OptiGrid Implementation on Multi-Variate Normal Datasets

Mangesh Sakordekar, Jacob James

Index Terms—Machine Learning, Clustering, Anomaly Detection, Statistics

1 Introduction

The Internet of Things has created an increasing demand for clustering algorithms to handle very large and high dimensional datasets. In regards to high dimensional data, "Hughes Phenomenon" states that as dimensionality increases, cluster performance increases as well until an optimum feature amount.[3] After the optimum feature amount, the performance decreases as feature dimensionality increases. A generic plot of "Hughes Phenomenon" is shown in Figure 1:

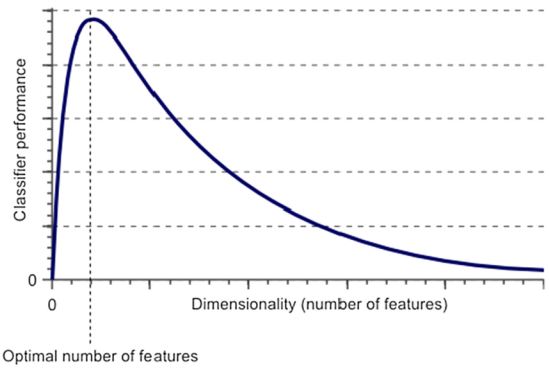

Fig. 1. Hughes Phenomenon

In an attempt to combat "Hughes Phenomenon", Hinnenburg and Keim developed a grid based clustering approach called OptiGrid which uses "optimum grid-partitioning" of the data using hyperplanes to project the data into partitions for each dimension. [1]

In this assignment, we were tasked with exploring the Optigrid clustering algorithm implementation using the mihailescum github repository[2], which follows the *Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering* research paper by Hinneburg and Keim. [1] The structure of this paper goes as follows: Background on Optigrid, Classes and Functions, Experiment and Results, Conclusion.

2 Background on OptiGrid

The structure of the Optirid Algorithm can be found in Figure 2. In the Figure, the number to the right of the words represent the function call order. Specific details for each function are discussed in the following sections.

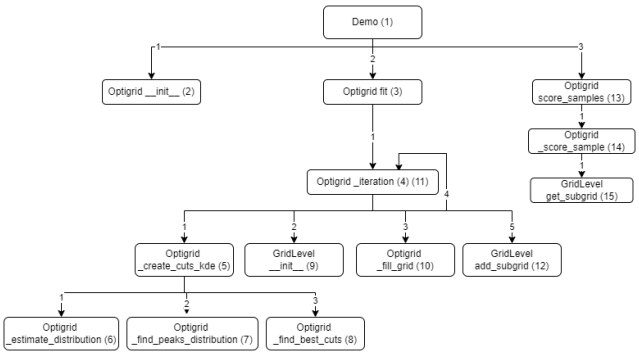

Fig. 2. Optigrid Algorithm Function Call Order

The following definitions will be taken from "Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering" which will be used throughout the paper to describe the Optigrid algorithm.[1]

Definition 1. "Let D be a set of n d-dimensional points and h be the smoothness level. Then, the **density function** $\hat{f}^D$ based on the kernel density estimator K is defined as:"

$$\hat{f}^D(x) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{x - x_i}{h})$$

There are many kernel density functions that have been used, but this implementation using a $1D$ Gaussian. The maxima of this density function above a given threshold is used to determine clusters.

Definition 2. "A **cutting plane** is a $(d-1)$ -dimensional hyper-plane consisting of all points y which fulfil the equation $\sum_{i=1}^{d} w_i y_i = 1$. The cutting plane partitions R^d into two half spaces. The decision function $H(x)$ determines the half space, where a point $x \in \mathbb{R}$ is located."

$$H(x) = \begin{cases} 1 & \text{if } \sum_{i=1}^{d} w_i x_i \geq 1 \\ 0 & \text{if } else \end{cases}$$

To implement this in the code, each dimension is projected onto it's axis and a $d-1$ dimensional "cut" is a line or plane with no slope along the respective dimension at the given kernel density minimum. A more complicated cutting

plane can be obtained by choosing different projection axis; however, this is out of the scope of the assignment and left for future research.

3 Classes

3.1 Optigrid

Optigrid class contains the methods and data to perform the Optigrid algorithm for high-dimensional clustering. It holds parameters like dimensionality of the data, number of cutting planes per iteration and noise level. The method Optigrid._iteration performs the recursive step of splitting the data using planes and the other functions act as helper functions.

Class Methods:

- __init__
- fit
- _iteration
- _create_cuts_kde
- _estimate_distribution
- _find_peaks_distribution
- _find_best_cuts
- _fill_grid
- score_samples
- _score_sample

Class Variables:

- d (int) : Dimension of the data.
- q (int) : Number of cutting planes per iteration.
- max_cut_score (double) : Maximum density of a cutting plane.
- noise_level (double) : Noise level used in finding peaks in the density distribution.
- root (GridLevel) : Represents the root of the hierarchical grid structure.
- clusters (list<int>) : Stores the clusters found by the algorithm
- num_clusters (int) : Represents the total number of clusters found.
- kde_bandwidth (double) : Bandwidth parameter for kernel density estimation.
- kde_grid_ticks (int) : Number of grid points for KDE.
- kde_num_samples (int) : Number of samples used for KDE.
- kde_atol (double) : Absolute tolerance for KDE.
- kde_rtol (double) : Relative tolerance for KDE.
- verbose (bool) : Flag to control verbosity.

3.2 GridLevel

GridLevel class contains the data structures used during the Optigrid algorithm. The data structure from this class is used to represent a single level of the grids and sub-grids representing the clusters throughout the Optigrid algorithm.

Class Methods:

- __init__
- add_subgrid
- get_sublevel

Class Variables:

- cutting_planes (list) :
- cluster_index (int) :
- subgrids (list) :
- subgrid_indices (list)

4 Functions

4.1 Optigrid.init

Function input Variables:

- self : optigrid member Function
- d (int) : Dimension of the data.
- q (int) : Number of cutting planes per iteration.
- max_cut_score (double) : Maximum density of a cutting plane.
- noise_level (double) : Noise level used in finding peaks in the density distribution.
- kde_bandwidth (double) : Bandwidth parameter for kernel density estimation.
- kde_grid_ticks (int) : Number of grid points for KDE.
- kde_num_samples (int) : Number of samples used for KDE.
- kde_atol (double) : Absolute tolerance for KDE.
- kde_rtol (double) : Relative tolerance for KDE.
- verbose (bool) : Flag to control verbosity.

Function Output Variables:

- None

Function Description:

Optigrid.__init__ initializes the variables which are required to perform the Optigrid algorithm. In this implementation, only d, q, max_cut_score and noise_level are mandatory parameters, others are optional. Parameter d is the dimension of the data, q is the number of cutting planes per iteration, max_cut_score holds the maximum density of the cutting plane and noise_level specifies the noise level used in finding peaks in the density distribution.

4.2 Optigrid.fit

Function input Variables:

- self : optigrid member Function
- data (ndarray) : Data to be Clusters
- weights (ndarray) :

Function Output Variables:

- None

Function Description:

The fit method initiates the clustering process and stores the resulting hierarchical grid structure and clusters for subsequent analysis. The method initializes the data structures and variables needed for clustering. Then it calls the _iteration function to perform the recursive clustering process to partition the data into subgrids. After the clustering process is complete, the method updates the class variables self.root, self.clusters and self.num_clusters to store the clustering results.

4.3 Optigrid.iteration

Function input Variables:

- self : optigrid member Function
- data (ndarray) : Data to be Clusters
- weights (ndarray) :
- cluster_indices (list <int>) : All indices that belong to the cluster
- percentage_of_values (double) : Percentage of values that lay in the current cluster (0-1)
- last_cluster_name (list<int>) : List containing one integer representing the previous cluster

Function Output Variables:

- GridLevel (GridLevel object) : The gridlevel at the current step with all its depth
- result (list<int>) : All clusters in the current data chunk

Function Description:

This function is a recursive function to do one step of the optigrid algorithm. It begins by determining the best cuts for all dimensions using the optigrid.create_cuts_kde function. It then sorts the cuts based on the density at the minima and selects the q best cuts. If the input verbose is true, it outputs the q cuts. It then generates and fills a grid containing subgrid for each cluster based on the cuts using the GridLevel class. Lastly, it iterates through all subgrid clusters by calling itself on each of the clusters recursively.

The best way to visualize this process is to associate it with depth first search algorithm on an n-ary tree. We keep on dividing the cluster into smaller clusters untill we do not find any cutting planes, and then we go back down the recursive stack to the previous level and move on the next cluster at that level.

There is an intermediate stopping condition, which is hit when the one cluster is completed. The final stopping condition is met at the end of the function which outputs the entire grid and clusters associated with it.

4.4 Optigrid.create_cuts_kde

Function input Variables:

- self : optigrid member Function
- data (ndarray) : Data to be Clusters
- cluster_indices (list<int>) : All indices that belong to the current cluster
- current_dimension (int) : Dimension on which to project
- percentage_of_values (double) : Percentage of values that lay in the current cluster $[0, 1]$
- weights (ndarray) :

Function Output Variables:

- best_cuts (list <list<double, int, double >>) : Contains the best q cuts in format of (position, dimension, cutting_score)

Function Description:

This function finds the best cuts for the specified dimensions by estimating the data intensity using the scipy gaussian_kde function. It begins by calling the optigrid.estimate_distribution function which returns the grid and kde for the current dimension. From the kde, the peaks are determined by calling the optigrid.find_peaks_distribution and are sorted. The grid, kde, sorted peaks, and current dimension are passed into the optigrid.find_best_cuts function which returns the best_cuts. This best_cuts variable is returned, which is a list containing the q best cut in the format of (position, dimension, cutting_score).

4.5 Optigrid.estimate_distribution

Function input Variables:

- self : optigrid member Function
- data (ndarray) : Data to be Clusters
- cluster_indices (list<int>) : All indices that belong to the current cluster
- current_dimension (int) : Dimension on which to project
- percentage_of_values (double) : Percentage of values that lay in the current cluster $[0, 1]$
- weights (ndarray) :

Function Output Variables:

- grid (list<double>) : An equally spaced grid
- density_of_grid(list<double>) : The density of the grid points

Function Description:

The function estimates the distribution using a sample of the data projected to a coordinate axis using the scikits kde estimate function.

It begins by amount of samples associated to a given cluster. It then shuffles the samples in the cluster and determines the min, max, and standard deviation. Using the scikit gaussian_kde function, it determines the kernel density estimation. Lastly, it generates a list from the min to max iterating by the grid ticks, along with the density of the kde and returns the grid list and the kde density normalized by the amount of points in the current cluster.

4.6 Optigrid.find_peaks_distribution

Function input Variables:

- self : optigrid member Function
- kde (list<double>: The density estimates on an arbitrary 1D grid

Function Output Variables:

- peaks (list<int>) : The corresponding indices of the grid where the kde has its peaks

Function Description:

This function determines the peaks of the kde that are above a specified noise threshold.

The function iterates through each index of the kde and appends all index values that are greater than the noise threshold, returning the peaks variable.

4.7 Optigrid.find_best_cuts

Function input Variables:

- self : optigrid member Function
- grid (list<double>) : The grid on which the density estimate was evaluated
- kde (list<double>: The density estimates on an arbitrary 1D grid
- peaks (list<double>) : The maximum of the density estimate on the grid
- current_dimension (int) : Dimension on which the data is projected

Function Output Variables:

- best_cuts (list<list<double, int, double>>) : Best q cutting planes in this dimension in the format (position, dimension, cutting_score)

Function Description:

The function uses the kde and its maxima to determine the best cutting plane.

The function iterates through all the input peak combinations. For each combination, if the peak is the smaller than the minimum peak, the index in the grid, current_dimension, and peak value are appended to the best_cuts list and is stored as the new minimum and continues until all combinations are accounted for.

4.8 GridLevel.init

Function input Variables:

- self : optigrid member Function
- cutting_planes (list) : The planes that are used to subdivide the grid level or None if it represents a cluster
- cluster_index (int) : The index of the represented cluster or None if it can be subdivided further
- subgrids (list) : Empty list to store subgrids through Optigrid algorithm
- subgrid_indices (list) : Empty list to store subgrid_indices through Optigrid algorithm

Function Output Variables:

- None

Function Description:

GirdLevel.__init__ initializes the variables which are required to generate the grid level object used throughout the Optigrid algorithm. This object contains lists of integers representing binary encoded representations of cutting planes for higher dimensional data.

4.9 Optigrid.fill_grid

Function input Variables:

- self : optigrid member Function
- data (ndarray) : Data to be clustered
- cluster_indices (list<int>) : All indices that belong to the current cluster
- cuts (list) : Cutting planes in the format (position, dimension, cutting_score)

Function Output Variables:

- grid (list<list<int>>) : 2 **num_cuts lists of indices representing the clusters in this level

Function Description:

This function partitions the data into clusters based on the cutting planes and returns a list of lists where each sublist contains the indices of datapoints belonging to the specific cluster. The parameter data is an ndarray, cluster_indices is a list of integers which contains all the indices belonging to the current cluster and cuts is a list of the cutting planes in the format (position, dimension, cutting_score) and returns a list of list of integers representing the clusters at this level.

The function initializes a variable grid_index, an array of zeros with the same length as cluster_indices. This array is used to keep track of which cell of the grid each data point falls into. It then iterates over the cutting planes and assigns a unique value to each cell in grid_index based on where the point lies with respect to the cutting plane in that dimension. Finally, the function creates a list of clusters by grouping the indices based on the unique values in grid_index. The result is a list of lists where each sublist represents a cluster in the current level of the grid.

4.10 GridLevel.add_subgrid

Function input Variables:

- self : GridLevel member Function
- subgrid_index (int) : For every cutting plane, the subgrid can lay either right or left. Used to binary encode
- subgrid (GridLevel object) : The subgrid to add

Function Output Variables:

- None

Function Description:

This function adds a deeper level to the grid by appending the input subgrid_index and subgrid to the class subgrid_indices and subgrids class objects.

4.11 Optigrid.score_samples

Function input Variables:

- self : optigrid member Function
- samples (list <ndarray>) : The samples to score.

Function Output Variables:

- cluster_pred (list<int>) : Cluster assignment for each sample or None if no cluster

Function Description:

The score_samples method provides a way to determine the membership of a set of samples based on the hierarchical clustering structure created by the algorithm. It iterates over each sample in samples list and passes on each sample to Optigrid._score_sample function to determine the scoring of that sample. The function then makes a list of all the scores and returns it.

4.12 Optigrid.score_sample

Function input Variables:

- self : optigrid member Function
- sample (ndarray) : The sample to score.

Function Output Variables:

- current_grid_level.cluster_index (int) : Cluster assignment for the sample or None if no cluster

Function Description:

The _score_sample method is a helper function to the score_samples method. It is responsible for determining the cluster to which a given sample belongs based on the hierarchical grid structure created during the Optigrid algorithm's fitting process.

The parameter sample is a ndarray which contains the sample to be scored. The method starts at the root level of the hierarchical grid structure. It traverses through the grid structure, moving down through subgrids, until it reaches a leaf node. If a leaf node is reached, the cluster index associated with that node is returned. If the sample does not fall into any cluster, the method returns None.

4.13 GridLevel.get_subgrid

Function input Variables:

- self : GridLevel member Function
- datapoint (ndarray) : The sample.

Function Output Variables:

- subgrid (GridLevel object) : The subgrid or -1 if it belongs to no subgrid (outlier)

Function Description:

This function determines the subgrid the input data point lies in. It iterates through the cutting planes list and determines the subgrid based on the binary value.

5 Experiments

The algorithm was tested using $2-5$ inclusive clusters consisting of $3D$ multivariate normal. The multivariate normal data was concatenated and used as training data to determine the optimum cutting planes. The resulting cutting planes were plotted to visualize the validity of the cutting planes. Additionally, ten samples of each class were generated and used as testing data and the overall clustering accuracy was determined for each class.

5.1 Dataset

Three dimensional Multivariate Normals with different means and similar covariance matrices were generated to determine each class. The classes were then concatenated into one numpy array which was used as the input data. The data was then standardized so the projections in the optigrid algorithm would be on the positive axis. Ten samples of each multivariate normal were used as the test data.

5.2 Experimental Results

For $3D$ data, cluster test accuracy was 100% and the cutting planes visually separated the clusters. The plots separating two and three clusters are shown in Figure 3 and 4 in the appendix.

As the clusters became more than three, the algorithm began predicting more clusters than in the labels. We predict this is due to the projections. The projections used in this example were to the respective dimensions, which have difficulty handling clusters with similar projections. In order to improve cluster accuracy, different projection axis could be used with slopes greater than zero. Additionally, adjusting the cluster centers to be further apart would improve accuracy, but this would deviate from real world data. The visual representations of the cutting planes are shown in Figure 5 and '6 in the Appendix.

6 Conclusion

The Optigrid algorithm with mihailscum's github implementation was thoroughly examined in this assignment. The implementation used cutting planes or projections to the respective axis, which created difficult cutting planes when cluster became close and similar in some dimensions. In future exploration, different projections can be explored to determine the accuracy improvements.

Appendix A
Cutting Planes for Clustering Figures

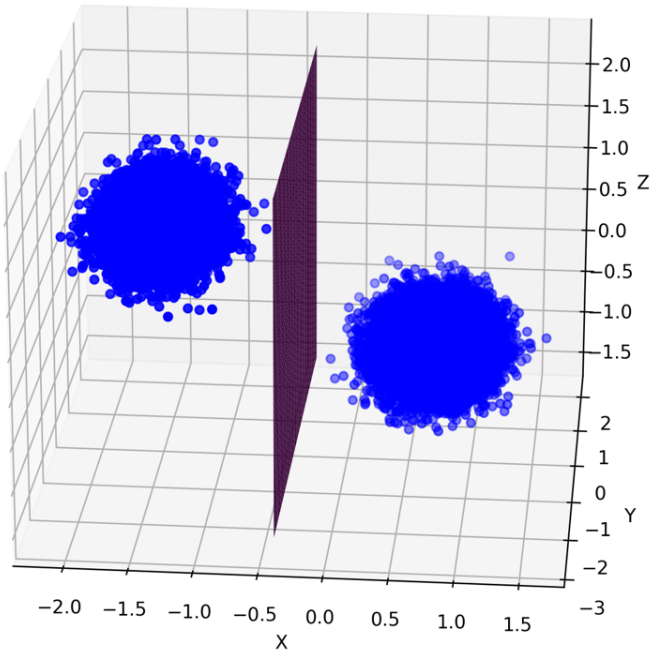

Fig. 3. Two Clusters with one Cutting Plane

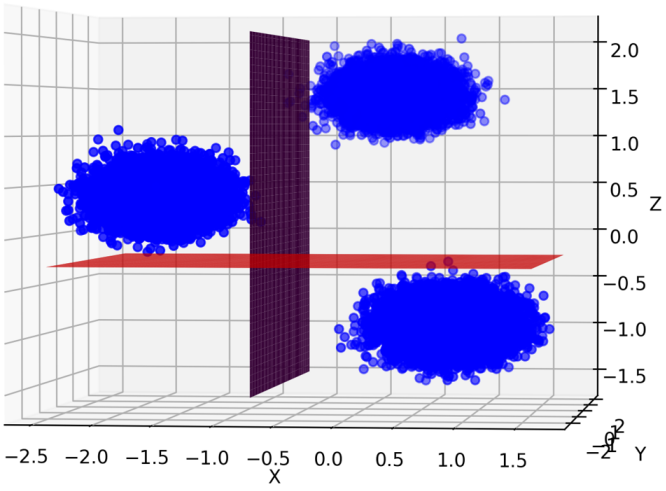

Fig. 4. Three Clusters with Two Cutting Planes

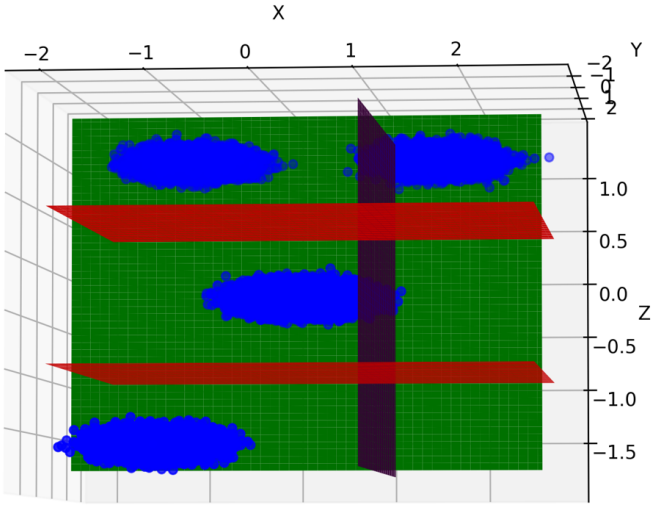

Fig. 5. Four Clusters with Four Cutting Plane

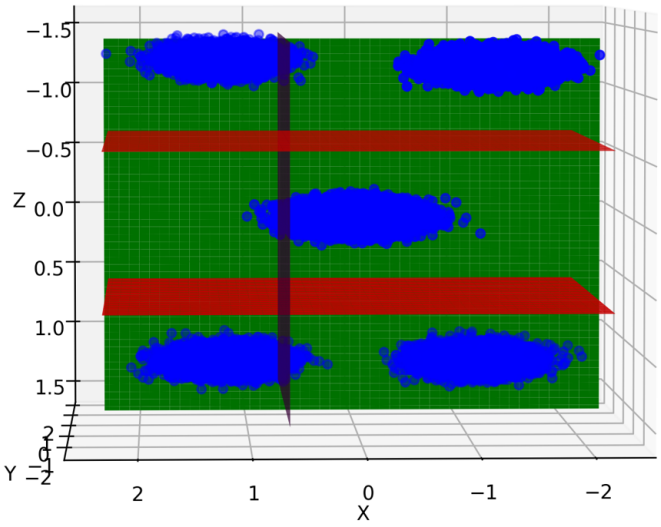

Fig. 6. Five Clusters with Five Cutting Plane

REFERENCES

[1] Alexander Hinnenburg and Daniel Keim. “Optimal Grid-Clustering: Towards Breaking the Curse of Dimensionality in High-Dimensional Clustering”. In: *International Conferene on Very Large Databases* (1999).

[2] mihailescum. *OPtigrid*. https : / / github . com / mihailescum/Optigrid. 2022.

[3] Badreesh Shetty and Jessica Powers. *What Is the Curse of Dimensionality?* https://builtin.com/data-science/curse-dimensionality. 2022.