**Team 3 Project CS/ME/ECE532, Fall 2019**

**Modeling & Taste Matching in a Real-World Application**

# Abstract

This activity will provide practical experience applying machine learning principles of modeling the preferences of a population and using the models to make recommendations. There will be a warm up section to provide an overview of clustering techniques, a comparison between k means clusters and SVD models using simulated real-world data, and a section on collaborative filtering to recommend

This exercise is provided in the context of a social media platform that will group users by similar board game preferences and match them to recommended games. The accuracy of clustering and SVD models can be compared against the original data, however, since the data is simulated, results from game matching will have to be taken on faith.

# Estimated time for completion

10 min – Introduction to Additional Clustering Algorithms

40 min – Warm-up activity for additional clustering algorithm

20 min – Clustering GameHaven users using simulated data

20 min – Matching games to users using collaborative filtering

***Note: Both Python and Matlab will be used in the completion of these exercises.***

# Introduction to Additional Clustering Algorithms

The purpose of this exercise and related activity is to compare the strengths and weaknesses of a few types of clustering algorithms. We will be looking in detail at:

* K-means clustering
* Mean-shift clustering
* Density-based spatial clustering of applications with noise (DBSCAN)

### K-means clustering

K-means is a popular example of a centroid-based method. In a general sense, the clusters are formed by partitioning the data points based upon their closeness to each cluster centroid. Mathematically, the algorithm attempts to minimize the coherence, which can be defined as:

where is the mean (or centroid) of points belonging to set . The standard k-means algorithm (naïve k-means) can be implemented as follows:

1. *Initialize*: choose the centroids randomly from the set of data points .
2. *Assignment*: Assign each point to the cluster with the closest (least squared Euclidian distance) centroid.
3. *Update*: Calculate the new means:
4. Check for convergence, which can be determined by the change in coherence:

The advantages of k-means clustering are that it is an easy algorithm to understand and to implement in code, there exist efficient heuristic algorithms (like the standard one outlined above) that converge quickly to a local optimum, and it’s easy to apply to very large data sets.

The major disadvantage of the k-means algorithm is that the data scientist is required to select the number of clusters. This is a significant drawback as one of the primary goals of implementing a clustering algorithm is to gain insight into the data. Choosing the wrong value for will result in poor results.

Another disadvantage is that the algorithm is not guaranteed to converge to a global minimum. Convergence to different local minima based upon the initial selection of centroids can produce clusters that are wildly different. Thus, the results are not consistent or repeatable.

Finally, because points are assigned to clusters based upon their Euclidian distance to the centroids, k-means can only form spherical clusters and the algorithm will generate poor results when the data does not have the same variance in all directions. This also causes the algorithm to be sensitive to outliers, which depending upon their distance, can have a large impact on the resulting clusters.

### Mean-shift clustering

Mean-shift is also a centroid-based method, but it assigns data points to clusters by shifting them toward the local modes, defined as the maxima of the density function.

The modes can be found by building a kernel density estimation (KDE) for the data set. A kernel is essentially a weighting function that is placed at each data point. There are many different types of kernels, but one of the most popular is the Gaussian kernel, and that is what is used in the accompanying activity:

where is the kernel bandwidth.

The mean-shift algorithm starts by creating a copy of the original points. Using the KDE, the copied points are then iteratively shifted uphill against the original points until they reach a peak or local maximum.

The biggest advantage of mean-shift over k-means is that it does not require specifying the number of clusters in advance. Instead, the algorithm is data-driven and determines the number of modes based upon the kernel and the bandwidth. The cluster centers intuitively converge toward points of maximum density to generate a reasonable number of clusters.

Further, the algorithm is model free, meaning that it does not assume any shape for the data clusters. It is also simple. It relies only on the selection of a single parameter, the bandwidth/window size , which has a physical meaning and is easy to understand.

The biggest disadvantage to mean-shift is that it is an N-squared algorithm, so that it is computationally intensive and slow, especially as the number of points grows large. Moreover, the selection of an appropriate is important and not trivial. If is too large, this will result in a wide smooth KDE surface and the merger of modes. For an extremely large , there will be only one peak and all the points will be assigned to the same cluster. On the other hand, if is too small, this will generate additional “shallow” modes. For an extremely small , there will be a peak for each point and each point will be assigned to its own cluster.

### Density-based spatial clustering of applications with noise (DBSCAN)

DBSCAN is a density-based model that attempts to assign data points to clusters based upon densities present in the space. Essentially, it assigns points to a cluster if there is a certain minimum number of points present within a distance (usually the Euclidian distance). Points that are in low-density regions are marked as outliers or noise.

The basic algorithm works as follows:

1. Select an arbitrary point.
2. Find all neighborhood points within a distance . If the number of neighbors is greater than some minimum , then start a new cluster. Otherwise, label the current point as noise and go back to step 1. (Although this point could later become part of another cluster.)
3. Each point in the neighborhood (that has not been previously assigned to another cluster) is visited and assigned to the current cluster. For each of these, find all neighborhood points within . If the number of neighbors is greater than , append each of these points to the neighborhood.
4. Repeat step 3 until all points within the neighborhood have been visited and labeled, and the density connected cluster has been fully discovered.
5. Return to step 1 and repeat until all points have been visited and labeled.

Like all clustering algorithms, the selection of parameters is important. In this case, those parameters are and .

If the value for is too small, then a large part of the data will be labeled as noise. Too large and the clusters will merge, and most of the data could be assigned to the same cluster. One technique for choosing an appropriate value for is to use a k-distance graph, although that analysis is beyond the scope of this exercise. It can be state that in general smaller values of are preferable.

The parameter can be set according to the dimensions of the data:

Larger values often perform better for larger data sets or for data containing significant noise.

Like mean-shift, a primary advantage of the DBSCAN algorithm is that it does not require specifying the number of clusters in advance. An additional advantage of DBSCAN is that it also identifies outliers as noise, whereas the mean-shift algorithm assigns all data to clusters regardless of how far afield a point might be.

Another significant advantage of DBSCAN is that it can find arbitrarily sized and arbitrarily shaped clusters, even if a cluster is completely encompassed by another. (although it must not be density-connected)

The primary disadvantage of DBSCAN is that it will perform poorly if the underlying clusters are of varying density. Also, the algorithm is also not entirely deterministic. Border points that are accessible to multiple clusters will be assigned based upon the order in which the data is processed. Although it should be noted that the interchangeability of some border points is trivial compared to the discrepancies that arise in runs of k-means due to the initial selection of the cluster centroids.

# Warm-up Activity for Additional Clustering Algorithms

*Estimated time: 20 min for P1, 10 min for P2, 15 min for P3*

Scripts are provided to help you visually investigate the k-means, mean-shift, and DBSCAN clustering algorithms using randomly generated two-dimensional data.

1. **K-means algorithm**: clusters are formed by partitioning the data points based upon their closeness to each cluster centroid.
2. *Section 1.1* of the associated script will create some random two-dimensional data points centered around a selectable number of centroids and a selectable amount of noise or variance in the data. Use the following parameters to get started:

|  |  |
| --- | --- |
| Number of points: | 100 |
| Number of centroids: | 3 |
| Noise: | 0.1 |

Using these parameters, select the “Generate points” button until the script produces a data set with three distinct clusters that can be separated, like in figure 1 below

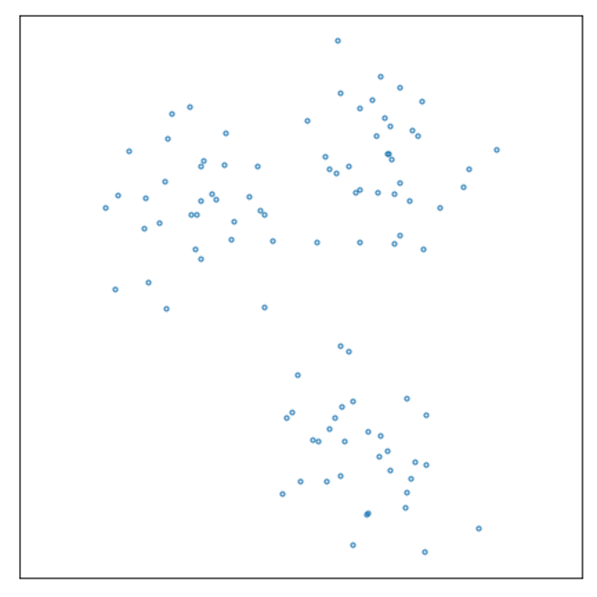


Figure 1: 100 random points in 3 distinct but loose clusters

1. *Section 1.2* of the script will cluster the data using the standard k-means algorithm, and it will show the clusters and overall coherence at each iteration step. For this activity, we will assume that the number of expected clusters is known, so leave the “Number of centroids” slider set to 3. Select the “Random centroids” button several times. The randomly selected centroids will be marked with a black “x”.
2. Are the randomly selected centroids necessarily distributed appropriately among the clusters? How do you anticipate this will impact performance? (i.e number of iterations and final assignment of points to clusters)
3. Could input from the user improve this step? Would this be possible with higher-dimensional data that might be difficult to visualize?
4. Now select the “Find clusters” button several times and observe the clusters and centroids evolve as the algorithm iterates. Do the locations of the centroids always follow the same path? Are the final clusters always the same?
5. Now, select new random centroids and find the clusters. Do this several times. Are the final clusters always the same? Does the algorithm always “correctly” identify the clusters? What strategies could be implemented to deal with the problem of local minima?
6. **Mean-shift algorithm**: a centroid-based method that assigns data points to clusters by shifting them toward the local modes, defined as the maxima of the density function.
7. *Section 1.3* of the script will build a kernel density estimation (KDE) using a selectable bandwidth and a gaussian kernel defined as:

where is the bandwidth and is the squared Euclidian distance. The resulting contours will be displayed on the plot of the points. Set the “Bandwidth” slider to 0.5 and select the “Show KDE contours” button. Now try setting the bandwidth to 5.0 and showing the contours. How are they different? How do you anticipate this will impact the final number of clusters?

1. Try different values for the bandwidth, select the “Mean shift” button, and observe as the points climb to a local KDE peak. How important is selecting the appropriate bandwidth to finding the “correct” number of clusters? (i.e. what type of range produces the correct number of clusters?)
2. **Density-based spatial clustering of applications with noise (DBSCAN) algorithm**: a density-based model that attempts to assign data points to clusters based upon densities present in the space.
3. *Section 1.4* of the script will cluster the data using the DBSCAN algorithm. The slider labeled “epsilon” assigns the parameter , which defines the Euclidian distance that comprises a point’s “neighborhood”. The slider labeled “Minimum points” assigns the parameter , which defines the minimum number of neighborhood points (i.e. the density) necessary to start or extend a cluster. Use the following parameters to get started:

|  |  |
| --- | --- |
| Epsilon: | 2.0 |
| Minimum points: | 4 |

Select the “DBSCAN” button. The “Animate” checkbox can be enabled to observe point assignment as the algorithm iterates through all the points.

1. Does the algorithm “correctly” find the clusters?
2. Are there some points that aren’t assigned to a cluster? How are these points labeled?
3. Leaving the “Epsilon” slider at 2.0, try different values for the “Minimum points” slider.
4. What happens to the number of clusters and noise when ?
5. What happens to the number of clusters and noise when ?
6. Reset the “Minimum points” slider to 4 and try different values for the “Epsilon” slider.
7. What happens to the number of clusters and noise when ?
8. What happens to the number of clusters and noise when ?
9. Based upon your observations above, how sensitive is this algorithm to parameter selection?

# Clustering GameHaven users using simulated data

*Estimated time: 20 min*

In this section, we will explore a real-world example to apply machine learning to building and comparing models.

### Application Background

GameHaven will be a social media platform enabling users to explore the board gaming hobby. GameHaven will track user interests and matching algorithms will determine new games that users are likely to enjoy and provide them with opportunities to find players and locations to play.

Games are described by many parameters that include primary categories and tags. Each game will be binned 1 through 5 in each of the following primary categories. Values for the game Chess will be provided as an example:

* Complexity (1, 2, 3, 4, 5) – Chess: 5, very complex
* Depth (1, 2, 3, 4, 5) – Chess: 5, very deep
* Speed (1, 2, 3, 4, 5) – Chess: 3, takes an average time to play a typical game
* Thematic (1, 2, 3, 4, 5) – Chess: 2, does not have a strong theme to it other than conflict
* Interaction (1, 2, 3, 4, 5) – Chess: 5, has very strong interaction with the opposing player
* Players (1, 2, 3, 4, 5) – Chess: 2, for 2 and only 2 players
* Mass (1, 2, 3, 4, 5) – Chess: 2, is a relatively small game
* Obscurity (1, 2, 3, 4, 5) – Chess: 1, one of the world’s most popular games

And tags are a list of attributes that a game either has or doesn’t: like dice, a space theme, drawing mechanics, bluffing etc.

Games are kept in a database that describe their vales for each of the categories and a list of applicable tags. Each time a player expresses “like” or “dislike” for a game, the users preferences associated with the attributes of the game are modified. Liked Chess? Your preference rating for the most complex category of games is increased.

While GameHaven is still in development, mock data has been generated to test the machine learning algorithms in the context of clustering users and matching games. It is believed that an unknown number of user clusters will develop. Example clusters might include groups of users that like: card games, party games, role-playing games, or “euro” games. To simulate these clusters, 4 patterns of users have been created as base preferences. The user profile data is a combination of the base pattern number, and variability is multiplied against the base. In other words, if the variability component were reduced to zero, the user data would be expressed by a rank-4 matrix.

**(please run section 2.1 of *Working with GameHaven Data.ipynb*)**

The of user preferences has been converted from .json placed into a taste profile matrix: users represent different rows, and each column represents a different preference. The first 40 columns are preferences for each of the 5 bins of eight categories, and the remaining 69 are preferences for the different game tags.

A lower rank model could be calculated offline and a low rank model could be used to provide the users a fast and responsive interface when interacting with the GameHaven platform. This project will look at two different mechanisms for creating a lower rank model of the user data. Both SVD and k-means will be used to create a model and the results will be compared.

**(please run section 2.2 of *Working with GameHaven Data.ipynb*)**

Section 2.2 runs a python SVD algorithm on the 500x109 user preference data. Using the results, rank(1) through rank (109) models of the data are compared to the original data. A plot is generated that shows the 2-Norm difference between each of the 109 different models and the original data.  
  
Questions:

What rank model would you recommend for this data and why?

It is expected that the real user data will have patterns, but not the four clear patterns that were used to generate the mock data. Would this change your answer in the previous question? How should the model rank be chosen if the results are a gradual continuum as the rank of the model is varied?

**(please run section 2.3 of *Working with GameHaven Data.ipynb*)**

If developing a model, it’s useful to check how well the model is fitting the data. Section 2.3 of the python code allows the comparison of specified users for different ranks of the model. Steps:

* Select ‘Model Rank’ and hit ‘Update Model’ button
* Select ‘User’ and hit ‘View User’ button
* Repeat as necessary

### Questions:

Please experiment with different values of model rank and some different users. In the context of the curve shown in section 2.2, how does the proximity of values in side-by-side model comparisons

**(please run section 2.4 and 2.5 of *Working with GameHaven Data.ipynb*)**

The k-means algorithm was employed to create an alternate low rank model. Again, all 109 model ranks were created and the 2-norm of the model vs. data are plotted as a function of rank.

### Questions:

Taking the SVD of the player data is computationally more complex than the k-means algorithm. Why does sweeping through 109 models take so much longer for k-means?

**(please run section 2.6 of *Working with GameHaven Data.ipynb*)**

A plot comparing the 2-norm of rank 1-109 models of both SVD and k-means are shown.  
  
Questions:  
Which technique creates a more accurate model of the data. Why?

The variable element of the mock data had to be greatly reduced to generate results that show strong results for a rank(4) model. What will this mean when GameHaven starts collecting data with a lot more noise?

# Matching Games to Users Using Collaborative Filtering

*Estimated time: 20 min*

For the second part of this problem, please open and run testp2.m in Matlab.

One method we will use to interact with the data is Collaborative Filtering. Collaborative Filtering is a method used to make recommendations to a user based on their preferences and tastes compared with a data set of many users. This method is helpful for GameHaven to recommend board games to their users based on their tastes.

The initial dataset of GameHaven users and their properties is created from a pattern + randomization method. This ensures that data will be predictable but random across the users. Similarly a matrix of game data is created with the same attributes as the users.

From the game and user attribute matrices, a random sparse ratings matrix is generated. The rating is based on how well the user attributes match the game attributes. Additionally to mimic a real world scenario the data is sparse so there are very few initial ratings.

The collaborative filtering system demonstrated here is done in the following way:

1: Cluster users based on their characteristics

First use SVD to determine a good rank for the user matrix, and use that value for the number of clusters.

2: Create similarity matrix between users in the same cluster

Collaborative filtering uses a similarity function. The goal of a similarity function is to measure how similarity between vectors **x** and **y.** In our example we are using cosine similarity.

A similarity matrix **S** can be constructed from **X** where  where .

3: User similarity matrix to fill in missing data in the ratings matrix

The ratings matrix maps the ratings of different games by users, so its size is the number of games by number of users. **R(I,j)** is the rating that user **i** has for game **j.** As mentioned before, most of **R** is zeros since most users do not have ratings for most games. To estimate the ratings for a user based on the given data, we do the following:

1. Extract the ratings of all the players in the same cluster as the given user **R1 = R(user in cluster 1).**
2. Fill in ratings matrix using 3 different collaborative filtering methods as follows:
   1. Unweighted cluster average
   2. Weighted cluster average
   3. Unweighted full average
   4. Weighted full average
3. Compare result, error and timing across 4 methods.

4: Sort ratings matrix and provide user top 10 recommended games

Questions:

1. What is the best number of clusters to represent this data set?
2. Implement the similarity\_matrix.m function in pseudocode
3. What is the highest recommended game for user # 15 using each different method?
4. What is the highest recommended game for user #134 in user\_t matrix using each different method?
5. Compare the timing of each method. Which method would be best for scaling?

# References

1. <https://scikit-learn.org/stable/modules/clustering.html>
2. <https://en.wikipedia.org/wiki/Cluster_analysis>
3. <https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68>
4. <https://www.geeksforgeeks.org/different-types-clustering-algorithm/>
5. <https://en.wikipedia.org/wiki/K-means_clustering>
6. <https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a>
7. <https://www.geeksforgeeks.org/k-means-clustering-introduction/>
8. <https://en.wikipedia.org/wiki/Mean_shift>
9. <https://spin.atomicobject.com/2015/05/26/mean-shift-clustering/>
10. <http://www.chioka.in/meanshift-algorithm-for-the-rest-of-us-python/>
11. <https://en.wikipedia.org/wiki/DBSCAN>
12. <https://towardsdatascience.com/dbscan-algorithm-complete-guide-and-application-with-python-scikit-learn-d690cbae4c5d>
13. <https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/>

# Appendix A