CSC730: Report for Assignment 3

South Dakota School of Mines and Technology

Carson Price, Kris Jensen

February 5, 2024

1 Introduction

Our task with assignment three is to describe the OptiGrid algorithm, analyze the Optigrid code available on Github [1], in great detail. After analyzing the algorithm and the code, we will modify the demo example to accept a 2D dataset of 30000 points and create visualizations of the data points and cutting planes. If time permits, we can earn extra credit by extending the modifications and visualization to a 3D dataset of 30000 points.

2 Description of OptiGrid

The OptiGrid algorithms works on a dataset by first determining if the dataset can be contracted, dimensionally reduced, by projecting the data onto a d-1 space. This means that if the dataset can be divided into two seperate spaces by a plane, then the dataset can be seperated into at least two classes. Simply drawing a plane through the data is insufficient to determine the best plane to divide the data.

The algorithm will determine if a plane exists to divide the data by calculating the score of the plane. The score is determined by applying a kernel density estimation function to the data in the current dataset. Depending on the kernel density esimation function and bandwidth, the score will contains some number of peaks. If there are zero or one peaks, then the splitting for this dataset is complete. If the score is above a certain threshold, then the plane is considered a good plane to divide the data. The data is then labeled as being left or right, or above or below the plane.

We will refer to left or top as A and right or bottom as B. The total dataset, D, is the union of A and B. Upon succesfull splitting of A and B, then the algorithm will recursively apply the same process to A and B. The algorithm will continue to split the dataset until the dataset is no longer able to be split.

3 Analysis of OptiGrid Code

Let's begin by setting the stage for the OptiGrid code, by analyzing the pseudocode set out by Hinneburg and Keim [2]. Then the structure of the code will be outlined, followed by a detailed analysis of each function and its purpose.

OptiGrid(dataset D, q, min_cut_score)

1. Determine a set of contracting projections P = {P_0, . . ., P_k}
2. Calculate all projections of the dataset $D \rightarrow P_0(D)$, . . ., $P_k(D)$
3. Initialize a list of cutting planes $BEST\_CUTS \leftarrow \emptyset, CUT \leftarrow \emptyset$
4. FOR i=0 TO k Do
   a. CUT ← Determine best_local_cuts($P_i(D)$)
   b. CUT_SCORE ← score_best_local_cuts($P_i(D)$)
   c. Insert all cutting planes with a score ≥ min_cut_score into $BEST\_CUTS$

   END FOR
5. IF $BEST\_CUT = \emptyset$ THEN RETURN D as a cluster
6. Determine the q cutting planes with highest score from $BEST\_CUTS$ and delete the rest
7. Construct a Multidimensional Grid G defined by the cutting planes in $BEST\_CUTS$ and insert all data points $x \in D$ into G
8. Determine clusters, i.e. determine the highly populated grid cells in G and add them to the set of cluster C
9. REFINE(C)
10. FOREACH Cluster $C_i \in C$ DO OptiGrid(C_i, q, min_cut_score)

3.1 Functions of class OptiGrid

1. **__init__(d, q, max_cut_score, ...)**
   The *__init__* functions acts as the class constructor and is called automatically upon instantiation. It initializes the class variables and sets the default values for the parameters. The parameters include dataset dimension (**d**), number of cuts per iteration (**q**), the max cut score density of a plane (**max_cut_score**), noise level for dataset(**noise_level**), several parameters related to the kernel density estimation including bandwidth (**kde_bandwidth**),

grid ticks (**kde_grid_ticks**), sample size (**kde_num_samples**), tolerance (**kde_atol**) and (**kde_rtol**), and finally an argument for turning on or off output (**verbose**) . This function sets the initial conditions of the OptiGrid algorithm.

2. **fit**(**data**, **weights**)
   The fit function is the function that is called to start the OptiGrid algorithm. It is the main function that calls all the other functions in the class. The fit function takes in the dataset and the initial weights as a parameter.

   The fit function first records the data length and initializes the list of clusters. Following this setup, the *_iteration* method is called which begins the OptiGrid algorithm.

3. **_iteration**(**data**, **weights**, **cluster_indices**, ...)
   The first step in the *_iteration* function, a pseudo-private method, is to create an empty list of cuts. The list of cuts is generated by looping through all dimensions of the dataset and calling the $\_find\_best\_cuts$ function. This loop will generate all cutting planes from d=1 to d=len(self.d). The $current\_dimension$ paramter sent to the $\_find\_best\_cuts$ function is incremented by 1 each iteration.

   If the list of cuts is empty, then the function returns the dataset as a cluster and indicate there are no further cutting planes available for this dataset. If the list of cuts is not empty, the list of cutting planes is sorted by score. The cutting planes discovered in the previous step are then passed to GridLevel to construct a multi-dimensional grid. The first call to GridLevel is made with the cutting planes list to construct the grid for this iteration in the recursive call stack. Then a grid of cutting planes is created from the data and clusters. This grid data contains the information if the data is left or right of the cutting plane and encoded as either 0 or 2^i

   At this point, the algorithm has created a grid and the data is labeled as being left or right, or above or below the plane. This data needs to be iterated through to recursively apply the same process to the left and right datasets if the size of cluster exceeds 0. When the first call to $self.\_iteration$ is made, the algorithm will continue to split the dataset until the dataset is no longer able to be split. Finally, when this first call returns, the algorithm will have found all the clusters in the dataset.

   This function is the top level code that implements the pseudocode for the OptiGrid algorithm. Lines 66 through 68 accomplish pseudocode lines 1 through 6. Line 81 accomplishes psuedocode line 7. Line 83 fulfills pseudocode line 8. Lines 85 through 96 accomplish psuedocode step 9 and 10.

4. **_fill_grid**(**data**, **cluster_indices**, **cuts**)
   The semi-private method $\_fill\_grid$ is called to fill the grid with the data and cluster indices. Reviewing the pseudocode, this function accomplishes step 8. A labelling scheme described in definition 5 and definition 6 of [2] is used to label the data bifurcated by the cutting planes. The method loops through all the cuts and sets the $grid\_index$ location to either 0 or 2^i depending on the conditional broadcasting that determines on which side of the plane the datapoints lie.

5. **_create_cuts_kde**(**data**, **cluster_indices**, **cuts**, ...)
   The $\_create\_cuts\_kde$ function is a high level function that accomplishes the pseudocode step 3 through step 6. The workflow of the function is to first create a kernel density estimation of the data. Following this, the function finds the peaks in the KDE and then finds the best cuts. The best cuts are selected in the the $\_find\_best\_cuts$ function.

6. **_find_best_cuts**(**grid**, **kde**, **peaks**, **current_dimension**)
   The $\_find\_best\_cuts$ function is called to find the best cuts for the current dimension. This is accomplished by searching between peaks for the minimum value. If the current value is less than the minimum value, then the current value is set to the minimum value. In this way, the algorithm finds the deepest valleys between peaks and adds that location to a list called $best\_cuts$.

7. **_find_peaks_distribution**(**kde**)
   This function iterates throug the KDE data searching for peaks. A peak is defined as a point that is greater than the points to the left and right of it. This algorithm is fairly naive and could be improved by using a more sophisticated peak finding algorithm. For instance a a sawtooth pattern would be detected as a series of peaks.T The peak finding algorithm from scipy.signal would be a better choice.

8. **_estimate_distribution**(**data**, **cluster_indices**, **current_dimension**, **percentage_of_values**, **weights**)
   This is function that applies the kernel density estimation to the data. The first step is to create a grid of points that will be used to estimate the density. The point grid sample size is determined by the number of samples parameter, unless it exceeds the length of the data, in which case the sample size is set to the length of the data. Then a sample set of data is createed using the np.random.choice function. The minimum and maximum values of the data are used to create a one-dimensional grid of points with uniform spacing. The KDE is evaluated on the data against the linear grid of points. The KDE is then normalized and the grid of points and the KDE are returned.

9. **score_samples**

10. **_score_sample**

3.2 Functions of class GridLevel

1. **__init__**

2. **add_subgrid**

3. **get_sublevel**

4 Results

5 Discussion

6 Conclusion

7 References

[1] mihailescumihai/optigrid. (2019). GitHub. https://github.com/mihailescum/Optigrid
[2] Hinneburg, A., & Keim, D. A. (1999). Optimal grid-clustering: Towards breaking the curse of dimensionality in high-dimensional clustering. In Proceedings of the 25th International Conference on Very Large Data Bases (pp. 506-517). Morgan Kaufmann Publishers Inc.