# Infection Clustering: A Proximity Based Unsupervised Machine Learning Algorithm

Collin A. Coil[*]

December 9, 2023

**Abstract**

Infection clustering is an unsupervised machine learning technique for data clustering. The goal is to simulate the behavior of an infectious disease passing through a population. In this case, the data points are the population of interest, and the clusters are the infection. Points get exposed to an infection based on their proximity to infected points, and the infected points have a chance to recover. This mechanism of iteratively infecting and uninfecting the data allows the clustering algorithm to effectively cluster data. Its performance was verified on several toy data sets and was compared to other commonly used clustering algorithms.

Code for this program can be accessed at https://github.com/CollinCoil/Infection-Clustering/tree/main

[*]American University, Center for Data Science, Washington, DC, USA. `cc6121a@american.edu`.

## 1 Background & History of the Problem

Most tasks in machine learning can be divided into two categories: supervised and unsupervised learning. In supervised tasks, the machine learning model is working with labeled data. The model uses those labels as a ground truth that it can use to iteratively improve the results. In unsupervised tasks, there are no labels for the data, so machines are required to discover patterns without explicit guidance.

One of the most widely discussed tasks in unsupervised machine learning is clustering. In clustering, the goal is to use an algorithm to group similar data points together. Researchers have developed numerous different clustering algorithms. Madhulatha (2012) provide an overview of several different types of clustering techniques. The first they describe is hierarchical clustering, in which each point starts as its own clusters, and clusters are progressively merged based on their proximity. Next, they describe partitional clustering where data initially assigned to groups, then iteratively reallocated to the groups until convergence. The $k$-means algorithm is one example of a partitional clustering method. Next, the authors describe density-based methods and provide DBSCAN as an example. Grid-based clustering is the next type of algorithm proposed, but this is an uncommon technique. Finally, the authors describe model-based clustering, which is common via the use of Gaussian mixture models.

Clustering algorithms have been used for a wide variety of applications. $k$-means clustering has been used for analyzing energy consumption behavior in mixed-used buildings (Culaba et al. 2020) and assessing student performance (Vankayalapati et al. 2021). Hierarchical clustering techniques have been deployed to forecast sales (Chen and Lu 2017). DBSCAN, a density-based clustering algorithm, was used to analyze seismic data to identify the distribution of earthquakes (Karmenova et al. 2022). The ubiquity of clustering techniques can be seen in the wide variety of applications in this non-exhaustive list.

Although there are many clustering algorithms that have been developed, there is not one algorithm that is clearly best for every application. Therefore, new approaches are constantly being developed. This paper contains the description and analysis of the effectiveness of one new approach, which I name "infection clustering."

The infection clustering algorithm is a proximity-based algorithm, assigning points that are near each other to the same cluster. The basic idea of this program is that of a pandemic. An infectious disease passes through a population, and there is some probability that a person exposed to a disease contracts it. Then, that person can pass on the disease or recover. The disease spreads rapidly in high-density areas but sometimes fizzles out in low-density areas. The infection clustering algorithm is based on this same principle; however, the data points are the ones contracting the cluster "disease" and passing it to other data points in close proximity.

## 2 The Infection Clustering Algorithm

This program begins by taking a data frame, and it makes every data point in the frame into a data point object. Next, it calculates the pairwise Euclidean distances between every data point and stores it as a matrix. This calculation is one of the most time-consuming parts of the code. Then, the program uses this matrix to determine the closest data points to each. These points are considered to be in range for infecting that data point. The last initialization step is to initialize the infected points. It does this through k-means ++ initialization, first described in Ball and Hall (1967). This algorithm randomly chooses one data point, then chooses the point farthest away from that first point to be the initial point of the next cluster. It continues choosing the points that are farthest from the existing points until it has initialized the number of infections input by the user.

After these initialization steps, the infection clustering algorithms begins a for loop. In this loop, it iterates through the uninfected points and assesses if it should be infected. If there is an infected point in the sphere of influence of the uninfected point and a uniform randomly generated number is less than the probability of infection, then that point gets infected with that cluster value. If a point has several exposures, the "strongest" infection, or the one where the most infected data points pass the infection to the uninfected point, wins. To ensure that infecting a point doesn't lead to a chain where it infects other points in one iteration, the actual "infection" step for all points happens after the algorithm has determined whether or not to infect each uninfected point. Next, the centers of each clustered infection is calculated.

After that step, some of the points "recover" from their infection and lose their cluster. For each point, a uniform random number is generated, and if that is below the recovery threshold, the point drops its cluster. This mechanism serves two purposes. First, it allows two different infections that occupy the same data cluster to compete with each other. As some points recover, they can be infected by the other cluster, and one cluster will end up being eradicated in an area. If this happens, an uninfected point is randomly selected as the new initialization point. This allows the algorithm to shift each infection cluster to a true unique cluster. Second, it can allow the infection clusters to move to the highest-density areas. This is especially true if we determine the points within a point's sphere of influence through drawing $n$-balls, as discussed soon in section 3.1. Some points could recover and not end up with any infection, signaling that they are anomalous points.

Once the infection and uninfection loop is finished, the algorithm takes a few more infection steps. This is necessary to reinfect those points within a true cluster that have lost their cluster assignment in the final uninfection. Then, it concludes by constructing a numpy array with the initial data points and a new column containing the cluster assignments. Outliers get assigned to a cluster with value -1.

## 3 Techniques from TSC

I used several techniques from TSC for this project. The two discussed below are the most significant.

### 3.1 Expected Number of Points in $n$-ball Contained within Hypercube

Because this algorithm is a proximity-based clustering algorithm, we need to determine which points are close to each individual points. My first attempt was to construct an $n$-ball around each point, where $n$ is the number of dimensions of the data. The goal was to construct the $n$-ball so that the expected number of points inside this ball was constant supplied by the user as a hyperparameter. This approach was inspired by the Monte Carlo technique we used to approximate $\pi$ in TSC programming assignment 1.

First, let the data be contained in a hypercube of dimension $n$, and construct an $n$-ball within the hypercube. This $n$-ball does not need to be the biggest $n$-ball that can fit inside of the hypercube, but rather the $n$-ball with size such that it contains the expected number of points. We know that the volume of the hypercube is

$$V_{n-\text{cube}} = \ell^n,$$

where $\ell$ is the side length. We also know that the volume of the $n$-ball is

$$V_{n-\text{ball}} = \frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)} r^n,$$

where $r$ is the radius of the $n$-ball. Thus, given the range of the data (which is the side length of the hypercube), the dimension of the data, and the expected number of points inside of the $n$-ball, we want to determine the requisite radius.

Since the expected number ($e$) of points inside the ball is the number of points ($p$) inside the hypercube multiplied by the ratio of the volumes, we find

$$\begin{aligned} e &= p \cdot \frac{V_{n-\text{ball}}}{V_{n-\text{cube}}} \\ &= p \cdot \frac{\frac{\pi^{n/2}}{\Gamma(\frac{n}{2}+1)} r^n}{\ell^n} \\ &= \frac{p r^n \pi^{n/2}}{\ell^n \Gamma(\frac{n}{2}+1)}. \end{aligned}$$

Then,

$$\begin{aligned} r^n &= \frac{e \ell^n \Gamma(\frac{n}{2}+1)}{p \pi^{n/2}} \\ r &= \sqrt[n]{\frac{e \ell^n \Gamma(\frac{n}{2}+1)}{p \pi^{n/2}}}. \end{aligned} \tag{1}$$

However, when using this formula to create $n$-balls around data points, the average number of points contained inside of the $n$-ball differed from the user-input expected value. This difference from the expected increased as $n$ increased. At this point, I realized a fundamental error in my assumption: I assumed that the $n$-ball would be contained entirely within the hypercube. However, when I was constructing $n$-balls around my data points, these were not entirely contained within the hypercube that delimited the range of the data. Furthermore, I realized why the average number of points within the $n$-ball decreased with dimension. This can be easily shown through the following propositions.

**Proposition 1.** *The probability of a point being within $\varepsilon$ of the edge of a hypercube goes to 1 as $n \to \infty$.*

*Proof.* Let $\varepsilon > 0$ be given. Construct an $n$-dimension hypercube with side length $\ell$, called $A$. Then the volume of $A$ is $\ell^n$. Construct an $n$-dimension hypercube inside of A with side length $\ell - 2\varepsilon$, called $B$. Clearly, there will be a band with width $\varepsilon$ between the sides of $A$ and $B$. Then, the probability of a point being within $\varepsilon$ of the edge of $A$ is

$$\begin{aligned} P &= 1 - P(\text{point in B}) \\ &= 1 - \frac{(\ell - 2\varepsilon)^n}{\ell^n} \\ &= 1 - \left(1 - \frac{2\varepsilon}{\ell}\right)^n. \end{aligned}$$

Finally, $\lim_{n\to\infty} 1 - \left(1 - \frac{2\varepsilon}{\ell}\right)^n = 1.$ □

At the extreme, this implies that all data points lie on the boundary of a hypercube given sufficiently large dimension.

**Proposition 2.** *The probability of an $n$-ball centered at a random point in a $n$-dimensional hypercube being contained entirely within the hypercube goes to 0 as $n \to \infty$.*

*Proof.* Follows trivially from previous. □

Thus, the average number of points within the $n$-ball should decrease as $n$ increases because the points are all contained within the hypercube.

This leads to an alternative interpretation of the curse of dimensionality. As the number of dimensions increases, the data become more sparse in that space. The data will all be close to the edge of the hypercube containing the data, implying that meaningful patterns in the data will be lost.

To get around this issue, I instead calculated distances between each point in the data. For each point, I selected the $e$ closest points, ensuring that no matter how many dimensions, there will always be a constant number of points that are used for the infection process. This process was also significantly easier to calculate. The downside of this deterministic approach is that anomaly detection using infection clustering is more difficult. If a user wants to use infection clustering for anomaly detection instead of for data clustering, the original $n$ ball code can be substituted in for the deterministic calculation of the $e$ closest points; however, users should be aware of the potential limitations of this approach in high dimension data.

## 3.2 Algorithm Analysis

Although we did not discuss time complexity in class, there are notes about it on Canvas. Therefore, I wanted to perform an analysis of the complexity of this algorithm. As I hinted to in the previous section, calculating the $e$ closest points to each point in the data set was easier to compute than the number of points being contained within an $n$-ball.

I wanted to test the time-complexity of this algorithm, so I conducted an experiment. For a fixed dimension, I timed how long the algorithm took to cluster 1000 data points. Then, I increased the number of data points by 1000, repeating the trial. I continued this until I had calculated the time necessary to cluster 30,000 data points. The graph illustrating the results is shown in figure 1.

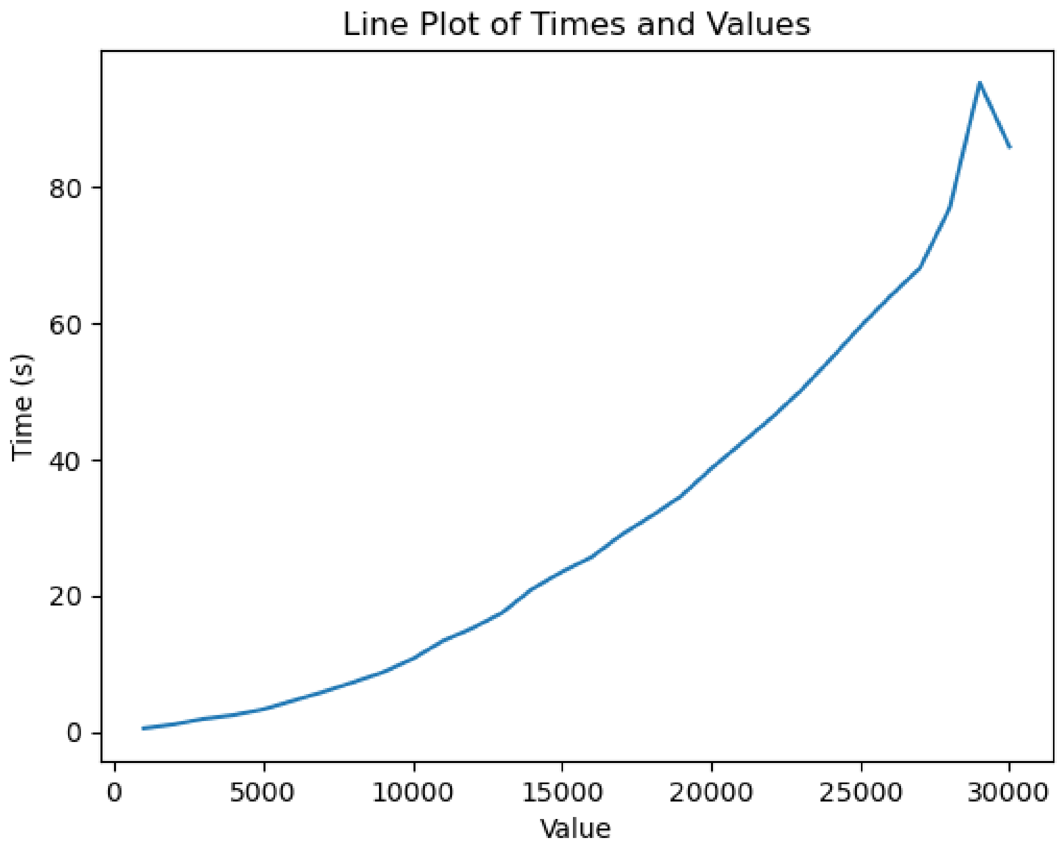


**Figure 1.** Clustering Time for Increasing Number of Points

As we can see, this algorithm appears to have $O(p^2)$ time complexity with regard to the number of data points $p$. This is confirmed by fitting a quadratic extremely closely to this data.

Additionally, I assessed the time complexity of this algorithm with regard to the number of dimensions. I randomly generated 20,000 data points and ran the algorithm on data sets of dimension 2 through 16. The times necessary to cluster these data are presented in figure 2.

As we can see, there is some random variation. However, it appears that there is a linear trend line, suggesting that the time complexity of this algorithm is $O(n)$, where $n$ is the dimension. Overall, the experimental time complexity of this algorithm appears to be $O(n \cdot p^2)$, where $n$ is the dimension and $p$ is the number of data points.

Finally, I compare my algorithm's performance to other standard clustering algorithms. Figure 3 below shows the performance of clustering algorithms in several toy examples derived from Sklearn's clustering module.

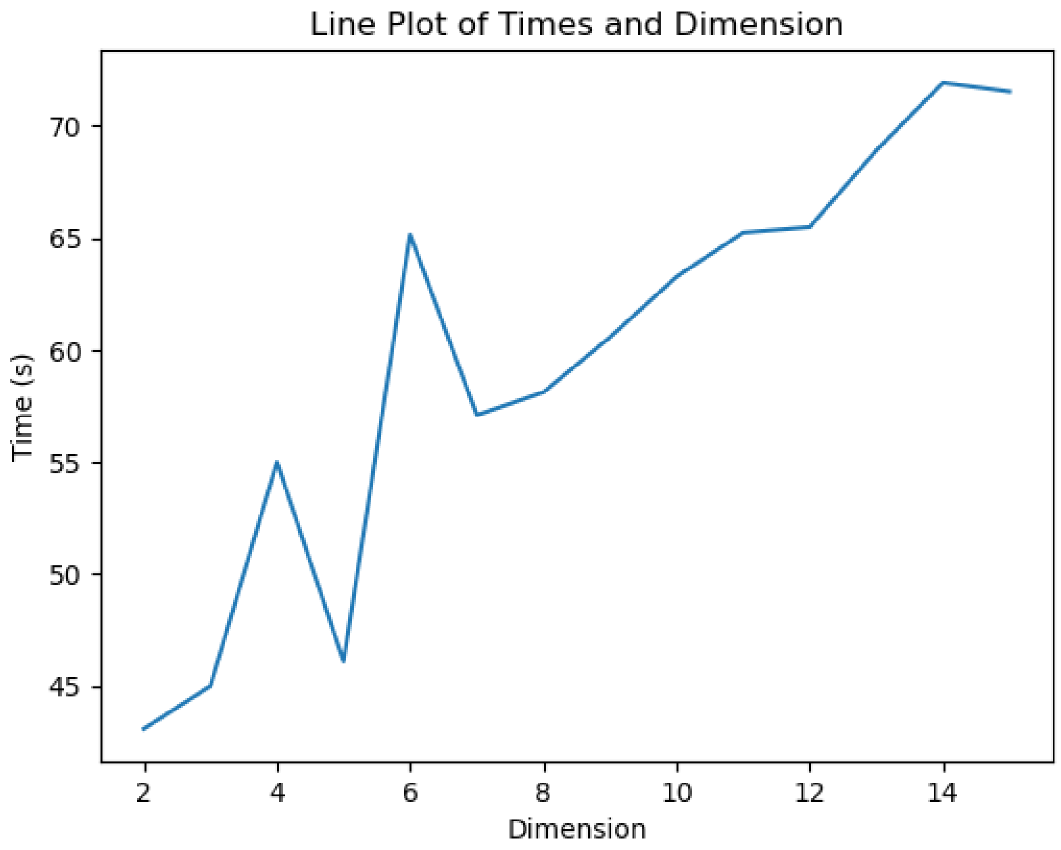


**Figure 2.** Clustering Time for Increasing Number of Dimension

Appendix 1 contains the results of infection clustering on these toy data sets. As we can see, infection clustering has very good performance on these data sets. It assigns clusters that are close approximations of the true clusters that were generated. These true clusters can be easily determined by observation. Only the spectral clustering algorithm had superior performance to infection clustering.

However, my infection clustering algorithm calculated the clusters much faster. Infection clustering produced results in 0.19 to 0.45 seconds. Spectral clustering produced results in 0.16 to 0.91 seconds. The speed of infection clustering may be increased by decreasing the maximum number of iterations, increasing the probability of infection, or decreasing the probability of recovery.

## 4 Conclusion

This article provided the history of machine learning clustering algorithms and proposed a new proximity-based algorithm. The development of this algorithm and the mathematics during it can be traced back to concepts learned in several classes, including Tools of Scientific Computing, Real Analysis, and Measure Theory.

Infection clustering was tested on several data sets. The first round of tests was to

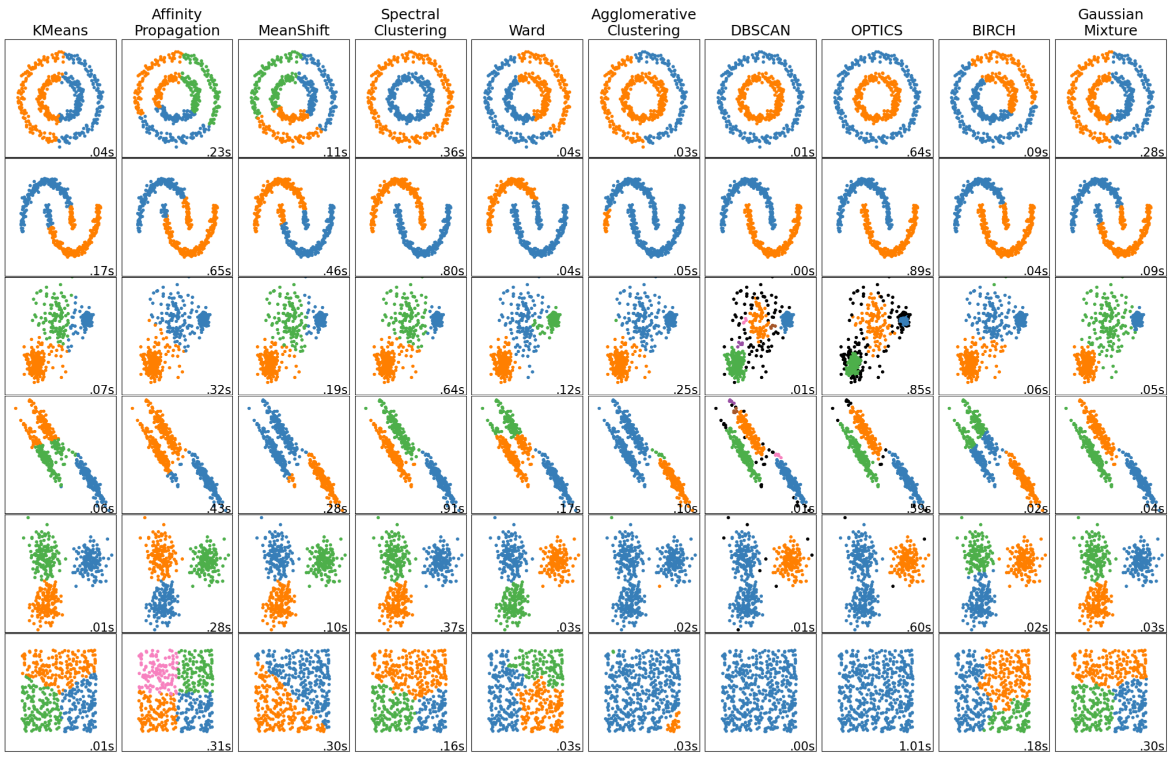


**Figure 3.** Performance of Common Algorithms on Toy Data

experimentally estimate the time complexity of the algorithm with regard to the number of data points and the dimension of the data. The second was to assess the performance of infection clustering on several toy data sets. Infection clustering had similar performance to the highest performing clustering technique at a mere fraction of the time necessary. Therefore, it is a useful new machine learning technique for clustering data.

## References

Ball, Geoffrey H and David J Hall. 1967. “A clustering technique for summarizing multivariate data.” *Behavioral science* 12(2):153–155.

Chen, I-Fei and Chi-Jie Lu. 2017. “Sales forecasting by combining clustering and machine-learning techniques for computer retailing.” *Neural Computing and Applications* 28:2633–2647.

Culaba, Alvin B, Aaron Jules R Del Rosario, Aristotle T Ubando and Jo-Shu Chang. 2020. “Machine learning-based energy consumption clustering and forecasting for mixed-use buildings.” *International Journal of Energy Research* 44(12):9659–9673.

Karmenova, Markhaba, Aizhan Tlebaldinova, Iurii Krak, Natalya Denissova, Galina Popova, Zheniskul Zhantassova and G Györök. 2022. “An approach for clustering of seismic events using unsupervised machine learning.” *Acta Polytechnica Hungarica* 19(5):7–22.

Madhulatha, T Soni. 2012. "An overview on clustering methods." *arXiv preprint arXiv:1205.1117* .

Vankayalapati, Revathi, Kalyani Balaso Ghutugade, Rekha Vannapuram and Bejjanki Pooja Sree Prasanna. 2021. "K-Means Algorithm for Clustering of Learners Performance Levels Using Machine Learning Techniques." *Revue d'Intelligence Artificielle* 35(1).

## Appendix

Below are graphics produced through infection clustering using the toy data sets.

**Figure 4.** Sklearn Circles Data Set

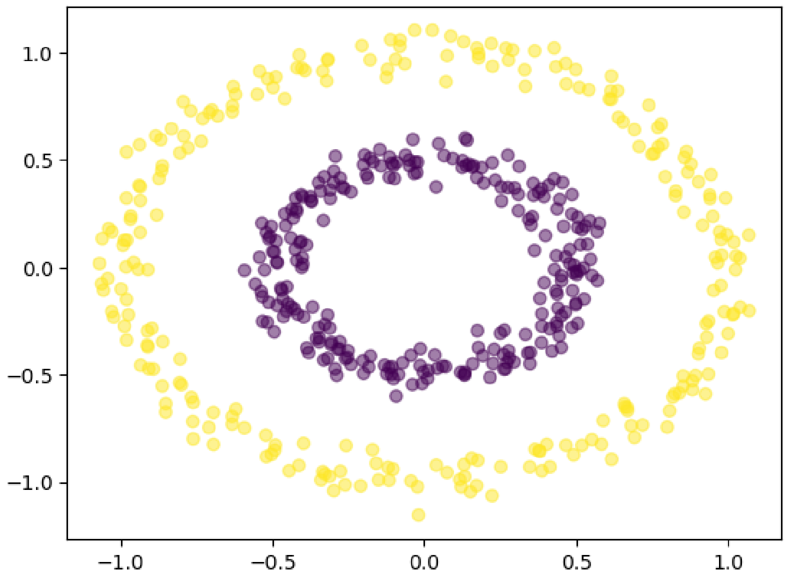


**Figure 5.** Sklearn Moons Data Set

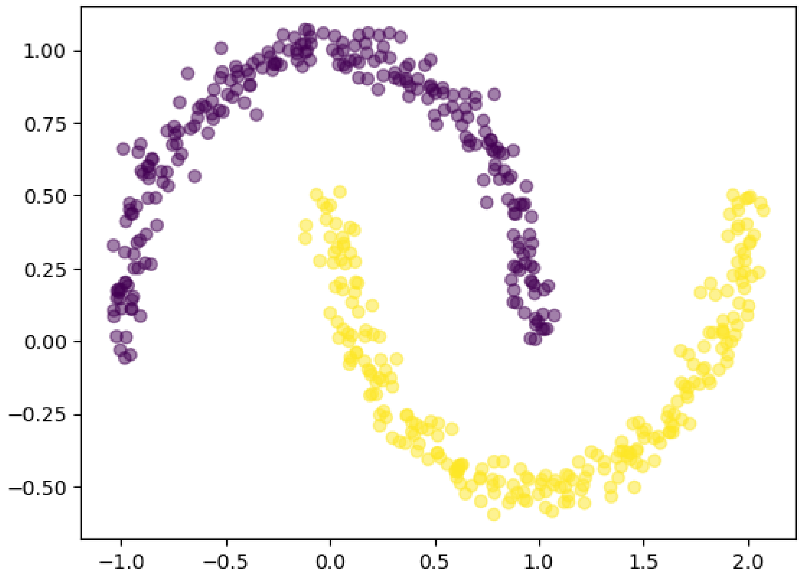


**Figure 6.** Sklearn Varied Data Set

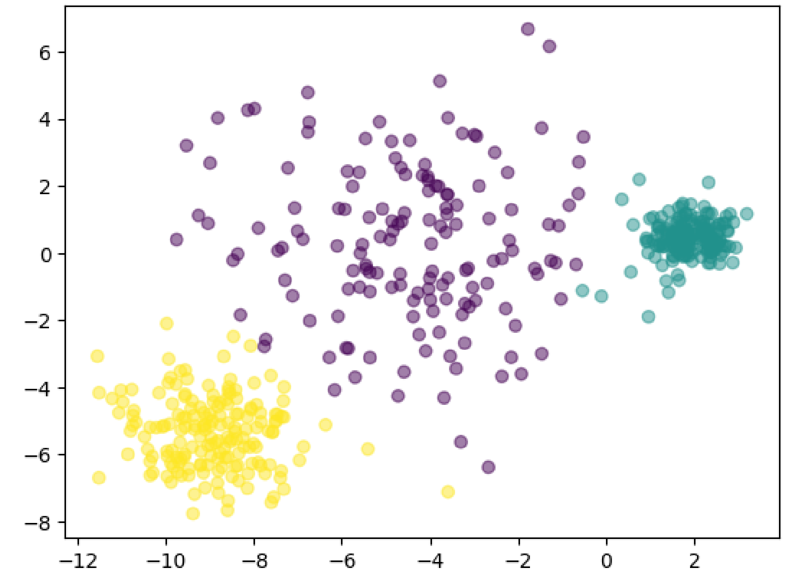


**Figure 7.** Sklearn Aniso Data Set

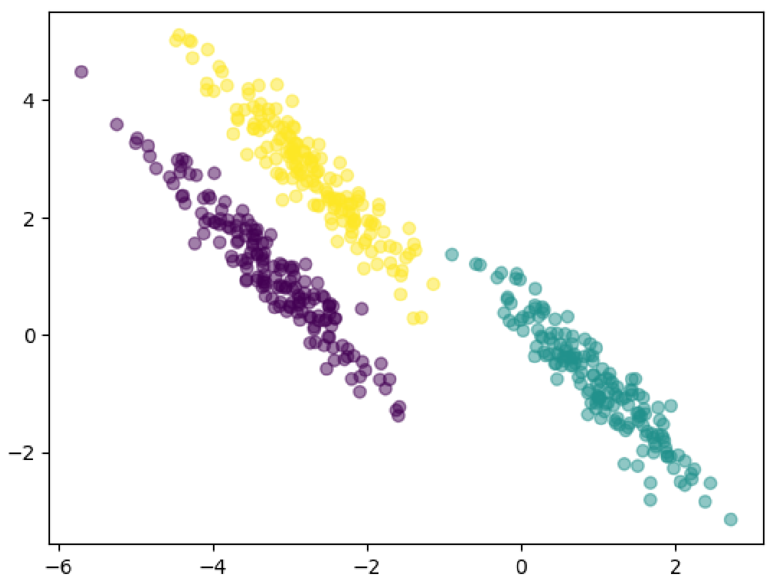


**Figure 8.** Sklearn Blobs Data Set

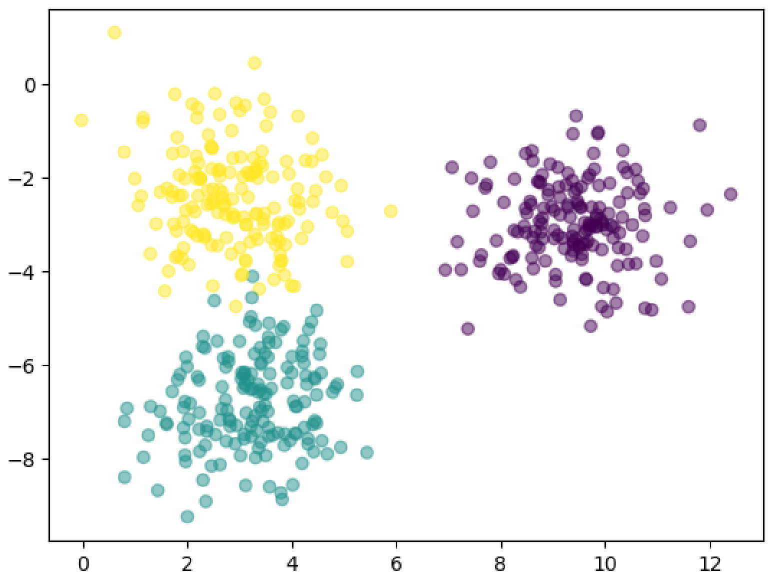
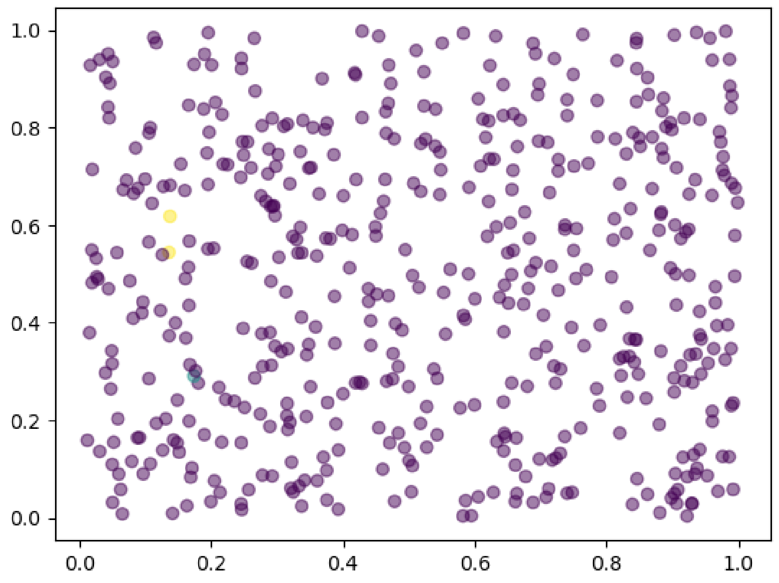


**Figure 9.** Sklearn No Strucutre Data Set



0.0
0.2
0.4
0.6
0.8
1.0