Comparative Analysis of Web Implemented Local Mesh Algorithm (WILMA) as an Unsupervised Learning Algorithm Against KMeans and DBSCAN

Executable Code

Abstract

Web Implemented Local Mesh Algorithm (WILMA), was initially inspired by the KNN algorithm due to its simplicity and ability to gain fairly accurate classification results. We wanted to expand this idea to an unsupervised clustering algorithm. After studying KMeans, we wanted to work on solving its core issue; nonlinear cluster divides. Furthermore, WILMA can work without the user needing to choose an exact value for K and can optimize its clusters until the groups are good. WILMA is implemented using graphs. Each instance is connected to $\boldsymbol{N}$ (an arbitrarily chosen number discussed later) nearest neighbors to form a list of graphs, where each separate graph represents a different cluster. We can then 'prune' the incorrect edges using a variety of techniques until each graph contains a decent cluster. Comparative performance of WILMA against novel unsupervised clustering algorithms (KMeans and DBSCAN) determined that WILMA was able to outperform these existing algorithms in both convenience and accuracy.

Introduction

Our algorithm takes in an unlabeled dataset of any dimension with continuous data and returns a method of grouping the data in the form of clusters. These unlabeled datasets are known as "unsupervised" datasets, which are collections of data that have no known classification. Without a classification, we cannot extrapolate much from the information other than summary statistics, and there have been many algorithms that attempt to take unsupervised data and group them. However, many of these algorithms are full of imperfections. Some algorithms need preset specifications, like the number of groups, which may not be known. Other algorithms face difficulties with handling high-dimensional data and data that is split in complex patterns (varying densities, for example). It's important to create/find a powerful unsupervised learning algorithm, as unsupervised learning can help solve a number of problems that we know very little about. DNA pattern analysis, for example, is very difficult to do despite having lots of data on DNA. DNA datasets, a majority of which do not have classified labels, are difficult to decode and understand. Unsupervised learning can help decipher the patterns and clusters present in these datasets, which can help us better understand underlying genetic patterns.

To test the performance of various unsupervised learning algorithms, ours included, we are using two cartesian-datasets: make_moons and make_circles (provided by scikit-learn). These datasets will be run against three unsupervised algorithms—KMeans, DBSCAN, and WILMA—which will produce a numpy matrix of points with their classifications. We will compare these matrices with the predetermined classification groups established by the datasets in order to determine which algorithm is the most accurate. Alongside accuracy, we will also note the time of the algorithms.

Related work

There is a lot of work that aims to improve existing unsupervised algorithms. There are many simple algorithms such as KMeans which intentionally have little complexity and can be used for very generic and low-demanding tasks. These algorithms are not commonly used with highly complex datasets (Dabbura, 2018). We resort to algorithms that are significantly more complex, like DBSCAN. It is regarded as a state-of-the-art unsupervised learning algorithm which has high performance in clustering data that has non-linear boundaries, a feat that KMeans cannot accomplish. DBSCAN, however, is not perfect. It is highly parameter-sensitive, requiring fine-tuned parameters that are difficult to determine beforehand, and is poor with handling high-dimensional data (Schubert et al., 2017). A number of papers have aimed to reduce the pitfalls of DBSCAN while also addressing the weaknesses of simpler algorithms like KMeans, which will be explored below.

In a study conducted by Oppel and Fischer in 2020, a novel unsupervised learning algorithm was used to classify rainfall. This algorithm utilized “dimensionless mass curves” which creates a curve for each point, dependent on their time and cumulative precipitation. These curves are then grouped via cluster analyses, which indirectly cluster-group the instances in the dataset. The study found that DBSCAN, which classified 96% of the data as noise and created fewer clusters than necessary.

Another study done by Makin and others (2018) used latent dynamics to decode arm movement from neurological recordings. The readings from neurological records are very high in dimension and aren’t linearly split, so the algorithm that was tested used a filter called ‘recurrent exponential-family harmonium’, or rEFH, which would allow for nonlinear dynamics to be better observed. KMeans, which lacks non-linear separation, and DBSCAN, which cannot handle high-dimension data, struggle with the data in this study.

Dataset and Features

We are using two tools provided by scikit-learn to analyze the performance of our algorithms: make_moons and make_circles. Make_moons is a tool that creates two distinct curves that are not linearly separable. This makes unsupervised algorithms that group via linear boundaries unable to properly separate the moons. Make_circles is another tool that creates two concentric circles that are non-linearly separable and are varied in density. It’s another useful visualization tool to see how effective an algorithm is at separating groups without linear boundaries and dealing with varied densities. These tools use various parameters—such as number of dots, amount of disturbance, and distance between each other—to generate graphs. In our tests, we made the number of dots in each dataset equal to 200 points, and set the noise/disturbance to 0.1, which is fairly spread apart (“Sklearn.cluster.DBSCAN”, 2022) (“Sklearn.cluster.Kmeans”, 2022).

Once using these tools we had our datasets, generated in the form of numpy arrays. These arrays hold collections of coordinates with two features: x and y. There is also an auxiliary numpy array which has a numerical value assigned to each coordinate, which tells us which cluster a dot belongs to, which we will compare to the results of the algorithms. Visualizations of the datasets can be seen in the ‘Experiments/Results/Discussion’ section.

Methods

KMeans

KMeans is perhaps the simplest unsupervised machine learning algorithm. It takes a number *k* and creates that many random centroids in a map of coordinates. The centroids update after each successive iteration, and once the centroids stop moving the algorithm stops (Dabbura, 2018). *See "KMeans pseudocode" in the Appendix.*

DBSCAN

DBSCAN works by clustering groups of high-density points. This thus makes it extremely good at filtering out noise and forming clusters by density. DBSCAN requires two parameters: epsilon and min_points. At each datapoint, it draws a circle (or hypersphere for higher dimensions) of radius epsilon. There are then three possible cases: if there are at least min_points other points within this circle, then the datapoint is identified as a "core point." If there are other points in the circle but less than min_points, it is identified as an "edge point." If there are no other points in the circle, it is classified as noise. Once the points are all classified in one of these three categories, the clusters can be extracted. Two points are in the same cluster if there is a chain of core points which can connect them together with connections of length epsilon or lower. It then expands the clusters, and all the points which do not fall into a cluster are considered noise (Schubert et al., 2017). *See "DBSCAN Pseudocode" in the Appendix.*

WILMA

WILMA creates a 'web' of links between points in order to separate the data into groups of links, or clusters. Points that are close have a higher likelihood of creating links and being in the same cluster. We modeled the process of creating links after the K-Nearest Neighbor algorithm: each point creates links to $\boldsymbol{N}$ number of neighbors. After doing this, we note all of the links and find ones that are significantly long (using the $Q_3 + 1.5 * IQR$ threshold that is used in statistics). We prune out these outliers, and after doing so we calculate the clusters that are created using a recursive method that searches through the links in a Breadth-First-Search manner. We do this entire process two times, one where $\boldsymbol{N}$ is a small number (like 2 or 3) and $\boldsymbol{N}$ is a larger number (like 5-8). The graph with the smaller N value has a tendency of underfitting the graph, creating an excessive number of clusters. On the other hand, the graph with the larger N values will overfit, creating a number of clusters that is too small with too many connections. We will compare the new links that are created between the two graphs, and note which clusters in the smaller-N graph are connected by the new links in the larger-N graph. We count the size of the clusters and the number of links that connect them, and attribute a score according to the following formula:

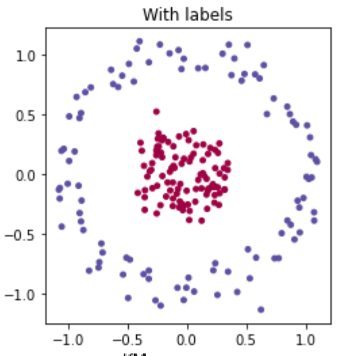
$$J = \frac{(\# \, of \, points \, in \, Cluster_1 + \# \, of \, points \, in \, Cluster_2)}{\# \, of \, links \, connecting \, C_1 \, and \, C_2 \, in \, large-N \, graph}$$

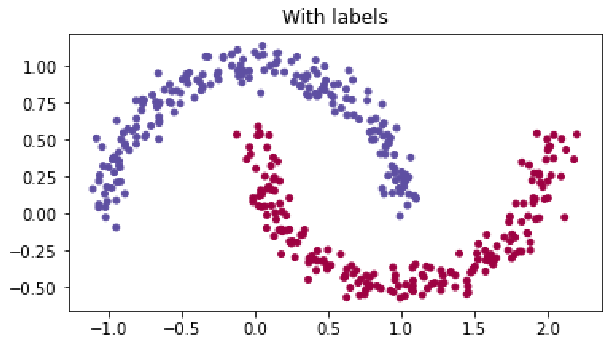
If a score is high, then the links that connect the two clusters should *not* be included, and should be pruned from the graph. Doing this will create a set of clusters that is perfectly in between an underfitted and overfitted graph. *See "WILMA Pseudocode" in the Appendix.*

Experiments/Results/Discussion

Data

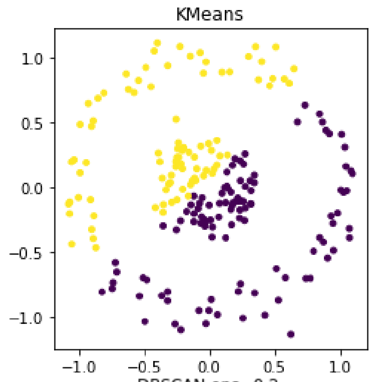
To the right are the two datasets (make_circles and make_moons) that we used against our three algorithms Notice how neither are linearly separable.

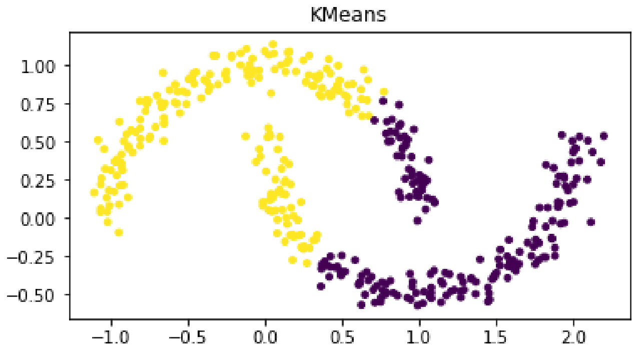




KMeans (k = 2)

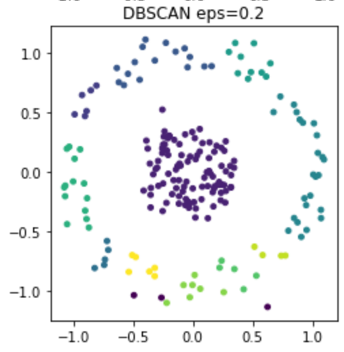
Here are the results for KMeans. Notice that we had to preset k = 2.

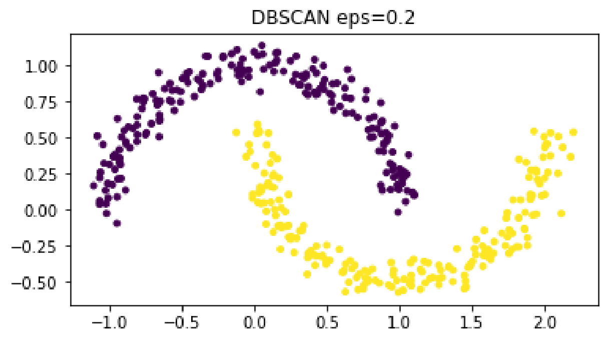




DBSCAN

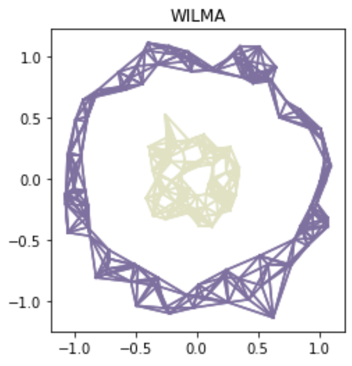
Here are the results for DBSCAN. Notice we had to give an estimate for epsilon (eps) = 0.2, which we found to be the most optimal.

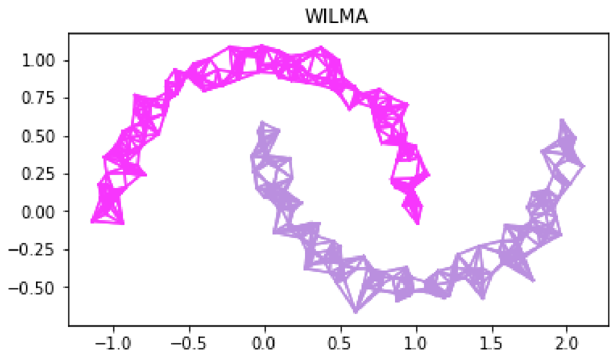




WILMA

Finally, here are the results for WILMA. There were no unique presets required in the process of using this algorithm.

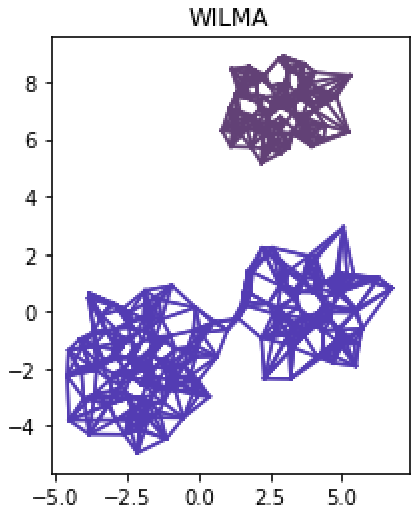




As shown by these clusterings, we see that K-Means is unsuitable for data of this shape, having a 52% accuracy for make_circles and a 69% accuracy for make_moons. DBSCAN performs well, with 100% accuracy, for make_moons but struggles severely with make_circles, creating 10 clusters instead of 2. DBSCAN stands out as being especially strong when parameters are tuned properly. For datasets where outliers need to be separately classified as "outliers," DBSCAN is an objectively better algorithm. However, due to there being two parameters to DBSCAN, namely epsilon and numpoints, it requires slightly more user involvement to optimize. WILMA also has optional parameters, ***N***-small and ***N***-large, however they have a far smaller effect on the results when compared to DBSCAN. Notice the make_moons dataset for DBSCAN: there is a fairly precise epsilon margin required to detect the moons while not classifying them as one continuous cluster. It should be noted that WILMA also has shortcomings, despite its perfect accuracy, as discussed in the next section.

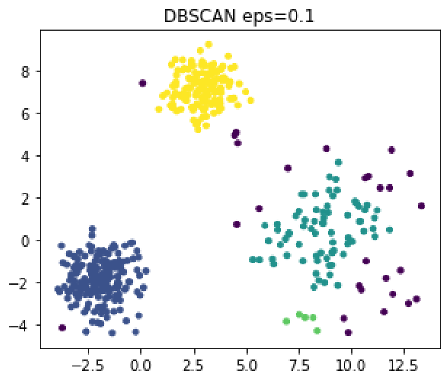
Conclusion/Future Work

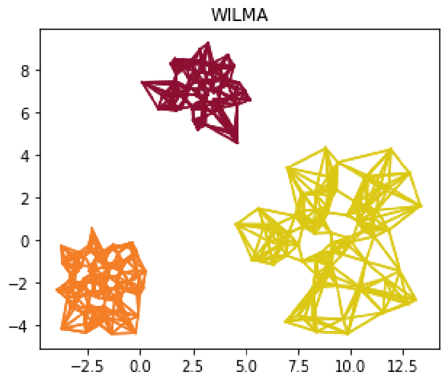
WILMA has potential for becoming a strong clustering algorithm, and there is no significant slowdown for higher-dimensional data, since all the calculations that WILMA performs are point-to-point distance calculations. However, WILMA does have difficulty in dealing with outliers, causing 'bridges' to form that should be kept separate, as shown to the right. The *J* formula attempts to solve this issue, but it is not perfect.



A "mutual neighbors" system, where two points are only connected if they *both* lie within *each other's* nearest ***N*** neighbors, would lead to some points having no links to begin with, which could be classified as "outliers."

WILMA's best advantage to DBSCAN remains being able to cluster data with varying densities, as seen below:





We estimate that an optimized version of WILMA will be slightly slower than DBSCAN, as both query the dataset to find neighbors. In order to not have to calculate the distance from each point to all other points, this becomes a *space partitioning* problem, with optimizations such as a K-d tree or a Quadtree. We wish to look at how DBSCAN is optimized within the sklearn libraries and see if those optimizations could be carried over to WILMA.

Contributions

Overall, our contributions to this paper as a whole were equivalent in both effort and variety. We didn't necessarily split up the required work into two separate batches for us to do separately, but rather worked through researching and creating/developing the algorithm together synchronously. We found that this was much more efficient as it gave us the opportunity to pour our ideas into creating an algorithm into one pool, which would have been much more difficult to do separately on our own intuitions. There were several times when we were stuck in coding our algorithm, but working together at the same time helped sort out any disarray and drastically sped up the process for reaching completion. There were also some instances where we happened to come up with a modification to our algorithm that could work, and we would code up this modification and show it to the other partner and see if it could be of any use. Such instances are listed below.

In terms of coming up with the idea, Nikhil was the one who theorized the Web-based approach of our algorithm. Hari and Nikhil thereafter worked together on coding the first drafts of the algorithm on a cloud-stored Jupyter-Notebook where we could actively code on the same file together. After the first draft was completed and we studied the performance of the initial algorithm, we experimented with new ways to improve upon older problems: Hari came up with a 'taxi-cab' algorithm which would attempt to find outlier links, and he also worked on using traditional statistical methods to find outlier links by using the interquartile range of lines and standard deviations. Meanwhile, Nikhil looked at considering an algorithm that would incorporate two separate runs of WILMA and find a "Goldilocks" set of clusters by combining the underfit and overfit models. We ended up using Hari's outlier technique for removing unnecessarily long lines, and Nikhil's two-run approach to finalize our algorithm.

References

Dabbura, I. (2018, September 17). *K-means clustering: Algorithm, applications, evaluation methods, and drawbacks*. Medium. Retrieved January 14, 2022, from https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a

Makin, J. G., O'Doherty, J. E., Cardoso, M. M. B., &amp; Sabes, P. N. (2018, January 25). Superior arm-movement decoding from cortex with a new, unsupervised-learning algorithm. Institute of Physics. Retrieved January 14, 2022, from https://iopscience.iop.org/article/10.1088/1741-2552/aa9e95/meta

Oppel, H., & Fischer, S. (2020, April 23). *A new unsupervised learning method to assess clusters of temporal distribution of rainfall and their coherence with flood types*. AGU Journals. Retrieved January 14, 2022, from https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2019WR026511

Schubert, E., Sander, J., Ester, M., Kriegel, H. P., & Xu, X. (2017, July 31). *DBSCAN Revisited, revisited: Why and how you should (still) use DBSCAN*. ACM Transactions on Database Systems (TODS). Retrieved January 14, 2022, from https://dl.acm.org/doi/abs/10.1145/3068335?casa_token=dfeNQrcoF6AAAAAA%3Aewu1h-nzMhzYlvaqOia5UfokUQA3WOSXrDjHSriaC8NJT4fDkg7v3Kp-tyU6hHhlKUd1W45w7_53ZxM

Sklearn.cluster.DBSCAN. scikit-learn. (n.d.). Retrieved January 14, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

Sklearn.cluster.Kmeans. scikit-learn. (n.d.). Retrieved January 14, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Appendix

KMeans Pseudocode

```
while centroids move:
      for each point in a graph:
            find and assign itself to the closest centroid
      for each centroid:
            average the coordinates of all of the assigned points
            update centroid to the average calculated above
```

DBSCAN Pseudocode

```
for point in graph:
      N = find neighbors
      if N is not dense:
            label point as noise
            continue
      C = new cluster
      label point as C
      for new_point in N:
            if new_point has no label:
                  continue
            label new_point as C
            N2 = find neighbors
            if neighbors is dense:
                  N.append(N2)
```

WILMA Pseudocode

```
for N in [small#, large#]:
      for point in graph:
            create links with N nearest neighbors
      clusters = group links that were just created
      two_webs.append(clusters)
new_links = two_webs[1] - two_webs[0]
for link in new_links:
      if the link connects two different clusters in two_webs[0]:
            clusters_connectors[(cluster#1, cluster#2)].append(link)
for (cluster#1, cluster#2) in clusters_connectors:
      score[(cluster#1, cluster#2)] = (#_of_points_in_cluster_1 +
            #_of_points_in_cluster_2 )/number_of_connecting_links
for cluster_score in scores:
      if scores[cluster_score] is large:
            prune links in clusters_connectors[cluster_score]
answer = two_webs[0]
```