “Distortion” will calculate the average distortion of the clusters and the middle button “CheckAccuracy” will run the “AssignPredictedChurn” function of ChurnPredictor.cs, which is only relevant to the extra credit.

Step 1: Download the Assignment4.zip file from eclass, unzip it, and open it via Unity. You can open it by either selecting the unzipped folder from Unity or double clicking “Data Visualization”.

Step 2: Open “Data Visualization” located inside Assets/HW4. Hit the play button. Click on two attributes and click the “Visualize” button to visualize the clusters according to those attributes. Each cluster will have a unique colour.

Step 3: Find your name in the Assets>HW4>Assignment 4 Data folder, and drag your training and visualization data into the appropriate slots on the DataVisualization component of the Main Camera in the scene Hierarchy. **Note**: Everyone has data unique to them. You **MUST make sure** that you use your assigned data for this assignment in order to receive full marks.



Step 4: Implement K means or K medoids clustering in the KClusterer.cs Cluster function from the pseudocode supplied in class. Alter the given Distance function to better compare between Datapoint values.

Step 5: Identify the correct value of K (K can only be >=2 and <=10) from visual inspection and the elbow method (as described in class) based on the average distortion, which can be printed to the console by clicking the “Distortion” button. **Note**: You will be unable to accomplish this step unless you first finish Step 4.

Step 6 (Optional): Write code in the “AssignPredictedChurn” function of ChurnPredictor.cs to improve the default accuracy of 50%.

Grading

This homework assignment is worth 10 points. Your solution will be graded by an autograder. The point breakdown is as follows:

4 points: A correct implementation of K means or K medoids clustering. This approach should not run infinitely and should not lead to Datapoint values clustered to the wrong center. You will receive partial credit depending on the percentage of Datapoint values correctly clustered to the closest center.
4 points: Your code correctly clusters Datapoint values from the same (hidden and inaccessible to you) clusters together. You will receive partial credit for this based on the percentage of correctly clustered datapoints. Given the random nature of clustering your code will be run 10 times with the best clusters used for grading.
2 points: Identification of the correct value of K for your data. You will lose 0.5 points for every 1 value of K you are off by. For example if the correct value of K was 3 and you used K=5, you would lose 1 point. You cannot lose more than 2 points for this part of the grading rubric.

You **may not** use any additional libraries (data visualization, machine learning, statistics, etc.) in any of the code you write for this assignment. You can only make changes inside KClusterer.cs and (if you do the extra credit) ChurnPredictor.cs.

In addition you will receive 0.5 extra credit points if your "AssignPredictedChurn" function successfully manages to achieve >0.55 accuracy on a test set of Datapoint values you do not have access to. You will receive 1 extra credit point if your implementation achieves >0.75 accuracy. You will receive 0.1 extra credit points for any substantial attempt at improving ChurnPredictor.cs

---

Hints

You are provided with a number of reference training and validation sets (under Assets>HW4>Assignment4 Data/example). They have the following true K values

example: 3
Matthew Guzdial: 6
Nazanin Yousefzadeh Khameneh: 5
Christoph Sydora: 9

You will need to change the Distance function inside KClusterer to succeed at this approach. Consider how you can ensure that each summarizing metric/variable has the same weight. Consider, should a difference of 1 Fruit Eaten be the same as a difference of 1 Ghost Eaten? A player is likely to have eaten far more ghosts than fruit.

It is unlikely you will be able to guess the correct value of K from purely visual inspection. I highly recommend first implementing K medoids or K means clustering, then use the elbow

method with the provided Distortion button (that will print out the average distortion ratio). Run this multiple times with different values of K to determine the best value according to the elbow method, and then confirm this by looking at the data directly or through visual inspection.

As a reminder from lecture, the elbow method looks to find the value of K with the greatest relative change in the slope of the line from before this value of K to after this value of K.

If you dislike the provided colours you can change them by changing the Cluster Colors array on the Data Visualizer component attached to the Main Camera.

Each visualized “X” and “O” has all the Datapoint information available on it, which you can use the Inspector to check. This may be helpful with the extra credit.

In C# you **can’t** just set one list to another with “=”. Take the example below:

```
List<int> a = new List<int>() { 1, 2, 3, 4 };
List<int> b = new List<int>();
b = a;
b.Remove(1);
Debug.Log("a: " + a.Count);
>“a: 3”
Debug.Log("b: " + b.Count);
>“b: 3”
```

Instead you’ll want to copy over the values of a list instead of using “=”.

You may find it helpful to use ToArray (https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1.toarray?view=netframework-4.8) to convert from a List to an array, or ToList to convert from any sequence to a List (https://docs.microsoft.com/en-us/dotnet/api/system.linq.enumerable.tolist?view=netframework-4.8). You might also find it helpful to get more familiar with c# Dictionaries. https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=netframework-4.8

Submission

To submit your solution, upload your modified KClusterer.cs file and your ChurnPredictor.cs file (if you did the extra credit) to eclass.

You should not modify or upload any other files in the game engine.

DO NOT upload the entire game engine.