COSC 254 Final Project Report – Clustering

Group 11 (M. C. Gîrjău, S. Lin, M. Pitts, R. Toledo)

19 May 2021

Contents

1 Introduction

1.1 What is Clustering?

Clustering is the data mining task of finding meaningful "groups" in data. In other words, clustering aims to partition a set of data points into groups containing similar data points (where the concept of similarity may be defined in a variety of ways). Generally, clustering is an unsupervised task since the data doesn't usually come with pre-existing group labels – it is up to the clustering algorithm to discover any potential partitions. In fact, one may not even know the number of groups to be found, or even have any intuitive way of defining objective criteria for similarity.

1.2 Applications

Clustering has a variety of applications:

- **Exploratory data analysis:** Clustering is a very useful tool for data summarization, since it helps detect high-level patterns in the population of interest while abstracting away the finer details of the specific sample. As such, clustering is frequently employed as a first step in exploratory data analysis, and it helps inform the future development of statistical hypotheses.
- **Genomics and genetic disease associations:** Clustering is frequently used in the analysis of DNA sequences in order to find groups of genes associated with particular phenotypes (i.e., trait expressions). This is particularly useful when trying to detect groups of genes associated with the expression of some specific disease, such as esophageal cancer.
- **Market research:** Clustering is frequently used for market segmentation, wherein companies are looking for segments of customers with similar tendencies. This helps corporations fine-tune their marketing strategies and can be used when designing recommender systems with collaborative filtering.

1.3 Challenges

Like all ambitious data mining tasks, clustering suffers from a host of computational challenges. One such challenge is computational complexity, particularly since real-world datasets are frequently high-dimensional and very large in size. Because clustering is fundamentally very computationally-intensive, finding groups in such complex datasets might take prohibitively long amounts of time. Nevertheless, techniques like sampling or dimensionality reduction (if appropriate in that specific situation) can help keep runtimes relatively reasonable. On a related note, clustering algorithms may take so long to extract insights that real-life trends may change in the meantime and render any previously-found patterns obsolete.

Another main challenge is one of the fundamental problems in clustering algorithm design, namely the issue of finding the number of groups. While some algorithms (such as k-means) require this number to be known beforehand, this may not be the case in the vast majority of real-world applications. Various methods have been designed in order to help find a reasonable number of clusters (e.g., the elbow plot), but this challenge continues to remain relevant nonetheless.

2 Algorithms

2.1 k-Means

The k-means algorithm is one of the most popular clustering algorithms. It partitions a set of data points into a pre-specified number of clusters k (hence the name) using the Euclidean notion of distance. As such, each data point gets iteratively assigned to the cluster with the nearest "center" (i.e., the mean), after which the centers of the clusters get recalculated. This process continues until the algorithm converges, meaning that the cluster centers stop moving around within some tolerance level. The k-means algorithm requires the initialization of cluster centers in order to begin running, which is usually done at random, though other approaches have also been proposed.

2.2 k-Means++

The k-means++ algorithm resembles the classic k-means in every way except for the initialization step. Specifically, k-means++ attempts to improve the performance of k-means by switching from the random initialization of cluster centers (which allows for two centers to be right next to each other) to a more spread-out approach. The first center is chosen uniformly at random (and without replacement) from all the data points, after which the subsequent centers are chosen with probability proportional to their squared distance from the closest already-initialized center. This approach has been shown to yield considerable improvements to k-means both in terms of accuracy and runtime.

2.3 k-Medians

The k-medians algorithm once again follows the same steps as k-means, but it differs in terms of the distance metric used. While k-means uses the Euclidean distance for measuring the similarity between data points, k-medians uses the Manhattan distance, which leads the cluster centers to be at the median (as opposed to the mean) of the data points in a group.

2.4 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is a density-based, nonparametric clustering algorithm that groups together points that occur in high-density areas together. DBSCAN does not require one to specify the number of groups, and it can successfully identify clusters of any shape since it is based on density rather than distance. DBSCAN relies on the idea of ε-neighborhoods as a way to iteratively construct the clusters one by one, moving on to unclustered data points upon finishing one cluster, and then terminating when all density-connected clusters have been completely specified.

2.5 T-DBSCAN

T-DBSCAN (Chen et. al., 2014) is a variant of DBSCAN intended for GPS trajectory segmentation (i.e., for finding stops in travel trajectories). T-DBSCAN attempts to fix one main issue that DBSCAN exhibits when applied to the GPS trajectory problem, namely that of false positives in high-density areas with very different visit times. To address this, T-DBSCAN uses two ε-neighborhoods to detect stops, and introduces a temporal dimension to the clustering task. This enables the algorithm to detect high-density areas in time as well as space, which is exactly what stops are.

3 Methods

3.1 General-Purpose Algorithms

The k-means, k-means++, k-medians, and DBSCAN algorithms are considered general-purpose algorithms, meaning that they can be used to cluster data from a wide variety of application areas in a domain-independent way. Ideally, the 4 aforementioned algorithms should aim to perform well on a wide variety of cluster shapes, which would increase our confidence in their results when applied to real-world problems (where we may not even be able to visualize the data properly).

3.1.1 Datasets

We set out to assess the accuracy of our general-purpose clustering algorithms on a variety of cluster shapes that might arise in practice. Such an assessment would enable us to determine how flexible and robust these algorithms are with respect to different data shapes. To this end, we have leveraged the artificial datasets functionality of the `sklearn` library in Python. Specifically, we have made use of the following 6 datasets, each with a total number of 1500 data points shaken around with slight noise:

1. **Circles:** two concentric circles representing two different clusters
2. **Moons:** two moon-shaped curves representing two different clusters
3. **Loose blobs:** three blobs with different variances, and one blob spreading into the other two
4. **Elongated blobs:** three parallel, diagonally-elongated blobs representing three clusters
5. **Tight blobs:** three tight blobs representing three clusters
6. **Random noise:** no clusters, just uniformly-distributed noise

Figure 1 displays plots of these 6 datasets, all of which are two-dimensional and easy for the human eye to cluster by visual inspection. This makes it very easy for us to determine whether the 4 general-purpose algorithms perform the clustering task accurately or not.

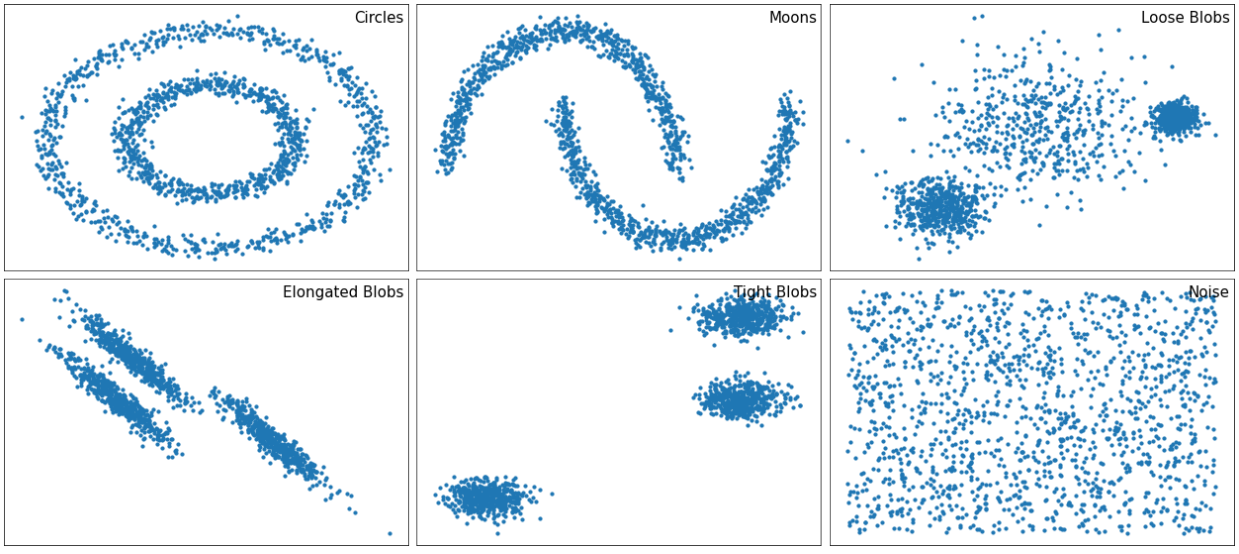


Figure 1: Artificial datasets used in our analysis of general-purpose algorithms

3.1.2 Parameters

Since k-means, k-means++, and k-medians are representative-based algorithms, they all require us to specify the number of clusters beforehand. When working with real-life data, we do not usually know the true number of clusters k, but many methods have been devised in order to determine the most probable value of k for a specific dataset. Luckily, our artificial datasets have the advantage that we can easily determine the true number of clusters simply by looking at their two-dimensional plots, eliminating the need for any additional preprocessing.

The circles and moons datasets have 2 clusters ($k = 2$), while all three blob varieties have 3 clusters ($k = 3$). On the other hand, the noise dataset does not have any distinguishable or meaningful clusters. In a way, this means that $k = 1$, but that renders the clustering task completely uninteresting and trivial despite its obvious accuracy. For the sake of demonstration only, we shall pick $k = 3$ in order to see how robust the representative-based algorithms are to the misspecification of the number of clusters.

Other than that, all other parameters for k-means, k-means++, and k-medians were kept at their default levels, with a centroid shift tolerance of 0.0001, and using at most 100 iterations.

By contrast, the DBSCAN algorithm does not require us to specify the number of clusters – it is able to determine that information implicitly using a density-based approach. Nonetheless, parameter tuning is still very important for DBSCAN, and in some ways even more difficult than for representative-based algorithms. DBSCAN requires us to specify an ε parameter that defines the high-density neighborhood radius around some specific point. Choosing ε cannot be done by inspection (which is how we managed to easily choose the values of k), and standardizing the data beforehand is essential in order to avoid dealing with measurement units. We chose a value of $\varepsilon = 0.3$ for the datasets with more separated clusters (circles, moons, blobs, and noise), and a value of $\varepsilon = 0.15$ for the datasets with slightly overlapping clusters (elongated blobs and loose blobs). Lower ε values prevent clusters that are close to each other from merging together, but it might also lead to the labeling of many fringe values as noise.

Further, DBSCAN also takes as a parameter the minimum number of points that forms a cluster. This parameter ensures that we do not detect small groups of noisy outliers as clusters in their own right, and its value might depend on the overall dataset size or on what is considered a meaningful cluster for some specific application. When it comes to our datasets, we have kept the minimum number of points at the default of 5, which seems reasonable upon inspecting the graphs in Figure 1.

3.2 T-DBSCAN

The T-DBSCAN algorithm is a specialized algorithm for finding stops in GPS trajectories (and labeling all other points as noise), so it cannot be compared with the other 4 general-purpose algorithms on the 6 cluster shapes above. T-DBSCAN takes its input formatted in a very specific way, as a time-ordered list of latitude-longitude pairs recorded at equally-spaced intervals. In terms of output, T-DBSCAN returns cluster labels that represent individual trajectory stops (e.g., pausing at a drive-through to get a coffee). This kind of information can be used to answer questions like "where do people stop?" and "why do people stop?", which are essential to traffic analysis, product marketing, or civil engineering.

3.2.1 Datasets

In order to experiment with our implementation of this highly-specialized algorithm, we decided to make use of real-life GPS data from bike rides in the Pioneer Valley. The valleybikeData R package contains trajectory

data collected at 5-second intervals during each trip, for a number of 147,944 bike rides between June 2018 and October 2020. This is exactly the kind of time-ordered coordinate input that T-DBSCAN requires, and it will enable us to carry out a practical analysis of where real people choose to stop on their bike rides.

We decided to look at 3 randomly-chosen route IDs, with lengths between one and three hours:

- `route_07_2018@40267493-aa94-48b1-8efa-d6971b32e15c` (July 2018)
- `route_09_2019@36f9aa1c-b63b-4b3c-80a1-0f00978db5f6` (September 2019)
- `route_10_2018@b71666ad-e83a-454a-a5c5-31cce8febbb9` (October 2018)

3.2.2 Parameters

The T-DBSCAN algorithm requires us to specify inner and outer ε-neighborhoods (in meters) that define respectively the high-density stop areas and the furthest error drift that can still be considered stationary. Once the trajectory steps outside of the outer ε-neighborhood, we assume that the bike is moving again. We have chosen the inner radius to be 10 meters, and the outer radius to be 40 meters, which hopefully accounts for small measurement errors in the bike GPS systems.

Similar to DBSCAN, the other parameter required by T-DBSCAN is the minimum number of points that forms a cluster. Since our data points are measured at 5-second intervals, this parameter basically translates to a minimum stop duration. In order to avoid uninteresting stops (e.g., at traffic lights), we have chosen the minimum stop duration to be 2 minutes, giving us a minimum number of 24 points per cluster.

4 Analysis

Figure 2 displays the results of our 4 general-purpose clustering algorithms on the 6 artificial datasets.

The clusters are displayed using different colors on the original dataset plots from Figure 1, and the median runtime of each algorithm over 5 independent runs is reported in the bottom right corner of each plot.

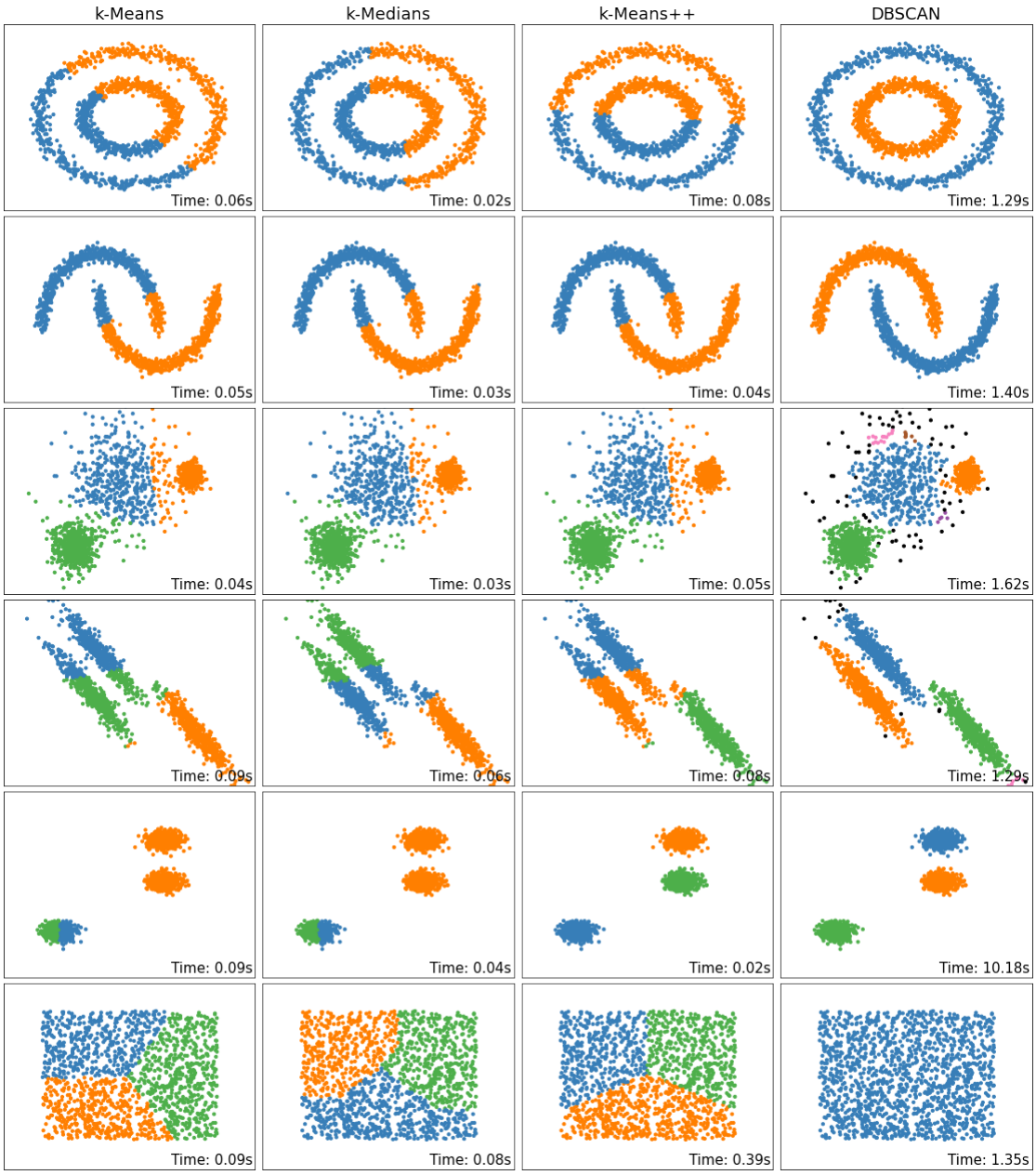


Figure 2: Clustering results for the 4 general-purpose algorithms under consideration

4.1 Accuracy

When it comes to the circles dataset, all 3 representative-based algorithms (i.e., k-means, k-means++, and k-medians) fail to identify the true clusters as the separate concentric circles. This is unsurprising, since all three algorithms are based on minimizing some measure of error (e.g., minimizing squared error for k-means and k-means++, and minimizing absolute error for k-medians). There is clearly no centroid position that would minimize distances to the outer circle without also falling close to the inner circle. To the human eye, the clusters seem to be identifiable not by measures of center, but by the agglomerative density of their points. As such, we would expect a density-based algorithm to perform better for this dataset. This is indeed the case, as DBSCAN manages to successfully identify the two concentric circles as separate clusters.

We encounter a similar situation when it comes to the moons dataset, which representative-based algorithms fail to cluster accurately, while DBSCAN succeeds perfectly. This is once again attributable to the fact that the moons are not identifiable as agglommerations of data points around a specific center, but rather as continuous areas of high density. Nonetheless, the accuracy of representative-based algorithms is much better for the moons dataset than for the circles dataset, since only the tip of each moon gets assigned to the wrong cluster.

Moving on, the k-means, k-means++, and k-medians algorithms do a reasonable job when applied to the loose blobs, through some points from the high-variance middle cluster get erroneously assigned to the tighter neighbors on either side. This illustrates yet another blunder of representative-based algorithms besides not being able to deal with elongated shapes – even for blob-like shapes, high variance in one or more of the clusters leads to fringe points being assigned to the wrong group. Even so, it is difficult to compare these results with that yielded by DBSCAN, since the latter ends up labeling a lot of the fringe points as noise. Depending on the application, we might prefer a slightly mixed output (as yielded by k-means, k-means++, and k-medians) over that of DBSCAN, particularly since we cannot extract much meaningful information from noisy data.

As we have seen for the circles and the moons, k-means, k-means++, and k-medians do not perform well when used on non-globular shapes. It is no surprise, then, that the 3 algorithms fail to identify the elongated blobs as separate clusters, while DBSCAN does so without a problem. As blobs begin to elongate, representative-based algorithms perform worse and worse, while the performance of DBSCAN maintains its reliability as long as the clusters don't become too sparse.

An interesting result can be noticed once we direct our attention to the tight blobs dataset. The k-means and k-medians algorithms split one of the blobs into two clusters despite its tight globular shape, and merge 2 other tight blobs into a single cluster. This is an unfortunate outcome caused by two of the blobs being closer together than they are to the third blob. Nevertheless, k-means++ manages to accurately identify the 3 blobs as separate clusters. This illustrates that the extra care taken by the k-means++ algorithm when initializing cluster centers does pay off in terms of accuracy – making sure that initial centers are well spread-out can significantly increase the reliability of our algorithm.

Finally, when it comes to the noisy dataset, the uniform density throughout the plane ensures that DBSCAN only identifies a single cluster. On the other hand, we have specified $k = 3$ for the representative-based algorithms in order to see what would happen. As a result, the noise has been split up into 3 equally-sized chunks. While these chunks do not constitute inherent groups in our data, they might still be useful in case we desire to artificially compartmentalize a population into sectors based on common characteristics.

4.2 Runtime

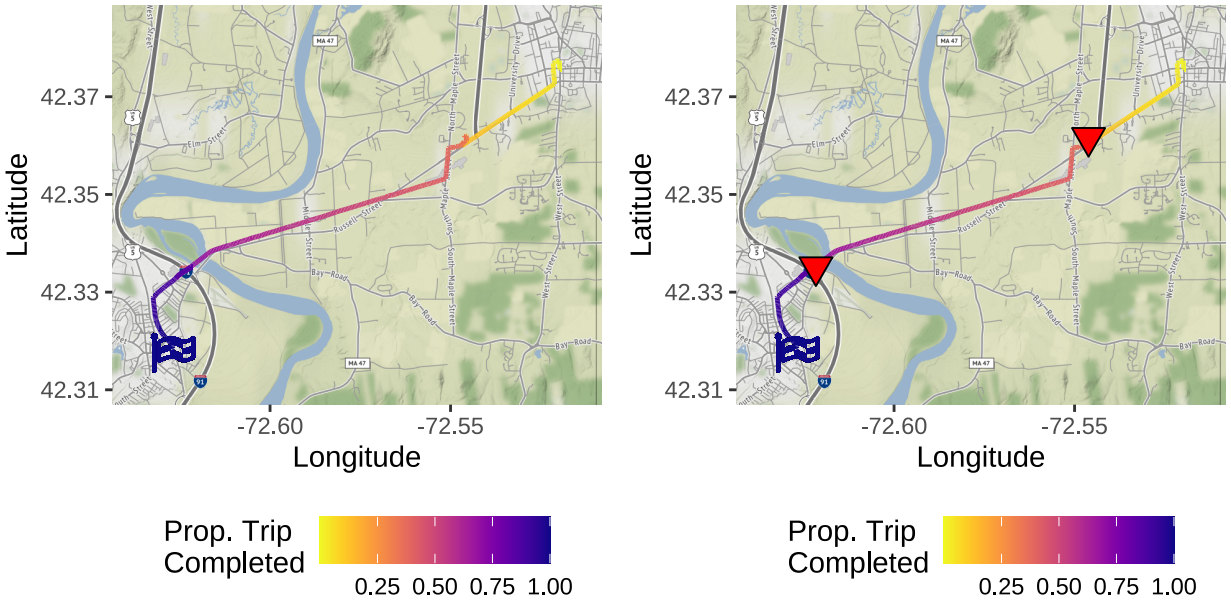
In terms of runtime, one thing we immediately notice is that the DBSCAN algorithm takes significantly longer to run than the representative-based algorithms. However, this might not be a fair comparison, since the DBSCAN algorithm is much more accurate than k-means, k-means++, and k-medians on a wide range of datasets. As such, it seems like there might be a trade-off between accuracy and runtime when comparing different classes of algorithms. Nevertheless, in the specific case where we are dealing with well-separated globular shapes, the representative-based approach is the clear winner in terms of runtime.

Diving deeper into representative-based algorithms, it is important to note that the runtime execution of k-means, k-medians, and even k-means++ might vary widely depending on the locations of the initial centers. If the centers are randomly initialized to be close to the true centers, the algorithm will take less time to converge. On the other hand, really badly-chosen initial centers will require more iterations to stabilize. The k-means++ algorithm attenuates this problem slightly by initializing centers within different areas of the data space, which avoids situations where all centers start out very close to each other.

5 T-DBSCAN Demonstration

5.1 First Route

We begin our exploration of the T-DBSCAN algorithm using a trip that took place on July 4th, 2018. This trip begins from the Amherst town hall and ends at the Northampton train station, lasting almost 3 hours. The plot below displays the trip trajectory on the local map, with the right-hand side including red triangle marks for every stop identified by the T-DBSCAN algorithm.



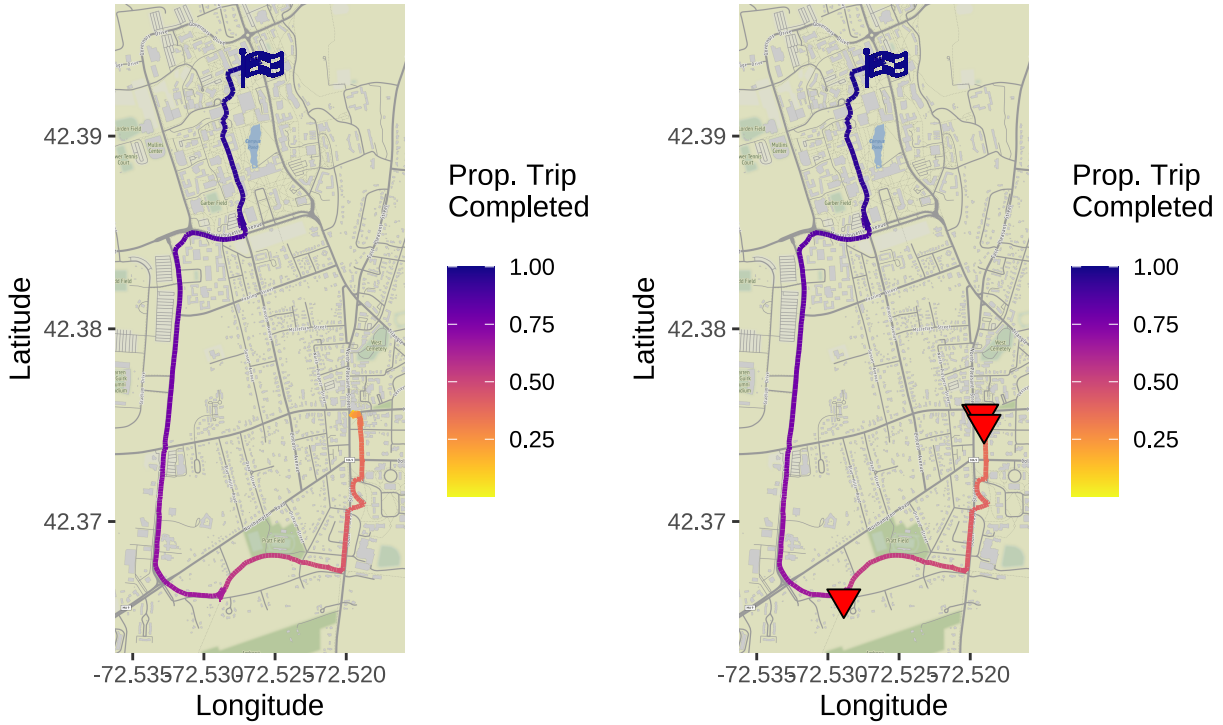
In this case, T-DBSCAN has found two stops. Upon zooming in with Google Maps on the first stop, we discover that it took place at a Dunkin Donuts on Route 9. This is a very plausible stop location, which seems to suggest that our algorithm has managed to successfully identify a real stop in a real GPS trajectory. Such information could potentially be used by franchises aiming to capture consumer patterns around Route 9 when choosing whether to establish a restaurant location on that road.

The second stop occurs at a large intersection right after the Connecticut river bridge between Amherst and Northampton. Once again, this is unsurprising given our previous knowledge of that location, which frequently suffers from traffic congestion. In contrast with the Dunkin Donuts stop, however, this second stop was probably not intentional. Rather, the biker might have stopped due to heavy traffic or they might have unmounted the bicycle in order to safely walk across the intersection. Either of these events would be recorded as a stop by T-DBSCAN, which might not be the kind of pattern we aim to find when looking for (ideally intentional) stops.

Finally, note that it is possible that other stops might have occurred at some point, but the T-DBSCAN algorithm did not succeed in detecting them. One might wonder what proportion of true stops T-DBSCAN actually managed to identify – completeness can be just as important as accuracy. Unfortunately, we have no way of quantifying the recall ability of this T-DBSCAN without making use of pre-labeled data. One might collect such information by personally going on a bike ride and performing a variety of meaningful and less-meaningful stops, which can then be used to better understand the power of T-DBSCAN.

5.2 Second Route

Moving on, we direct our attention to a trip that occurred on September 1st, 2019. It began once again from the Amherst town hall, ending at UMass Knowlton and lasting a little over an hour. As for the first route, the plot below displays the trip trajectory on the left-hand side, with added markers on the right-hand size for every stop identified by T-DBSCAN.

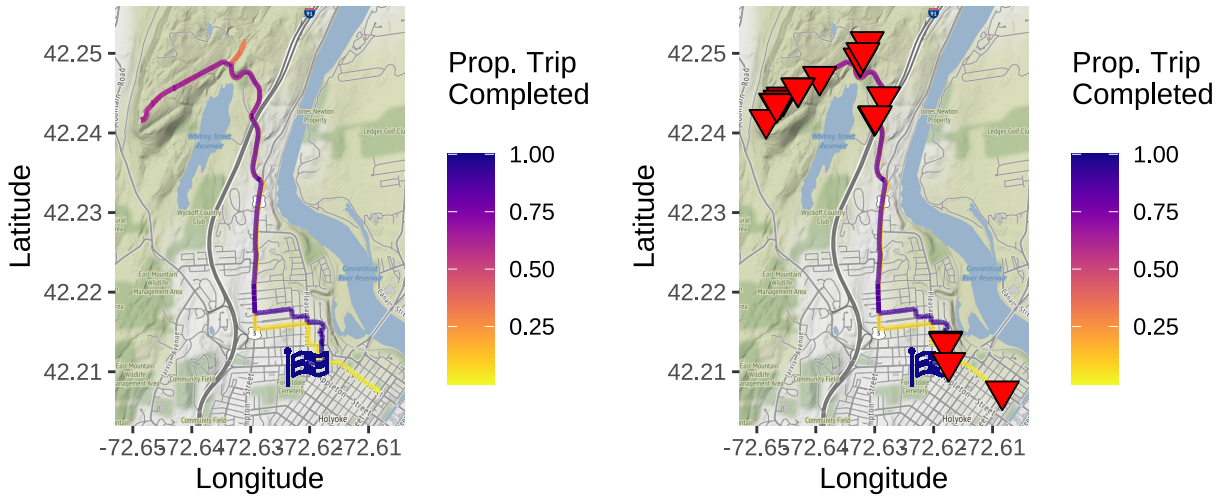


For this route, the algorithm identifies a bunch of stops at the very beginning of the trip. These stops are probably not meaningful, and they are most likely caused by the biker taking some time to get ready for the ride. This suggests a possible improvement to the T-DBSCAN algorithm, namely disregarding stops that occur too close to the beginning or end of a trajectory.

T-DBSCAN also identified another stop somewhere in the middle of the trip. Zooming in on the map, we find that the biker took the Norwottuck nature trail until reaching an intersection with a main road, which is where the stop occurred. Once again, we might have stumbled upon an unintentional stop caused by traffic as the biker was waiting to turn into the road from the bike path.

5.3 Third Route

Finally, we take a look at a trip from October 14th, 2018, which began from downtown Holyoke and ended at Pleasant station. This trip lasted almost two and a half hours. The plot below displays the plain trip on the left-hand side, and the right-hand side includes markers for the identified stops.



Within this trip, T-DBSCAN identifies a large number of stops (18 to be precise). Some of these occur at the very beginning of the trip or at highway intersections, which are not particularly interesting instances, but most stops seem to occur in the northern part of the trajectory. Interestingly, that area is full of conservation parks and nature trails. We can think of a variety of reasons why someone would stop so frequently when making a trip through nature – admiring the view, taking photographs, or even resting after an uphill slope.

While T-DBSCAN can help us identify the most common stop locations within a GPS trajectory, the interpretation and justification of these stops remains incredibly difficult. To ease this task of pruning uninteresting stops, we might think of potential extensions to the algorithm, such as discarding stops that occur in irrelevant locations, utilizing traffic data from that specific timeframe in order to quantify the effect of traffic jams, or attempting to find stops that occur in a large proportion of trips.

6 Conclusions

In this project, we have implemented 5 clustering algorithms – 4 general-purpose ones (k-Means, k-Means++, k-Medians, DBSCAN) and a specialized one (T-DBSCAN).

Through an accuracy analysis on 6 artificially-generated dataset shapes, we have found that DBSCAN performs much better across the board than all three representative-based algorithms. Nevertheless, this high degree of accuracy comes with much higher runtimes. This caveat might not make the DBSCAN algorithm very scalable to high-dimensional datasets with a lot of observations. Still, big data is not necessarily a problem, since we could make use of sampling or dimensionality reduction methods instead of using the entire dataset.

Further, we have seen that the k-means++ algorithm provides a healthy improvement over k-means and k-medians, as it ensures the cluster centers are initialized in a spread-out way. This can help avoid situations where representative-based algorithms converge erroneously due to poorly-initialized centers, and it provides an additional degree of confidence in the results.

Finally, in our exploration of the specialized T-DBSCAN algorithm, we have found that it is well-suited for identifying meaningful stops in trajectories that help inform us about certain events on a trip (e.g., stopping to get food or stopping to admire the view). However, T-DBSCAN is also prone to identifying unintentional stops that do not carry any interesting information about the trip, such as unusually long waiting times in a traffic jam. As a last limitation, it is impossible to tell how good T-DBSCAN is at identifying stops without knowing all of the stops in advance. For such an analysis, one would need to work with data where stops are already labeled, which might help us estimate the recall precision of the algorithm for a wide variety of stop types.