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

Executable Code

Overview of WILMA Quarter 2 Project

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 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. However, WILMA had several issues, including slow run times and low resistance to outliers. In Quarter three, we aimed to recreate WILMA to account for these weaknesses.

WILMA Quarter 3 Project

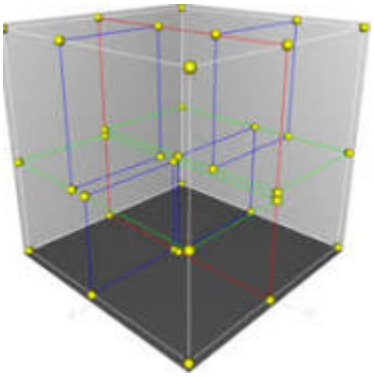
For the third quarter, we had two main goals with WILMA: decrease run time and increase resistivity to outliers. At the end of our quarter 2 project, we already had inklings of how to approach these two problems:

- To decrease run time, we wanted to implement a highly-complex algorithm to computer the N nearest-neighbors, which would likely be done in a faster language than python (C or Java)
- To improve outlier-handling, we wanted to implement a "mutual neighbors" condition for our underfit cluster, which would only establish links between two points if both points had a link going to each other
    - Beforehand, links were created even if only one point had other points as its neighbor, and the neighbor didn't have the original point as one

STEP 1: SPEEDUP

While doing the first step of this process, we found that scikit-learn has an incredibly efficient Nearest-Neighbors algorithm, conveniently located in sklearn.neighbors, as the class "NearestNeighbors". This tool was very handy because it would intake a NumPy array of points and use C-based code to find the nearest neighbors. In Quarter 2, we used a simple $O(n^2 \log n)$ algorithm for finding nearest neighbors:

```
for point in clusters: # O(n)
     neighbors = top N+1 entries in sorted(clusters) # O(nlogn)
```

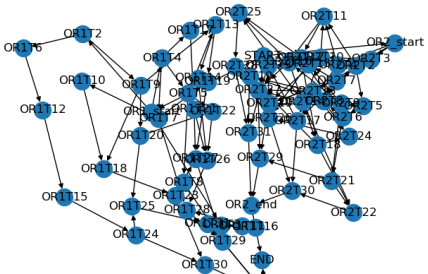


When working with a dataset of 2,000 points, this computation could take up to 15 to 20 seconds, which isn't horrible. However, when working with datasets of 200,000 points, or even a million points, doing this computation was *extremely* tasking. Using NearestNeighbors, we were able to work with more advanced algorithms rooted in C, which were not only more efficient, but were also much faster since they were written in C. We particularly used the 'kd_tree' algorithm which works by partitioning a k-dimensional space of data into smaller sub-portions which contribute to significant speedup, having an efficiency of only O(kn), where k is the number of dimensions. To illustrate how efficient kd-trees are, the algorithm can calculate the nearest neighbors for 2,000 points in only 0.006 seconds, which is miles faster than our previous method.

Another speedup that we implemented in our algorithm was the calculation of clusters using the `networkx` python package. This package works specifically to "study the structure, dynamics, and functions of complex networks," which is what WILMA relies heavily on. Beforehand, we used a BFS method written in python for calculating clusters, where we would visit every point in a graph:

```
clusters = []
not_visted = all point in a graph
for point in not_visited:
     look at neighbors of point & remove them from not_visited
     repeat this process until no more new neighbors
     add all visited points to clusters as a set of points
```

Admittedly, this wasn't too inefficient of an algorithm; it is essentially an O(n) algorithm because we only visit each point



in the graph only once. However, we decided to implement networkx because it was written in C, C++, and FORTRAN (very fast), computing 2,000 points within 0.202 seconds, which is quite fast in comparison to our older method which would take anything from five to ten seconds. We also used networkx because it had special customizations in cluster creating which were necessary in the outlier-handling portion of the project.

Final results:
On average, computing the old WILMA algorithm on 2,000 points took around 40 seconds, and the optimized WILMA algorithm took around 6 seconds, with a majority of the algorithm being taken up by the j-pruning algorithm.

STEP 2: OUTLIER-HANDLING

In handling outliers, we used one critical technique which we were in the process of developing in Quarter 2 but couldn't entirely complete in time: mutual-neighbor conditioning. In our old algorithm, a line between two points (points A and B) meant one of three things:

1. A has B as a nearest neighbor, but B doesn't have A as a nearest neighbor
2. The reverse of 1: B has A as a nearest neighbor, but A doesn't have B as a neighbor
3. Both A and B are neighbors of each other

Of the three scenarios posed above, most of the links between points are option 3, and fewer are options 1 and 2. We found that the instances in which scenarios 1 & 2 showed up were largely when either A or B was an outlier. This makes sense as the nearest neighbors for an outlier may be far away, but those nearest neighbors would likely have their nearest neighbors to be some other closer points.

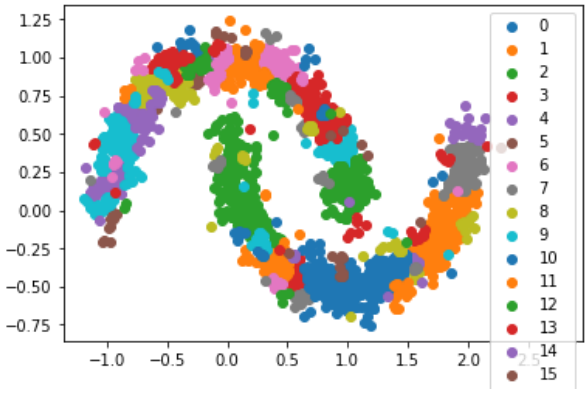
When we noticed this pattern, we came up with the "mutual-neighbor" contingency for the process of creating clusters, which would purposefully NOT create links between two arbitrary points if they weren't mutually connected (scenario 3). This results in a graph with fewer points, which we paired with our underfit graphs to make them further optimally underfit.

Implementing a mutual neighbor contingency in our project was possible by using the networkx package, because we can instantiate edges in our graph using mutual neighbor logic:

```
edges = []
for start, end in links:
     if end has start as links: # mutual
          edges.append(start, end)
```

The networkx algorithm was able to take these edge pairs we created and produced clusters using these edge lists.

After implementing the mutual-neighbor contingency for our underfit graphs, we found that it was able to single-out many outliers present in the graph. The graph on the right, for example, has 115 clusters, of which around 45% are single-point clusters that represent outliers. We modified the j-pruning algorithm to ignore these outliers when calculating j-scores, and once the whole algorithm finishes, it takes these outliers and simply assigns them to the same clusters as its nearest neighbor, effectively handling the outliers without affecting the integral clustering and j-pruning processes of WILMA that are extremely sensitive to outliers.



Conclusion

During the past quarter, we were not only able to significantly reduce the runtime of our algorithm by over 600% but were also able to increase the resistance of WILMA to outliers, both of which we had concerns about at the close of WILMA in the second quarter. We are very proud and confident in our results, which we planned out over the course of the third quarter and did in equal amounts. We are looking forward to publishing a further refined version of WILMA in the summer when we have more time on our hands. We also saw ICoMS 2022 and AMS 2022, two opportunities to publish research when perusing online for places to submit our work, but those deadlines had already passed by the end of April. We also saw that many places for publishing computer science research are going on in November of this year, which we will be ready for after our preparation in the summer.