Towards a Scalable Online Hierarchical Clustering Algorithm

by

Ishita Doshi

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Master of Technology
in Computer Science and Engineering,
Indian Institute of Technology Gandhinagar
2019

Advisory Committee:

Prof. Anirban Dasgupta
Prof. Manoj Gupta
Prof. Shanmuganathan Raman

CERTIFICATE

It is certified that the work contained in the thesis titled "**Towards a Scalable Online Hierarchical Clustering Algorithm**" by **Ishita Doshi** (Roll No. 17210038) has been carried out under my supervision and this work has not been submitted elsewhere for a degree.

Prof. Anirban Dasgupta
Associate Professor
Computer Science and Engineering
Indian Institute of Technology Gandhinagar
Gujarat - 382355, India

Date: June 11, 2019
Place: Gandhinagar

Abstract

Clustering algorithms play a vital role in organizing data in small meaningful groups. Hierarchical clustering has some added benefits. It does not require specification of the number of clusters and it outputs a more structured hierarchy which could help in understanding the data better. However, the use of hierarchical clustering in practice has been hampered by the lack of implementable algorithms with a provable theoretical bound on the quality. To address this situation, we propose splitting strategies for divisive hierarchical clustering which are scalable in terms of time, have linear space usage and have theoretical guarantees in terms of a specific hierarchical clustering objective. We also demonstrate the excellent empirical performance of an efficient implementation of the proposed strategies for real-time near-neighbor classification and anomaly detection.

In many modern applications, due to the large scale of data and the unavailability of data prior to building the model, an online or streaming approach is required. To solve the online hierarchical clustering problem, we first propose two naive heuristics and demonstrate their performance empirically. We also identify a natural sub-problem - *Online eigenvector estimation.* We propose an algorithm to maintain the top eigenvector for a stream of operations where the data may be inserted or deleted. The performance of the algorithm is evaluated theoretically on a model proposed by us, the *orthogonal dual spiked covariance model*, and empirically on the MNIST dataset.

Acknowledgements

It is my pleasure to acknowledge the roles of several individuals and institutions who were instrumental for the completion of this thesis.

First of all, I would like to express my sincere appreciation to Prof. Anirban Dasgupta for his invaluable guidance. His support, suggestions, and patience have been instrumental for the development of this work.

I would also like to thank everyone in the Computer Science field of Indian Institute of Technology Gandhinagar for their constant support and for the various opportunities given to me throughout my Masters program.

I would like to thank Jayesh Choudhari, Dr. Rushi Bhatt, Ram Madhavan and Sreekalyan Sajjala for the suggestions, discussions and constant support.

Apart from technical support, I would like to express my appreciation towards my family, friends and colleagues for their support, patience, understanding and unconditional love.

Contents

List of Figures

List of Tables

List of Algorithms

Abbreviations

PSD	**Positive S**emi-**definite**
EV	**E**igen**V**ector
AEV	**A**pproximate **E**igen**V**ector
RP	**R**andom Hyper**p**lane
HTNN	**H**eirarchical **T**ree based **N**earest **Neighbors**
ANN	**A**pproximate **N**earest **N**eighbors
LT	**L**inkage based **T**rees
SVM	**S**upport **V**ector **M**achine
RBF	**R**adial **B**asis **F**unctions
LSH	**L**ocality **S**ensitive **H**ashing
ALOI	**A**msterdam **L**ibrary **o**f **O**bjects
GloVe	**Glo**bal **Ve**ctors for word representation
PERCH	**P**urity **E**nhancing **R**otations for **C**luster **H**ierarchies
P	**P**recision
R	**R**ecall
F1	**F1**-Score

Symbols

λ	Eigenvalue or Singular Value
$\|\| \cdot \|\|_2$	2-norm
$\sigma(\cdot)$	Sparsest Cut
$\phi(\cdot)$	Expansion
$\gamma(\cdot)$	Conductance
$\mathcal{L}$	Laplacian
$\mathbb{E}$	Expectation
$\mathbb{R}$	The set of Real numbers
$\sum_i^j$	Summation from i to j
$\| \cdot \|$	Size of the set
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ and variance σ^2
$< \mathbf{a}, \mathbf{b} >$	Dot product between **a** and **b**
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	Graph $\mathcal{G}$ with $\mathcal{V}$ vertices and $\mathcal{E}$ edges
$i \vee j$	Least common ancestor of i and j

Introduction

Identifying interesting content for recommendation and personalization, or finding suspicious activity is a fundamental problem for many social networks. The motivating problem for us was identification of inappropriate and suspicious content posted on social media. As better models evolve, spammers and hackers also evolve to find new methods to evade detection. Thus, supervised learning and batch learning approaches would always have high latency to identify the latest loopholes the adversaries would have found. Therefore, faster, unsupervised, incremental algorithms are required to find newer types of content.

Identifying clusters of content which may differ from previously seen types is the first step to pro-active identification of large spam attacks. Since we are faced with a wide range of content types, many of which are new, it does not make sense to use flat clustering algorithms with a pre-set number of clusters. However, the content types do have natural subtype-supertype relations which are potentially useful for identification of new variants of known types.

While it is essential to identify new content types, it is also necessary have the ability to modify our model incrementally (in an online manner) to account for the newer content types. Since data is arriving at a rapid speed, it is not feasible to get all the new content types labelled as soon as they arrive. Thus, once a new content type has been identified, the model needs to be modified to account for that type.

To summarize, we need the following capabilities from our content partitioning algorithm.

1. The algorithm needs to be unsupervised to ensure rapid response to new spam attacks which have not been labelled yet, as well as for agnosticity with respect to different types of labelling strategies.
2. It should have the ability to tag content in a hierarchy of classes to aid the process of manual reviewing as well as to borrow strength from content subtype-supertype relations and from user feedback.

3. Query time, i.e., identification of the content type, needs to be near-real time to prevent damage. Space usage should be scalable, preferably linear.
4. The algorithm should allow for incremental updates for fast incorporation of newly identified content types.

Hierarchical clustering, a method of cluster analysis seeks to build a hierarchy of clusters. A hierarchical clustering is usually represented by a dendrogram. In this work, we propose an unsupervised online hierarchical clustering algorithm to serve the aforementioned needs for data streams with a multitude of dynamic content types. While there are a number of hierarchical clustering algorithms, until recently, there has not been any objective function to quantify the quality of a hierarchical clustering. Dasgupta (2016) was the first to propose an objective function for hierarchical clustering. Cohen-Addad et al. (2017) demonstrate the appropriateness of the proposed cost function.

Dasgupta (2016) shows that obtaining the optimal clustering is NP-Hard through a reduction from a variant of the *3-SAT* problem called the *not-all-equal-SAT*. While there are approximation algorithms that have been proposed by Dasgupta (2016) for batch hierarchical clustering, these are based on max-flow and semi-definite programming Arora et al. (2009); Leighton and Rao (1999). These algorithms are not appropriate in our setting because of two reasons

1. They are not practical for large datasets, and
2. Resulting hierarchies cannot be used to fine where the new point fits in without reconstructing the entire hierarchy.

We propose a hierarchical clustering algorithm based on eigenvectors, approximate eigenvectors and random hyperplanes that optimizes the cost function proposed by Dasgupta (2016).

Though batch hierarchical clustering has most of the desirable properties, we cannot incrementally modify the model to account for the new content types that have been identified. Therefore, it is desirable to have an “online” hierarchical clustering algorithm. An online algorithm processes data piece-by-piece in a serial fashion, and at any point in time, it maintains a solution for all data seen till that point.

Online hierarchical clustering algorithms have not received their fair share of attention. Though algorithms such as the one proposed by Kobren et al. (2017) exist to perform online hierarchical clustering, no work exists with a guaranteed approximation to any cost function.

We first propose scalable algorithms based on eigenvectors, approximate eigenvectors and random hyperplanes for the batch hierarchical clustering problem with guaranteed approximations to the above cost function. We then empirically demonstrate how to use the hierarchy obtained in order to label new content using a variant of near-neighbor classification, as well as in identification of anomalous content.

We also propose heuristics for the online hierarchical clustering problem and demonstrate empirically that there is little quality degradation when compared to the batch hierarchical clustering.

While trying to give theoretical bounds on the quality of online hierarchical clustering algorithms, we encountered the sub-problem of online eigenvector updates. While all existing work, for e.g., Mitliagkas et al. (2013); Garber et al. (2015) propose algorithms to maintain eigenvectors in data streams which only deal with addition of new data, our algorithm inherently requires addition as well as deletion of points. We propose algorithms to maintain the largest eigenvectors in a data stream under addition and deletion of points, give theoretical guarantees for a random model, i.e., the *orthogonal dual spiked covariance model* (described in Section 6.0.2), and demonstrate empirical performance of the proposed algorithms.

1.0.1 Our Contributions

Specifically, our contributions are as follows

1. Practically efficient hierarchical clustering algorithms based on eigenvectors, approximate eigenvectors and random hyperplanes.
2. The hierarchical clustering obtained using these strategies have guaranteed approximations to the hierarchical clustering objective.
3. Empirical demonstration of the use of the hierarchy to label new content using a variant of near neighbor classification, and to identify anomalous content.
4. We demonstrate the performance of the eigenvector based hierarchical clustering strategy on the *planted partition* random graph model.
5. Heuristics for the online hierarchical clustering problem and empirically demonstrate the excellent performance of the heuristics.
6. An algorithm to calculate and maintain eigenvectors in a dynamic stream where data may be added or deleted. We propose the *orthogonal dual spiked covariance model* and give theoretical results for the proposed algorithm for this model, and we also demonstrate the empirical performance on an open-source dataset.

1.0.2 Organization of the Thesis

In Chapter 2, we outline the preliminaries, followed by the related work in Chapter 3. In Chapter 4, we elaborate the proposed solution for efficient hierarchical clustering, give theoretical bounds on the quality of solution and demonstrate the empirical performance of the algorithms. In Chapter 5, we give two heuristics for the online hierarchical clustering problem and demonstrate its performance empirically. Chapter 6 gives algorithm for online eigenvector updates in a dynamic stream. We also give some theoretical results and demonstrate the performance on the MNIST dataset. We also present algorithms for the case where top-k eigenvectors are to be maintained. In Section 7, we conclude and outline some possible future work.

Preliminaries

This chapter aims to build the background required for the theoretical part of the thesis. We provide some definitions and quote some theorems to help the reader understand our proofs.

2.0.1 Cost Function for Hierarchical Clustering

In Dasgupta (2016) proposed the cost function for a weighted undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$. Let T be a rooted hierarchical clustering tree (not necessarily binary) for graph G, where the leaves of the tree T are the nodes in the graph G. If $i \vee j$ denotes the least common ancestor of i, j (say N) in the tree T, and $|leaves(i \vee j)|$ denotes the number of leaves in the subtree rooted at N. The proposed cost function is given by:

$$cost_{\mathcal{G}}(\mathcal{T}) = \sum_{(i,j)\in\mathcal{E}} w_{ij}|leaves(i \vee j)| \tag{2.1}$$

If the weights of the edges denote the similarity between the two endpoints, then,

$$cost_{\mathcal{G}}(\mathcal{T}^*) = min_{\mathcal{T}}\, cost_{\mathcal{G}}(\mathcal{T})$$

If the weights of the edges denote the distance between the two endpoints, then,

$$cost_{\mathcal{G}}(\mathcal{T}^*) = max_{\mathcal{T}}\, cost_{\mathcal{G}}(\mathcal{T})$$

where T^* is the optimal clustering tree.

The dual to this cost function was given in Bateni et al. (2017) as

$$rev_{\mathcal{G}}(\mathcal{T}) = \sum_{(i,j)\in\mathcal{E}} w_{ij}|non - leaves(i \vee j)|$$

If the weights of the edges denote similarity between the two endpoints, then,

$$rev_{\mathcal{G}}(\mathcal{T}^*) = max_{\mathcal{T}}\, rev_{\mathcal{G}}(\mathcal{T})$$

If the weights of the edges denote the distance between the two endpoints, then

$$rev_{\mathcal{G}}(\mathcal{T}^*) = min_{\mathcal{T}}\, rev_{\mathcal{G}}(\mathcal{T})$$

where, $|non - leaves(i)|$ gives $|\mathcal{V}| - |leaves(i)|$.

This function also has the same properties as $cost_{\mathcal{G}}(\mathcal{T}) + rev_{\mathcal{G}}(\mathcal{T}) = |\mathcal{V}| \sum_{(i,j)\in\mathcal{E}} w_{ij}$.

For the rest of the paper, we will focus on the cost function proposed in Dasgupta (2016). This function was shown to have a number of intuitive properties by Cohen-Addad et al. (2017). We consider the edge weights denoting the similarity between two end points, i.e.,

$$cost_{\mathcal{G}}(\mathcal{T}) = \sum_{(i,j)\in\mathcal{E}} w_{ij}|leaves(i \vee j)|$$

In Dasgupta (2016) it was shown that by doing a reduction from the **not-all-equal-SAT**, it was shown that the proposed cost function is NP-Hard to optimize. In the same paper, he proposed a natural greedy criterion to create the hierarchy by creating a split at every node that minimizes

$$\frac{E(S,\overline{S})}{|S||\overline{S}|}$$

where, $S, \overline{S}$ are the two partitions and $S \cup \overline{S} = \mathcal{V}$, $E(S,\overline{S}) = \sum_{i\in S, j\in\overline{S}} w_i j$.

This quantity is also known as **sparsest cut** which will be discussed further in the Section 2.0.2.

2.0.2 Sparsest Cut

In this section we provide some definitions about the sparsest cut, conductance and expansion from Trevisan (2013).

Definition 2.1. (Sparsest Cut) Let $\mathcal{G} = (\mathcal{V},\mathcal{E})$ be a graph and let $(S,\overline{S})$ be a partition of the vertices (a cut). Then the sparsity of the cut is

$$\sigma(S) := \frac{|\mathcal{V}|^2 E(S,\overline{S})}{2|\mathcal{E}||S||\overline{S}|}$$

where $E(S,\overline{S})$ is the number of edges in $\mathcal{E}$ that have one endpoint in S and one endpoint in $\overline{S}$. The sparsest cut is, given a graph, to find the set of minimal sparsity. The sparsity of a graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ is

$$\sigma(\mathcal{G}) = min_{S \subseteq V} \sigma(S)$$

It is also common to define $\sigma(S) := \frac{E(S,\overline{S})}{|S||\overline{S}|}$ as $\frac{|\mathcal{V}|^2}{|\mathcal{E}|}$ is a fixed quantity for a graph.

Definition 2.2. (Expansion) Let $\mathcal{G} = (\mathcal{V},\mathcal{E})$ be a graph and let $(S,\overline{S})$ be a partition of the vertices (a cut). For a d-regular graph, the expansion of the cut is

$$\phi(S) := \frac{E(S,\overline{S})}{d \cdot min(|S|,|\overline{S}|)}$$

where $E(S,\overline{S})$ is the number of edges in $\mathcal{E}$ that have one endpoint in S and one endpoint in $\overline{S}$. The expansion of the cut is, given a graph, to find the set of minimal expansion. The expansion of a graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ is

$$\phi(\mathcal{G}) = min_{S \subseteq V} \phi(S)$$

For every regular graph, if $|S| \leq \frac{|\mathcal{V}|}{2}$

$$\frac{1}{2}\sigma(S) \leq \phi(S) \leq \sigma(S)$$

Since $\phi(S) = \phi(\overline{S})$ and $\sigma(S) = \sigma(\overline{S})$,

$$\frac{1}{2}\sigma(G) \leq \phi(G) \leq \sigma(G)$$

Definition 2.3. (Conductance) Let $\mathcal{G} = (\mathcal{V},\mathcal{E},w)$ be a weighted graph and let $(S,\overline{S})$ be a partition of the vertices (a cut). The conductance of the cut is

$$\gamma(S) := \frac{E(S,\overline{S})}{\cdot min(vol(S), vol(\overline{S})}$$

where $E(S,\overline{S})$ is the sum of weight of edges in $\mathcal{E}$ that have one endpoint in S and one endpoint in $\overline{S}$. and $vol(S)$ is the sum of weights of edges with both end points in S. The conductance of the cut is, given a graph, to find the set of minimal conductance. The expansion of a graph $\mathcal{G} = (\mathcal{V},\mathcal{E}), w$ is

$$\gamma(\mathcal{G}) = min_{S \subseteq V} \gamma(S)$$

2.0.3 Laplacian, Eigenvalues and Expansion

In this section, we provide some background on Laplacians, eigenvalues and the Cheeger Inequality which will be the background for a large part of our first work.

Definition 2.4. If $\mathcal{G}$ is a graph, with adjacency matrix $\mathcal{A}$ and D is a diagonal matrix with entry $D_{ii} = \sum_j A_{ij}$, then the Laplacian is defined as $\mathcal{L} = D - \mathcal{A}$. When the degrees are fairly irregular, it is common to work with the normalized Laplacian matrix, $\mathcal{L} = I - D^{-1/2}\mathcal{A}D^{-1/2}$. We deal with $D^{-1/2}\mathcal{A}D^{-1/2}$ instead of $\mathcal{L}$.

Theorem 2.5. *For a graph $\mathcal{G}$ with n vertices, if $\lambda_1 \geq \ldots \geq \lambda_n$ are the eigenvalues of $D^{-1/2}\mathcal{A}D^{-1/2}$, then $1-\lambda_1 \leq 1-\lambda_2 \leq \ldots \leq 1-\lambda_n$ are the eigenvalues of the corresponding laplacian $\mathcal{L}$, and*

- $1 - \lambda_1 = 0$
- $1 - \lambda_2 = 0$ *iff $\mathcal{G}$ has two or more components*
- $1 - \lambda_n \leq 2$
- $1 - \lambda_n = 2$ *iff $\mathcal{G}$ is bi-partite.*

Theorem 2.6. *(Cheeger Inequality) Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a d-regular graph, $\frac{1}{d}\mathcal{A}$ be its' adjacency matrix, If $1 = \lambda_1 \geq .. \geq \lambda_n \geq -1$ are the eigenvalues of $\frac{1}{d}\mathcal{A}$, then, the Cheeger inequality states that*

$$\frac{1-\lambda_2}{2} \leq \frac{\sigma(\mathcal{G})}{2} \leq \phi(\mathcal{G}) \leq \sqrt{1-\lambda_2} \tag{2.2}$$

Corollary 2.7. *If $\mathcal{G}$ is an irregular graph, the Cheeger inequality of Equation 2.2 still holds.*

2.0.4 Power Method

The power method Trevisan (2013) is an algorithm that approximates the largest eigenvalue and the corresponding vector of a given symmetric PSD matrix. The working of the algorithm is elaborated in Algorithm 1.

To find the second largest eigenvalue and the corresponding eigenvector, we simply orthogonalize the vector $\mathbf{x_0}$ after step 3 of Algorithm 1 as $\mathbf{x_0} \leftarrow \mathbf{x_0} - \mathbf{v_1} < \mathbf{x_0}, \mathbf{v_1} >$ where v_1 is the eigenvector corresponding to the largest eigenvalue. We quote the following result about the second largest eigenvalue from Trevisan (2013)

Algorithm 1 Power Method

```
1: procedure PowerMethod(PSD Matrix $M$, int $k$)
2:     Let $M \in \mathbb{R}^{n\times n}$
3:     $\mathbf{x_0} \leftarrow -1, 1^n$
4:     for $i = 1$ to $k$ do
5:         $\mathbf{x_i} \leftarrow M\mathbf{x_{i-1}}$
6:     return $\mathbf{x_k}$
```

Algorithm 2 Power Method to Find Second Largest Eigenvector

```
1: procedure PowerMethod2(PSD Matrix $M$, int $k$, vector $\mathbf{v_1}$)
2:     Let $M \in \mathbb{R}^{n\times n}$
3:     $\mathbf{x_0} \leftarrow \{-1, 1\}^n$
4:     $\mathbf{x_0} \leftarrow \mathbf{x_0} - \mathbf{x_0^T v_1 v_1}$
5:     for $i = 1$ to $k$ do
6:         $\mathbf{x_i} \leftarrow M\mathbf{x_{i-1}}$
7:     return $\mathbf{x_k}$
```

Theorem 2.8. *For every PSD matrix M, positive integer k and parameter $\epsilon > 0$, if $\mathbf{v_1}$ is a length-1 eigenvector of the largest eigenvalue of M, then with probability $\geq 3/16$ over the choice of $\mathbf{x_0}$, the power method as described in Algorithm 2, outputs a vector $\mathbf{x_k} \perp \mathbf{v_1}$ such that*

$$\frac{\mathbf{x_k}^T M \mathbf{x_k}}{\mathbf{x_k}^T \mathbf{x_k}} \geq \lambda_2 (1-\epsilon) \frac{1}{1 + 4n(1-\epsilon)^{2k}} \tag{2.3}$$

where λ_2 is the second largest eigenvalue of M, counting multiplicities.

Related Work

In this chapter, we survey the related work.

3.0.1 Cost function for Hierarchical Clustering

In most learning algorithms, we try to optimize a certain cost function. Prior to 2015, no cost function existed for hierarchical clustering. Dasgupta (2016) proposed the cost function for a weighted undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, w)$.

The proposed cost function has some desirable properties as described below.

- If the graph $\mathcal{G}$ is disconnected, the components are separated out earlier while building the hierarchy.
- Vertices which are closer to each other are separated lower down in the tree. If they are split high up, they incur a high penalty.
- Unit weighted cliques have the same cost for any tree.

A "dual" for the cost function was proposed in Bateni et al. (2017). They termed it as **revenue**. The sum of the cost and the revenue is a constant for a given graph. For both these functions, the graph considered is a complete graph with weights on every edge.

Both these cost functions are NP-Hard to optimize. It was also shown in Dasgupta (2016) that a natural greedy criterion is the minimize the **sparsest cut** ratio at every internal node which is elaborated in Section 2.0.2.

Cohen-Addad et al. (2017) analyze the Dasgupta (2016) cost function and define the necessary properties that a good cost function for hierarchical clustering should follow. They claim that a good cost function must i) have the same cost for any dendrogram for unit weighted cliques, ii) must have the same cost if the left and right children are swapped in the dendrogram, and iii) the cost is non decreasing if a node is added to the graph. Monath et al. (2017) also propose a probabilistic and continuous version of the cost function.

3.0.2 Algorithms

Hierarchical clustering algorithms can be either **agglomerative** or **divisive**. Agglomerative algorithms are essentially bottom-up algorithms while divisive algorithms are top-down. Hierarchical Agglomerative Clustering algorithms treat each data point to be in a cluster on its own and merge these clusters based on some criteria. In Hierarchical Divisive Clustering algorithms, all points are in in one cluster and they are split based on some criteria. The order of merges or splits define the hierarchy.

3.0.2.1 Agglomerative Merging Criteria

Some well known merging criteria are:

1. **Single Linkage:** Each point is treated as a cluster and the merges are based on minimizing the **minimum** distance between the clusters.
2. **Average Linkage:** Each point is treated as a cluster and the merges are based on minimizing the **average** distance between the clusters.
3. **Complete Linkage:** Each point is treated as a cluster and the merges are based on minimizing the **maximum** distance between the clusters.
4. **Ward Linkage:** Each point is treated as a cluster and the merges are based on minimizing the **variance** inside the merged cluster.

When the clusters in question have more than one point, the distance computed is pairwise distance between their respective points. All these algorithms have time complexity $O(n^2)$ and space complexity $O(n)$ where n is the number of points.

3.0.2.2 Divisive Splitting Criteria

A splitting criteria popularly used for divisive hierarchical clustering is:

1. **k-means:** All points are in a single cluster and are split into k sub-clusters based on the k-means algorithm. This algorithm has a time complexity of $O(nk)$ per split when there are n points in the cluster and we want to split them into k clusters.

3.0.2.3 Recent advancements

In Dasgupta (2016), it was shown that, if an algorithm with an approximation of $O(\alpha_n)$ to the sparsest cut is used for the hierarchical clustering, an $O(\alpha_n \log(n))$ approximation to the cost function is obtained. This was later improved by Charikar and Chatziafratis (2017) to show that the approximation to the cost function is $O(\alpha_n)$ and not $O(\alpha_n \log(n))$.

The two best known approximation algorithms for the sparsest cut are the Leighton Rao algorithm (LR) and the Arora Rao Vazirani algorithm (ARV).

The LR (Leighton and Rao (1999)) algorithm gives an $\alpha_n = O(\log(n))$ approximation to the sparsest cut. The algorithm makes use of multi-commodity flow to find a solution for the sparsest cut problem. Though the proposed algorithm is has polynomial time complexity, it is $O(n^2 \log(n) + n^3)$.

The ARV (Arora et al. (2009)) algorithm exploited semi-definite programming to propose an $O(\sqrt{\log(n)})$ approximation to the sparsest cut. Though it gives a good approximation, the time complexity of the algorithm is $O(n^4)$.

The main drawback of these algorithms are:

- Time to build the clustering using the LR and ARV are extremely high. Most real applications have enormous amounts of data and limited computational power which makes it impractical.
- Traversing down the hierarchy to identify clusters is not possible with the LR or ARV algorithms. For a new query, it is unclear which branch of the dendrogram should be taken.
- Extending the LR or ARV algorithm for the Online hierarchical clustering problem is not feasible.

3.0.3 Online Algorithms

In many settings, we may not have access to the whole data at once, or we may not be able to process all the data in one shot. In such cases, algorithms are designed to build a model in a piece by piece, serial fashion. In some cases, the data stream may be arriving in a possibly never ending stream and a good-quality solution needs to be maintained.

3.0.3.1 Hierarchical Clustering

For the case where we need to maintain a hierarchical clustering and data is arriving in a possibly never ending stream, no algorithm with provable theoretical guarantees have been developed.

The most recent work on Online Hierarchical Clustering was proposed by Kobren et al. (2017). If a new point (say v) is to be added, v is added as a sibling to u where $u = argmin_x||v - x||_2$. This is achieved through maintaining a *bounding box* at each internal node of the hierarchy. This *bounding box* maintains the maximum and the minimum distance for each dimension. This addition of a point may separate two closer nodes which is rectified through a series of rotations which are based on comparisons between a node and its' aunt. They also promote balanced, shallow trees over deep trees due to the high insertion and rotation times.

To target the online hierarchical clustering problem, we identify the sub-problem of online updation of eigenvectors under insertion and deletion operations.

3.0.3.2 Streaming Eigenvector Updates

When we have a never ending stream of incoming points, calculating and maintaining the top-k right singular vectors has been studied before. Mitliagkas et al. (2013) proposed an algorithm to maintain the top-k singular vectors and singular values. The proposed algorithm performed one step of the power method to calculate the singular values for every data point. They study the performance of this algorithm on the *spiked covariance model*. It is shown that the vector(s) they estimate are off by a small error term, E, bounded by $||E||_2 = \epsilon$. It was also the first work which maintained the top-k singular values with $O(kd)$ space where d is the number of dimensions in the data and had provable theoretic bounds.

There are other works which focus on the online singular vectors and online Principal Component Analysis (PCA) problem. Online PCA with regret minimization has been considered in Garber et al. (2015). Sub-sampling and dimensionality reduction has also been considered in Clarkson and Woodruff (2009); Halko et al. (2011).

The more interesting problem is the update of singular vectors if the stream consists of both insertions and deletions of points as, in many real-life scenarios, points may get added as well as deleted. One such example is that of social networks. People, comments, images may get added and they may also get deleted. Till date, there are no theoretical guarantees for this problem.

Efficient Hierarchical Clustering

Overview

In this chapter, we present a scalable, divisive hierarchical clustering algorithm with guaranteed worst case bounds for the cost function given in Equation 2.1. The divisive hierarchical clustering algorithms starts by taking all points to be in the same cluster and recursively splits them into two sub-clusters based on some `SPLITTING RULE`. We consider binary splits, as one of the properties of the cost function is that the optimal tree is binary in nature. We also show the performance of one splitting rule on a simple stochastic block model.

We then empirically demonstrate two tasks which make use of the hierarchy. Firstly, we use the hierarchy in order to label new content using a variant of near-neighbor classification. Secondly, we use the hierarchy for identification of new anomalous content. We perform experiments on three open source datasets as well as on a dataset obtained from one of the largest social networking sites (LinkedIn). Algorithm 3 outlines the pseudocode for the proposed hierarchical clustering and the querying mechanism used for near-neighbor classification.

4.0.1 Proposed Splitting Methods

We propose four strategies that we use for the `SPLITTINGRULE`. Let t denote a generic tree-node, in which we have the points $V_t = \{v_1, \ldots, v_\ell\}$. We describe the following techniques to compute the splitting hyperplane and the split point, (h_t, s_t). Note that the hierarchy is being created in a batch mode, i.e., where all data is available while building the model. However, maintaining the splitting hyperplane and split point for each node enable us to query the hierarchy in real time with individual query points.

`Random Hyperplane (RP)`. We sample h_t uniformly at random from $S^{d-1} = \{x \in \mathbb{R}^d, \|x\|_2 = 1\}$. Note that this can be done by sampling $y \sim \mathcal{N}(0, I_{d\times d})$ and returning $x = \frac{y}{\|y\|}$. Define

Algorithm 3 Hierarchical Clustering

```
1: procedure MakeTree($V$)
2:     if $|V| = 1$ then
3:         return leaf containing $|V|$
4:     $h_r, s_r \leftarrow$splittingRule($V$)
5:     $S \leftarrow \{x \in V, h_r \cdot x \leq s_r\}$
6:     LeftTree $\leftarrow$ MakeTree(S)
7:     RightTree $\leftarrow$ MakeTree($V \setminus S$)
8:     return [$h_r$, $s_r$, LeftTree, RightTree]
9: procedure Query($T, x$)
10:    $r$ =root($T$).
11:    if $T$ contains less than $B$ points or $T$ has no children then
12:        return all of them.
13:    else
14:        $h_r, s_r$ be the splitting hyperplane and location for $r$.
15:        if  $x \cdot h_r \leq s_r$ then    Query left child of $r$ with $x$
16:        else   Query right child of $r$ with $x$
17: procedure Classify($T, x, k$)
18:    $S \leftarrow$Query($T, x$)
19:    $C \leftarrow k$ nearest neighbors to $x$ from $S$.
20:    $l_x \leftarrow$ majority vote of the labels of $C$.
21:    return $l_x$
```

Algorithm 4 Eigenvector partitioning

```
1: procedure splittingrule($V$)
2:     if $|V| = 1$ then
3:         return leaf containing $|V|$
4:     Let $A \in \mathbb{R}^{\ell \times d}$ be the matrix formed by the points.
5:     $d = Ae$ where $e$ is vector of all ones, $D = \mathsf{diag}(d)$.
6:     $\tilde{A} = D^{-1/2}A$.
7:     $h_t \in \mathbb{R}^d$ be the second largest right singular vector of $\tilde{A}$.
8:     Create $X = \{\tilde{A}_{i*} \cdot h_t,\ \forall i \in [\ell]\}$.
9:     Consider the sorted sequence $X = \langle X_1, \ldots, X_\ell \rangle$. Find the set $S_j = \{k \in \ell, X_k \leq$
   $X_j\}$ that has smallest $\gamma(S_j)$.
10:    return $(h_t, X_j)$
```

$X = \{h_t \cdot v_i, i = 1, \ldots, \ell\}$. The splitting point s_t is chosen as the median of X. The pseudocode is presented in Algorithm 5.

`Eigenvector (EV)`: Let $D \in \mathbb{R}^{\ell \times \ell}$ be the diagonal matrix with $D_{ii} = \sum_j A_{ij}$. We calculate the normalized feature-feature matrix $F = A^t D^{-1} A \in \mathbb{R}^{d \times d}$. Let x_2 be the eigenvector corresponding to the second largest eigenvalue of F. We define $u_2 = D^{-1/2}Ax_2/\|D^{-1/2}Ax_2\|_2$. Note that u_2 is the second largest right singular vector of $D^{-1/2}AA^tD^{-1/2}$. The coordinates of u_2 are sorted. We find the value in u_2, say X_j, and return the set $S_j = \{k \in \ell, X_k \leq X_j\}$

Algorithm 5 Random Partitioning

```
1: procedure SEPARATINGHYPERPLANE(V)
2:     $h_t \sim N(0, I_{dxd})$, $s_t \leftarrow 0$.
3:     return $h_t/\|h_t\|_2, s_t$.
```

that has the smallest $\gamma(S_j)$. The pair (x_2, X_j) is returned as the splitting hyperplane and threshold. The pseudocode is presented in Algorithm 4.

`Approximate Eigenvector (AEV):` Inspired by McCartin-Lim et al. (2012), we use an approximate eigenvector, rather than the exact one. In Algorithm 6, we present the pseudocode of how we use power iterations to estimate the eigenvector corresponding to the second largest eigenvalue of the normalized covariance matrix.

Algorithm 6 Approximate Eigenvector

```
1: procedure SPLITTINGRULE(V)
2:     Let $A \in \mathbb{R}^{\ell \times d}$ be the matrix formed by $\{v_1, \ldots, v_\ell\}$.
3:     $v \leftarrow$ random vector in $\mathbb{R}^d$.
4:     $d = Ae$ where $e$ is vector of all ones, $D = \mathsf{diag}(d)$.
5:     $\tilde{A} = D^{-1/2}A$.
6:     for $i \in 1 \ldots M_1$ do                                  ▷ $M_1$ is $O(\log(n)/\epsilon)$.
7:         $v \leftarrow \tilde{A}^t(\tilde{A}v)$, $v \leftarrow v/\|v\|_2$
8:     $u \leftarrow$ random vector in $\mathbb{R}^d$.
9:     for $i \in 1 \ldots M_2$ do                                  ▷ $M_2$ is $O(\log(n)/\epsilon)$.
10:        $u \leftarrow (I - vv^t)\tilde{A}^t\tilde{A}u$.
11:        $u \leftarrow u/\|u\|_2$
12:    Create $X = \{\tilde{A}_{i*} \cdot u, \ \forall i \in [\ell]\}$.
13:    Consider the sorted sequence $X = \langle X_1, \ldots, X_\ell \rangle$. Find the set $S_j = \{k \in \ell, X_k \leq$
   $X_j\}$ that has smallest $\gamma(S_j)$.
14:    return $(u, X_j)$.
```

`2-means:` For this strategy, we find a 2-means clustering of the data and define h_t to be the hyperplane that is equally distant from the two centers. The threshold s_t is set to be 0, and the two clusters are returned as the partitions.

Algorithm 7 2-means

```
1: procedure SEPARATINGHYPERPLANE(V)
2:     Let $A \in \mathbb{R}^{\ell \times d}$ be the matrix formed by $\{v_1, \ldots, v_\ell\}$.
3:     Use k-means++ Arthur and Vassilvitskii (2007) to get a bi-partitioning of $V$.
4:     Let $c_1$ and $c_2$ be the two cluster centers.
5:     return $h_t = 2(c_1 - c_2)$, $s_t \leftarrow \|c_1^2\| - \|c_2\|^2$.
```

4.0.2 Theoretical Guarantees

4.0.2.1 Hierarchical Clustering using Eigenvectors

We first show that our partitioning strategy of using the second left singular vector of the matrix $V \in \mathbb{R}^{\ell \times d}$ gives an approximation to the objective given in Equation 2.1. In order to do this, we first quote a particular lemma from Dasgupta (2016) that characterizes the structure of any tree, in particular the optimal tree.

Lemma 4.1. *(Lemma 11 of Dasgupta (2016)): Pick any binary tree, T on V. There exists a partition A, B of V, where $\frac{|V|}{3} \leq |A|, |B| \leq \frac{2|V|}{3}$ such that $\frac{C(A,B)}{|A||B|} < \frac{27}{4|V|^3} cost_V(T)$.*

We now show a bound on the cost of the tree formed by using an exact eigenvector as the splitting subroutine in Algorithm 3.

Theorem 4.2. *Let V contain n points. The point-point similarities are encoded as $C : V \times V \to [0,1]$. Let tree T^* be a minimizer of $cost_V(\cdot)$ and let T be the tree returned by using the second largest eigenvector of $D^{-1/2}VV^tD^{-1/2}$ as a splitting rule. Then $cost_V(T) \leq cn\ \log(n)\ \sqrt{cost_V(T^*)}$ for some $c \leq \Delta_{max}\sqrt{\frac{27}{\Delta_{min}}}$.*

Proof. We prove this by induction. The base case, when $n = 1$, the $cost(T) = 0$. Let us assume that the claim holds for graphs with at most $n-1$ nodes.

Suppose the optimal tree is T^* and consider the root of T^*, with the entire set of points V. From Lemma 4.1, we can say that there exists a partition (A, B) of V such that $\frac{C(A,B)}{|A||B|}$ will be bounded by $\frac{27cost(T^*)}{4n^3}$. When an exact eigenvector is used as the splitting strategy, for a node t, if V is the data matrix which contains all the points as the rows, and h_t is the second right singular vector of $D^{-1/2}V$, then $D^{-1/2}V \cdot h_t$ is the (scaled) second left singular vector of $D^{-1/2}V$, as well as the second largest eigenvector of $D^{-1/2}VV^tD^{-1/2} = D^{-1/2}CD^{-1/2}$. Let (A, B) (say $|A| \leq |A \cup B|/2$) denote the spectral cut found. Using Lemma 2.2, we know that $\gamma(A) \leq O(\sqrt{\gamma(C)})$, where $\gamma(C)$ denotes the conductance of C. Given that the degrees of C all lie in the range $[\Delta_{min}, \Delta_{max}]$, we know that $\phi(C) \geq \Delta_{min} \cdot \gamma(C)$, and $\phi(A) \leq \Delta_{max} \cdot \gamma(A)$. Hence,

$$\phi(A) \leq \Delta_{max} \cdot \gamma(A), \quad \phi(A) \leq \Delta_{max}\sqrt{\gamma(C)} \leq \frac{\Delta_{max}}{\sqrt{\Delta_{min}}}\sqrt{\phi(C)}$$

Note that for any partitioning (S, S^c), $\frac{1}{n}\phi(S) \leq \frac{C(S,S^c)}{|S||S^c|} \leq \frac{2}{n}\phi(S)$. Suppose (A', B') is the split guaranteed by Lemma 4.1. Recall that (A, B) is the split returned by the eigenvector.

Let $\theta = \frac{\Delta_{max}}{\sqrt{\Delta_{min}}}$. Then, using Lemma 4.1

$$\begin{aligned}\frac{C(A,B)}{|A||B|} &\leq \frac{2\phi(A,B)}{n} \leq \frac{\theta\sqrt{\phi(A',B')}}{n} \leq \frac{\theta}{n}\sqrt{n\frac{C(A',B')}{|A'||B'|}} \\ &\leq \frac{\theta}{\sqrt{n}}\sqrt{\frac{27}{4|V|^3}cost_V(T^*)} = \theta\sqrt{\frac{27cost_V(T^*)}{4|V|^4}}\end{aligned}$$

$$\frac{nC(A,B)}{|A||B|} \leq \frac{C(A,B)}{|A|n/2} \leq \sqrt{\frac{27}{4|V|^3}cost_V(T^*)}$$

Let us consider trees T^*_A, T_A and T^*_B, T_B which are the trees T^* and T restricted to the nodes in A and B respectively. Recall that T^* is the optimal tree.

So, Let $0 < |A| = pn \leq \frac{n}{2}$. Let $|A| \leq |B| = (1-p)n$. Since $|A|, |B| < n$, we apply the induction hypothesis.

$$\begin{aligned}cost_A(T_A) + cost_B(T_B) &\leq n\theta\sqrt{\frac{27}{4}}\big(p\sqrt{cost_A(T^*_A)}\log(pn) + (1-p)\sqrt{cost_B(T^*_B)}\log((1-p))\big) \\ &\leq n\theta\sqrt{\frac{27cost_V(T^*)}{4}}(p\log p + (1-p)\log(1-p) + \log n)\end{aligned}$$

Now, $cost_V(T) \leq nC(A,B) + cost(A) + cost(B) \leq n\theta\frac{\sqrt{27cost_V(T^*)}}{2}(p(1-p) + p\log p + (1-p)\log(1-p) + \log n) \leq n\log n\theta\frac{\sqrt{27cost(T^*)}}{2}$. □

4.0.2.2 Approximation to the Planted Partition

In this section, we show that the bound that spectral partitioning gives us is significantly better if the data does have a nice structure. Traditionally, the merits of spectral partitioning have been theoretically demonstrated using stochastic block models or mixture models. We consider a similar setting. Note that this is an idealized setting, only meant to demonstrate that spectral partitioning based splitting can give good approximation to the hierarchical cost under simple generative models.

Assume that the similarity matrix C is generated using a (n,p,q)-planted partition model with two equisized partitions. C is constructed through the following generative process: for every pair of distinct nodes, i,j, an edge is added with probability p, if i and j belong to the same partition and q otherwise. All these $\binom{n}{2}$ decisions are independent. We use the following result that shows that given C, the eigenvector based sparsest cut algorithm would find a partition close to the ground truth.

Lemma 4.3. *From McSherry (2001): In the planted partition model, the spectral sparsest cut algorithm mis-clusters at most a constant, k number of points where*

$$k = \frac{36p}{(p-q)^2}$$

Lemma 4.4. *Let $\mathcal{G}(V,E)$ be an unweighted undirected clique on n points, and let $\mathcal{G}^{'}(V^{'},E^{'})$ be another unweighted undirected clique on $n+k$ nodes, then $cost_{\mathcal{G}'}(\cdot) = (c)cost_{\mathcal{G}}(\cdot)$ for $n >> 1$ and $k \leq n$ and c is some constant.*

Proof. From Dasgupta (2016), we know that $cost_{\mathcal{G}}(T) = \frac{1}{3}(n^3 - n)$ and $cost_{\mathcal{G}'}(T) = \frac{1}{3}((n+k)^3 - n - k)$, and every tree has the same cost.

Now,

$$\begin{aligned}
\frac{cost_{\mathcal{G}'}(T)}{cost_{\mathcal{G}}(T)} &= \frac{\frac{1}{3}((n+k)^3 - n - k)}{\frac{1}{3}(n^3-n)} && = \frac{(n^3 + k^3 + 3nk(n+k)) - n - k}{n^3 - n} \\
&= \frac{n^3(1 + \frac{k^3}{n^3} + 3\frac{k(n+k)}{n^2} - \frac{1}{n^2} - \frac{k}{n^3})}{n^3(1-\frac{1}{n^2})} && = \frac{(1+1+3)}{1} \qquad (Since\, k \leq n) \\
&= (1+\epsilon)cost_{\mathcal{G}'}(T) = (c)cost_{\mathcal{G}}(T)
\end{aligned}$$

□

Theorem 4.5. *Let $\mathcal{G}(V,E)$ be an unweighted undirected graph generated from planted partition model. Let C contain adjacency matrix on V, and let T^* be a minimizer of $cost_V(\cdot)$ and if T is the tree returtned by using the splitting rule of Algorithm 4 inside Algorithm 3, then $cost_V(T) \leq c_1 cost_V(T^*)$ where $c_1 = max(1 + \frac{72p}{qn(p-q)}, c)$ and c is a constant.*

Proof. Let d be the expected degree of $\mathcal{G}$. Then, $d = (p+q)\frac{n}{2} - p \approx (p+q)\frac{n}{2}$. Since $\mathcal{G}$ is an almost regular graph, we do not perform the degree normalization for this theorem.

We quote a small result on the first eigenvector of C and its corresponding $\mathcal{L} = 1 - D^{-1/2}CD^{-1/2}$ where D is the diagonal matrix with $d_{ii} = d$. The first eigenvectors of $C, \mathcal{L}$ are given as

$C\mathbf{1} = \frac{n}{2}(p+q)\mathbf{1}$ and $\mathcal{L}\mathbf{v} = 1\mathbf{v} \qquad (\mathbf{v} = D^{-1/2}\mathbf{1})$

Let T be the tree that we get using Algorithm 4 inside Algorithm 3, and T^* be the optimal tree. For the planted partitions, the expected optimal cost is given by

$$\begin{aligned}\mathbb{E}cost_V(T^*) &= \sum_{i,j} \mathbb{E}w_{ij}|leaves(T^*[i \vee j])| = \sum_{i,j} Prob[edge\ (i,j)]\ |leaves(T^*[i \vee j])| \\ &= \frac{qn^3}{4} + 2\frac{p}{3}(\frac{n^3}{8} - \frac{n}{2})\end{aligned}$$

Using Lemma 4.3, we misclassify at most $k = \frac{36p}{(p-q)^2}$ vertices, the cost of the top split in T is

$$\mathbb{E}cost_V(T) \leq \frac{qn^3}{4} + \frac{36p}{2(p-q)^2}pn^2 - \frac{36p}{2(p-q)^2}qn^2 = \frac{qn^3}{4} + \frac{36pn^2}{2(p-q)} = \frac{qn^3}{4}(1 + \frac{72p}{qn(p-q)})$$

For the future splits, we make two assumptions

1. Some edges will have weight $C_{ij} < p$, and
2. The size of both the partitions will be in the range $[\frac{n}{2} - k, \frac{n}{2} + k]$, and $k = \frac{36p}{(p-q)^2}$.

Also, since we have two equi-sized clusters, $k \leq \frac{n}{2}$. Now, let the two partitions identified by A, B. Using Lemma 4.4,

$$\mathbb{E}cost_A(T) + \mathbb{E}cost_B(T) \leq \frac{2p}{3}c(\frac{n^3}{8} - \frac{n}{2})$$

Thus, the total cost can be bounded by

$$\begin{aligned}\mathbb{E}cost_V(T) &\leq \frac{qn^3}{4}(1 + \frac{72p}{qn(p-q)}) + c\frac{2p}{3}(\frac{n^3}{8} - \frac{n}{2}) \\ &\leq c_1(\frac{qn^3}{4} + \frac{2p}{3}(\frac{n^3}{8} - \frac{n}{2})) \qquad (where\ c_1 = max(1 + \frac{72p}{qn(p-q)}, c)) \\ &= c_1\mathbb{E}cost_V(T^*)\end{aligned}$$

□

While we state the above result in the simple planted partition model, it is possible to come up with an analogous statement for other variants, e.g. when the points come from a Gaussian mixture model.

4.0.2.3 Hierarchical Clustering using Approximate Eigenvector

Algorithm 2 starts with a random vector and applies power iterations to the vector. We thus have a close approximation to the first eigenvector, using which we get a close approximation to the second, again by applying power iteration. We first quote a result that shows that assuming we have approximated the first singular vector v_1, the second singular vector of $\tilde{C}$ is also well approximated.

Lemma 4.6. *Consider a matrix $M = I - L = D^{-1/2}XX^tD^{-1/2}$ and $0 = \lambda_1 \leq ... \leq \lambda_n \leq 2$ are the eigenvalues of L. By running power iteration on $M' = X^TD^{-1}X$, we get $\frac{x^tLx}{x^tx} \leq \lambda_2 + \epsilon$*

Proof. If $0 = \lambda_1 \leq \ldots \leq \lambda_n \leq 2$ are the eigenvalues of L, $1 = 1 - \lambda_1 \geq \ldots \geq 1 - \lambda_n \geq -1$ are the eigenvalues of M.

Now, $M = D^{-1/2}XX^tD^{-1/2} = (X^tD^{-1/2})^t(X^tD^{-1/2})$. Note that M is a PSD matrix, and let $M = Y^tY$, where $Y = (A^tD^{-1/2})$. Also, eigenvalues for M will be bounded in the range $[0, 1]$. Also, eigenvalues for Y^tY are the same as the eigenvalues for YY^t. And, $YY^t = X^tD^{-1}X = M'$. This also implies that, since we use $C = XX^t$, the eigenvalues of L are also in the range $[0, 1]$.

Thus, we prove the inequality 4.6 using M instead of M'. Using Theorem 2.8, using power method and setting $k = O(\frac{\log n}{\epsilon})$

$$\frac{x^TMx}{x^Tx} \geq (1 - \lambda_2)(1 - \epsilon)$$
$$\frac{x^TLx}{x^Tx} \leq \lambda_2 + \epsilon$$

□

Corollary 4.7. *Suppose $\frac{x^TLx}{x^Tx} \leq \lambda_2 + \epsilon \leq 2\phi(G) + \epsilon$. It also follows that $\phi(S) \leq \sqrt{2(\lambda_2 + \epsilon)} \leq \sqrt{4(\phi(G) + \epsilon)}$. I.e. we can find a cut of expansion $O(\sqrt{\phi(G) + \epsilon})$*

Theorem 4.8. *Let V contain n points. The point-point similarities are encoded as $C : V \times V \to [0, 1]$. Let tree T^* be a minimizer of $cost_V(\cdot)$ and let T be the tree returned by using the Algorithm 3 with the splitting rule defined in Algorithm 6. Then $cost_V(T) \leq cn\ \log(n)\ \sqrt{cost_V(T^*) + \epsilon}$ for some $c \leq \Delta_{max}\sqrt{\frac{27}{\Delta_{min}}}$ and ϵ is a small quantity.*

Proof. The proof is similar to that of Theorem 4.2 and is provided here for completeness.

We prove this by induction. The base case, when $n = 1$, the $cost(T) = 0$. Let us assume that the claim holds for graphs with at most $n - 1$ nodes.

Suppose the optimal tree is T^* and consider the root of T^*, with the entire set of points V. From Lemma 4.1, we can say that there exists a partition (A, B) of V such that $\frac{C(A,B)}{|A||B|}$ will be bounded by $\frac{27cost(T^*)}{4n^3}$. When an exact eigenvector is used as the splitting strategy, for a node t, if V is the data matrix which contains all the points as the rows, and h_t is the second right singular vector of $D^{-1/2}V$, then $D^{-1/2}V \cdot h_t$ is the (scaled) second left singular vector of $D^{-1/2}V$, as well as the second largest eigenvector of $D^{-1/2}VV^tD^{-1/2} = D^{-1/2}CD^{-1/2}$. Let (A, B) (say $|A| \leq |A \cup B|/2$) denote the spectral cut found. Using Lemma 2.2, we know that $\gamma(A) \leq O(\sqrt{\gamma(C)})$, where $\gamma(C)$ denotes the conductance of C. Given that the degrees of C all lie in the range $[\Delta_{min}, \Delta_{max}]$, we know that $\phi(C) \geq \Delta_{min} \cdot \gamma(C)$, and $\phi(A) \leq \Delta_{max} \cdot \gamma(A)$. Hence,

$$\phi(A) \leq \Delta_{max} \cdot \gamma(A), \quad \phi(A) \leq \Delta_{max}\sqrt{\gamma(C)} \leq \frac{\Delta_{max}}{\sqrt{\Delta_{min}}}\sqrt{\phi(C)}$$

Note that for any partitioning (S, S^c), $\frac{1}{n}\phi(S) \leq \frac{C(S,S^c)}{|S||S^c|} \leq \frac{2}{n}\phi(S)$. Suppose (A', B') is the split guaranteed by Lemma 4.1. Now, since we are using approximate eigenvector, then (A, B) is the split returned. Let $\theta = \frac{\Delta_{max}}{\sqrt{\Delta_{min}}}$. Then, using Lemm 4.1,

$$\begin{aligned}\frac{C(A,B)}{|A||B|} &\leq \frac{2\phi(A,B)}{n} \leq \frac{\theta\sqrt{\phi(A',B')}}{n} \leq \frac{\theta}{n}\sqrt{n\frac{C(A',B')}{|A'||B'|}} \\ &\leq \frac{\theta}{\sqrt{n}}\sqrt{\frac{27}{4|V|^3}cost_V(T^*)} = \theta\sqrt{\frac{27(cost_V(T^*))}{4|V|^4}}\end{aligned}$$

Therefore, using Lemma 4.7,

$$\frac{nC(A,B)}{|A||B|} \leq \frac{C(A,B)}{|A|n/2} \leq \sqrt{\frac{27}{4|V|^3}(cost_V(T^*) + \epsilon)}$$

Let us consider trees T_A^*, T_A and T_B^*, T_B which are the trees T^* and T restricted to the nodes in A and B respectively. Recall that T^* is the optimal tree.

So, Let $0 < |A| = pn \leq \frac{n}{2}$. Let $|A| \leq |B| = (1-p)n$. Since $|A|, |B| < n$, we apply the induction hypothesis.

$$\begin{aligned} cost_A(T_A) + cost_B(T_B) &\leq n\theta\sqrt{\frac{27}{4}}\big(p\sqrt{cost_A(T_A^*) + \epsilon}\log(pn) \\ &\quad + (1-p)\sqrt{cost_B(T_B^*) + \epsilon}\log((1-p))\big) \\ &\leq n\theta\sqrt{\frac{27(cost_V(T^*) + \epsilon)}{4}}(p\log p + (1-p)\log(1-p) + \log n) \end{aligned}$$

Now, $cost_V(T) \leq nC(A,B) + cost(A) + cost(B) \leq n\theta\frac{\sqrt{27cost_V(T^*)+\epsilon}}{2}(p(1-p) + p\log p + (1-p)\log(1-p) + \log n) \leq n\log n\theta\frac{\sqrt{27cost(T^*)+\epsilon}}{2}$. ☐

4.0.2.4 Hierarchical Clustering using Random Hyperplanes

For this strategy, we use a random hyperplane to partition the data. We now show a result that shows that using random hyperplanes to do a clustering has a provable worst case guarantee.

Theorem 4.9. *For any arbitrary dot product similarity matrix C, with all $0 \leq C_{ij} \leq 1$, the hierarchical clustering produced using a random partitioning has an approximation factor of $O(n)$.*

Proof. Let the angle between points i and j be denoted by $\theta_{ij} = \cos^{-1}(C_{ij})$. Given that the partitioning vector or equivalently, the hyperplane, has a direction which is uniformly chosen, and the points i and j are split iff they lie on different sides of the resulting hyperplane, at any given tree node that has both i and j, $P(i,j\ split) = \frac{\theta_{ij}}{\pi}$. and $P(i,j\ not\ split) = 1 - \frac{\theta_{ij}}{\pi}$. Hence,

$$\begin{aligned} P(i,j \text{ split at level } l) =& P(i,j \text{ not split till level } l-1) \cdot P(i,j \text{ split at level } l) \\ =& \left(1 - \frac{\theta_{ij}}{\pi}\right)^{(l-1)} \frac{\theta_{ij}}{\pi} \end{aligned}$$

If we use a median based splitting strategy, then each node at level l has $n/2^l$ leaves. Hence if the pair (i,j) are split at level l, then the contribution of the edge is $nC_{ij}/2^l$. Thus the expected cost using a median splitting strategy is:

$$\mathbb{E}\ cost_V(T) \leq \sum_{i,j} nC_{ij} \sum_{l=1}^{\log(n)} \left(1 - \frac{\theta_{ij}}{\pi}\right)^{(l-1)} \frac{\theta_{ij}}{\pi}\left(\frac{1}{2}\right)^l$$

The term inside the second summation is a GP.

$$\mathbb{E}\ cost_V(T) \leq \sum_{i,j} \frac{n}{2} C_{ij} \frac{\theta_{ij}}{\pi} \left(\frac{1 - \left(\frac{1}{2}\left(1 - \frac{\theta_{ij}}{\pi}\right)\right)^{(\log(n)-1)}}{1 - \frac{1}{2}\left(1 - \frac{\theta_{ij}}{\pi}\right)} \right) \leq \sum_{i,j} \frac{n}{2} C_{ij} \frac{\theta_{ij}}{\pi} 2 \leq \sum_{i,j} nC_{ij}$$

$$(As\ 0 \leq \theta_{ij} \leq \pi)$$

Now, we lower bound the optimal cost, $cost_V(T^*)$. There can be two cases, (1) $C_{ij} = 0$, and (2) $C_{ij} > 0$. For the second case $|leaves(T[i \vee j])| >= 2$. So, $cost_V(T^*) \geq 2\sum_{ij} C_{ij}$, where T^* is the optimal tree. Thus,

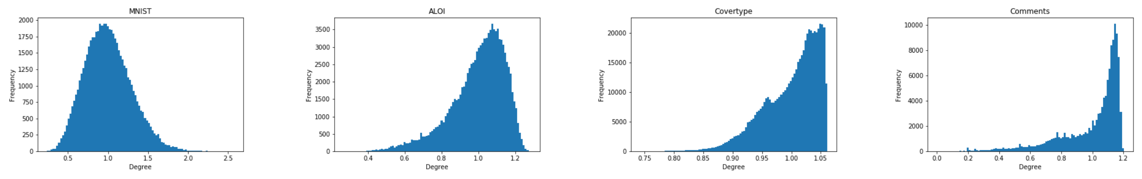
$$\frac{\mathbb{E}cost_V(T)}{cost_V(T^*)} \leq \frac{\sum_{i,j} nC_{ij}}{\sum_{i,j} 2C_{ij}} = \frac{n\sum_{i,j} C_{i,j}}{2\sum_{i,j} C_{i,j}} = \frac{n}{2}$$

□

4.0.3 Experimental Evaluation

In this section, we evaluate the performance of our proposed solutions, which will hereforth be referred to as Hierarchical Tree Based NN (`HTNN`), on four splitting rules against state of the art baselines for three open-source datasets and a real world dataset from the social networking site. The datasets used are `ALOI` from Rocha and Goldenstein (2014) (Dimensions = 128, Classes = 1k, `Train` size = 88k, `Test` size = 20k), `MNIST` from LeCun et al. (1998) (Dimensions = 784, Classes = 10, `Train` size = 60k, `Test` size = 10k), `Covertype` from Dheeru and Karra Taniskidou (2017) (Dimensions = 53, Classes = 7, `Train` size = 480k, `Test` size = 100k). The real world dataset is a `Comments` dataset which consists of textual data. Each word in the comment is converted to a 50-dimensional vector using pretrained GloVe embeddings from Pennington et al. (2014) and the comment is the average of these embeddings. The dataset consists of $\sim$ 100k `Train` comments and 20k `Test` comments. This dataset has 3 labels, namely, `Spam` (Content which is offensive, harmful, abusive or disruptive. These comments violate the Terms and Conditions of the networking site and are to be completely removed from the site), `Low-Quality` (Content which is irrelevant to the discussion or unappealing. Such content may receive a diminished visibility on the site. In our observations, `Spam` and `Low-Quality` categories contain very similar content with minor differences, which leads to a difficult classification problem), and `Clear` (Content which is okay to receive unrestricted distribution on the site).

Our implementation for benchmarking use the following changes to make the algorithms more efficient in practice. For a spectral partitioning with the desired worst case guarantees, as described in Algorithm 4, all possible sequence based partitions $\{S_j, = 1\ldots, n\}$



MNIST Dataset ALOI Dataset Covertype Dataset Comments Dataset

FIGURE 4.1: Degree Distributions for Datasets considered

need to be searched. This takes time $O(|E|)$ where $|E|$ is the number of edges. Also, spectral partitioning is not guaranteed to return a balanced cut, which is what we need in order to ensure a logarithmic query time. In order to tackle these issues, for experiments we choose a splitting point u.a.r. in the interval $[X_{\ell/3}, X_{2\ell/3}]$, which denote the $(\ell/3, 2\ell/3)$ medians of $\{X_i\}$.

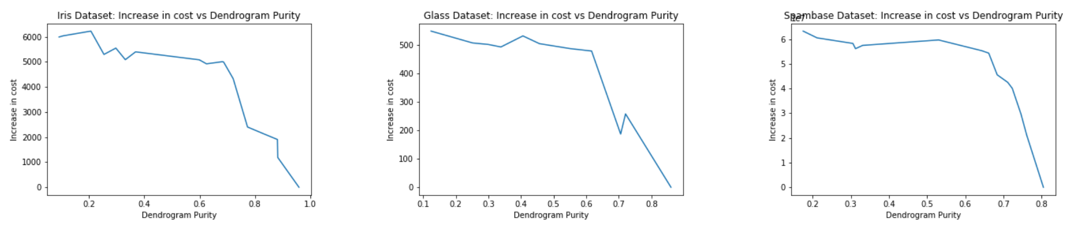
In Algorithm 4 and Algorithm 6 we use degree normalization for the similarity matrices, in order to obtain low-conductance cuts. As shown in Figure 4.1, our datasets have almost regular degree distribution, i.e., if the mean degree is d then $\Delta_{min} = (1-\alpha)d$ and $\Delta_{max} = (1+\alpha)d$. Since degree normalization has to be done at every internal node, for large datasets, namely `Covertype`, we do not perform degree normalization in the experiments to avoid the extra overhead.

4.0.3.1 Cost Function and Dendrogram Purity

In this task, we aim to find a relation between the cost function proposed by Dasgupta (2016) with dendrogram purity Kobren et al. (2017). We generated a tree with high purity using `AEV` and evaluate the corresponding cost of the tree. To obtain trees with lower purity, we randomly sample and shuffle a fixed fraction of the leaves of the tree and evaluate the cost and purity for the resultant tree. As the fraction of leaves to be shuffled increased, the purity decreased while the cost increased. The trend is shown in Figure 4.2 on the `Iris`(150 points, 3 attributes, 4 classes), `Glass`(214 points, 10 attributes, 7 classes) and `Spambase`(4601 points, 57 attributes, 2 classes) taken from Dheeru and Karra Taniskidou (2017).

4.0.3.2 Nearest Neighbor Classification using Hierarchies

In this task, we use our proposed hierarchical clustering method for the nearest neighbor classification task. We use the tree to find the approximate NN of the query point by traversing the tree to narrow down the candidate set size and doing a brute force search on a small number of points. The pseudocode for the querying and classification mechanism is given in Algorithm 3.



Iris Dataset Glass Dataset Spambase Dataset

Figure 4.2: Relation between Cost and Dendrogram Purty on Small Datasets

We report our results on the four datasets described above for two types of metrics. First, *classification performance*, is the standard classification analysis which includes macro-averaged Precision, Recall, and F1 Scores on the datasets. Second, *system performance*, compares system time taken for querying the models. The `Comments` classification task has specific business requirements that require definition of two binary sub-problems for the dataset. First subtask, `Clear` v/s (`Low-Quality` + `Spam`), determines whether a comment should receive unrestricted distribution on the site. Second subtask, (`Clear` + `Low-Quality`) v/s `Spam` identifies spam comments that should be removed from the site.

We compare performance of the following algorithms:

1. Our proposed methods, Hierarchical Tree Based NN (`HTNN`) with splitting rules: `EV` (using the exact eigenvector), `AEV` (using an approximate eigenvector), `RP` (using a random plane) and `2-means` (k-means with $k = 2$).

2. Linkage based trees (`LT`): Single, Average, Complete and Wards Linkages which minimize minimum, average, maximum and the merged variance between clusters respectively. There isn't any direct way to query in this structure as there is no distance information at the internal nodes that can be used. The tree is cut-off when there are as many clusters as the number of class labels. Each cluster is taken to be as one class. We take pairs of points from the `Test` set and then measure (using precision/recall/F1-Score) whether the algorithm accurately predicts the pairs of point to be from the same class.

3. SVM: The standard Support vector machine classifier with the best kernel among (`polynomial`, `rbf`, `sigmoid`).

4. Approximate Nearest Neighbor Structures (`ANN`): The standard k-d Tree Bentley (1975) and `LSH` Indyk and Motwani (1998) used for approximate nearest neighbor searches. For k-d-trees, we use the standard k-d-tree querying mechanism. For LSH, we use the multi-probe strategy with 2 probes per table. The chosen k which gave the best performance was 10. The `falconn` library (by Andoni et al. (2015)) function `LSHConstructionParameters` gives us an initial estimate for the `LSH` parameters.

Table 4.1: Performance in terms of Precision(P), Recall(R), F1-Score(F) and Query Time(QT) in milliseconds per query on MNIST and ALOI datasets (LSH is with $k = 3$ and $L = 20$)

		MNIST				ALOI			
	Methods	**P**	**R**	**F**	**QT**	**P**	**R**	**F**	**QT**
HTNN	**EV**	0.930	0.929	0.930	0.702	0.893	0.889	0.891	0.701
	AEV	0.924	0.924	0.924	0.729	0.893	0.890	0.891	0.67
	RP	0.818	0.815	0.816	0.789	0.776	0.768	0.772	0.55
	2-means	0.929	0.929	0.929	1.243	0.892	0.889	0.890	0.97
LT	**Average**	0.114	0.955	0.204	∼2k	0.036	0.670	0.069	∼2k
	Single	0.100	0.999	0.182	∼2k	0.001	0.971	0.002	∼2k
	Complete	0.180	0.393	0.247	∼2k	0.124	0.519	0.200	∼2k
	Ward	0.548	0.602	0.574	∼2k	0.393	0.562	0.463	∼2k
	SVM	0.93	0.93	0.93	-	0.843	0.830	0.836	-
ANN	**LSH**	0.954	0.953	0.953	0.85	0.936	0.92	0.934	0.68
	k-d Tree	0.969	0.968	0.968	540.38	0.949	0.948	0.948	22.30

Table 4.2: Performance in terms of Precision(P), Recall(R), F1-Score(F) and Query Time(QT) in milliseconds per query on Covertype and Comments datasets (LSH is with $k = 3$ and $L = 20$)

		Covertype				Comments					
						Clea vs LQ+Spam			Clear + LQ vs Spam		
	Method	**P**	**R**	**F**	**QT**	**P**	**R**	**F**	**P**	**R**	**F**
HTNN	**EV**	0.925	0.921	0.923	0.38	0.75	0.84	0.8	0.73	0.74	0.74
	AEV	0.925	0.922	0.923	0.37	0.76	0.84	0.8	0.73	0.74	0.74
	RP	0.904	0.897	0.901	0.37	-	-	-	-	-	-
	2-means	0.930	0.930	0.930	0.81	0.63	0.61	0.62	0.64	0.70	0.67
	SVM	0.211	0.308	0.250	-	-	-	-	-	-	-
ANN	**LSH**	0.581	0.739	0.650	44.72	0.71	0.74	0.73	0.62	0.63	0.62
	k-d Tree	0.942	0.935	0.938	0.76	-	-	-	-	-	-

4.0.3.3 Results

In this section, we summarize performance comparisons between algorithms described in Section 4.0.3.2 on the `ALOI` and `MNIST` datasets (Table 4.1) and on the `Covertype` and `Comments` datasets (Table 4.2). We use 20 tables for `LSH` as $\log(n)$ for all datasets lie in $[16, 20]$ (n is the size of `Train` set).

Classification and Performance Analysis : Here, we compare performance of the above algorithms in terms of their classification/run-time performances.

We report performance numbers on a `Test` set averaged over five runs for each dataset. We also consider **10** nearest neighbours for the `HTNN` and `ANN` algorithms. From Table 4.1 we see that precision, recall and F1-score of `HTNN` is comparable to `ANN` and `HTNN` performs better than `LT` on all tasks. Also, `HTNN` performs similar to `SVM` on `MNIST` and `ALOI` but for `Covertype` dataset the precision/recall and F1 score for `SVM` is poor as the `SVM` algorithm did not converge for the large dataset.

LSH, though has comparable quality and querying time, it does not maintain any hierarchy, and thus will be unsuitable for tasks where the hierarchical structure may be beneficial. It also does not perform well for the real-world dataset. The k-d tree performs poorly in terms of query time (Table 4.1) and its query time increases with data dimensionality, which makes it unfeasible for high dimensional problems. There are two possible reasons– we use the standard kd-tree nearest neighbor querying mechanism, which can take time almost linear since it searches the cells adjacent to the one the query point falls in. Furthermore, our larger datasets are sparse, and partitioning the data on a coordinate's median value is not useful if more than 50% of the values are zero. HTNN is 741, 33, and 1.88 times faster than kd-tree on the open source datasets MNIST, ALOI, and Covertype respectively. As the number of data points increases, memory usage of LT based data structures becomes unfeasible as observed on the Covertype dataset. Query time for SVM and LT are also too large to feature in Table 4.2. We also perform the test of significance for the ALOI and the MNIST datasets on 5 samples obtained from different simulations of the algorithms.

Clear v/s (Low-Quality+ Spam) and *(Clear+ Low-Quality) v/s Spam Tasks:* These tasks are binary sub-problems on Comments dataset, which are important for the site. As mentioned above, the (Clear+ Low-Quality) v/s Spam task is harder because there is significant overlap between Low-Quality and Spam categories. For Clear v/s (Low-Quality+ Spam) task, HTNN performs approximately 9% better than LSH (Table 4.2). For the (Clear+ Low-Quality) v/s Spam task the F1-Score for both LSH and HTNN reduces, but HTNN still outperforms LSH. Note that LSH is not hierarchical tree based data structure which is a need for the application. Tree variants of LSH (e.g., *LSH Forest*) exist, but it is unclear how to merge and interpret multiple trees of the forest for the application. For Comments dataset we do not show the performances of LT, SVM , and k-d tree based data structures because of their time and space consumption which makes them infeasible for the application.

We run all algorithms on an Intel(R) Xeon(R) CPU E5-1620 with 64 GB RAM. All algorithms proposed in this paper were coded in Python3.6 using Numpy 1.15.1. SVM, k-d Tree and LT algorithms were used from python's scikit-learn library version 0.19.1. LSH from the falconn library was used and LSHConstructionParameters was used to give an initial estimate of the required parameters.

4.0.3.4 Test of Significance

We perform the test of significance using scipy library in-built function ttest_ind. This was a *Student's t-test*. We run the AEV, SVM, kd-tree and LSH algorithms five times and compare the results for them in Table 4.3.

Table 4.3: Test of Significance with AEV Splitting Rule

	SVM	**k-d Tree**	**LSH**	**p-value**
ALOI	Reject	Reject	Reject	0.05
MNIST	Fail to reject	Reject	Reject	0.05

Table 4.4: Performance in terms of Precision(P), Recall(R), F1-Score(F) for the Anomaly Detection Task

	AEV				**SVM**			
	Threshold	**P**	**R**	**F**	**Threshold**	**P**	**R**	**F**
Data1	0.2	0.53	0.78	0.63	0.05	0.42	0.82	0.55
	0.3	0.58	0.66	0.61	0.08	0.42	0.76	0.54
	0.4	0.62	0.53	0.57	0.1	0.42	0.56	0.48
Data2	0.2	0.53	0.80	0.64	0.05	0.42	0.84	0.56
	0.3	0.58	0.66	0.62	0.08	0.44	0.80	0.56
	0.4	0.63	0.54	0.58	0.1	0.45	0.61	0.52
Data3	0.2	0.52	0.78	0.62	0.05	0.42	0.86	0.57
	0.3	0.57	0.67	0.62	0.08	0.43	0.78	0.55
	0.4	0.61	0.53	0.57	0.1	0.43	0.60	0.50

Let H_0 be the hypothesis that the F1-Score for the two algorithms come from the same distribution.

For `MNIST`, we fail to reject the null hypothesis for `AEV` vs `SVM` as the values obtained are very close to each other. For all other datasets, we are able to reject the null hypothesis with a p-value of 0.05.

4.0.3.5 Anomaly Detection using Hierarchies

In this task, we use our hierarchical clustering method for anomaly detection. Since `ALOI` has 1k classes, we separate out 50 classes from the dataset. We also sample points from other classes. We create 3 such test sets, each with about 5k points from the unseen classes and 10k points from seen classes. The train set consists of $\sim$ 93k points.

We follow a modified version of the classification mechanism used in the nearest neighbor classification task. The two main modifications are i) we calculate the average pairwise distance of the S points returned from query (say d_1), find the k-NN from the query and compare the average k-NN distances (say d_2 to d_1, and ii) if $d_2 > d_1$ by a significant amount, then we mark a node T at a height of $\frac{3}{4}h$ where h is the depth of the tree. For a new point, if T is marked and $d_2 > d_1$ by a substantial amount (threshold), then mark that query as a possible anomaly. For the baseline, we use `SVM`'s prediction probabilities. If a point is from a seen class, it should have a high prediction probability (threshold) for that class, and a new point should not have a high prediction probability for any class. This intuition was used for the experiments. From Table 4.4 we observe that for similar recall, we get a higher precision. This indicates that we are catching the new points with higher accuracy (i.e., we label non-anomaly points to be an anomaly lesser than the `SVM` model). We also observe that the `AEV` algorithm achieves a better F1-Score than `SVM`.

4.0.4 Conclusion

We present a hierarchical clustering algorithm with three splitting strategies. Each of these have a guaranteed theoretical bound in terms of the cost function as given in Equation 2.1. All proposed splitting strategies have time complexity $O(nd^2)$ if n is the number of points and d is the dimensions. We also demonstrate the performance of this algorithm on three open source dataset and one real dataset for two applications, namely, near-neighbor classification and anomaly detection.

Though this model satisfies the requirements of being unsupervised and hierarchical with low query time, there is no support for incremental updates. We target this requirement in the next chapter.

Online Hierarchical Clustering

5.0.1 Introduction

One of the major requirements, as mentioned in Section 1, was to allow incremental updates to the model. This allows the model to take into account the new content types that were encountered. Thus, we focus on the online setting, where data does not arrive as a batch of points but as a stream. Hierarchical clustering tree updates must be carried out on the fly without access to yet-to-arrive data. Our goal is to maintain a cost efficient hierarchical data structure. Though Kobren et al. (2017) proposed an online hierarchical clustering algorithm recently, it does not have any theoretical bounds in terms of any specific objective.

We present two heuristics for the online hierarchical clustering problem, empirically demonstrate that the quality degradation as compared to the batch hierarchical clustering algorithms is very little.

5.0.2 Heuristics

We propose a heuristic extension to the `HTNN` data structure. Intuition behind the heuristic is as follows. When a new data point arrives, it is traversed down the tree to the most relevant leaf and is added as a sibling of that leaf. If at any internal node u along the path, a rebuild condition (described below) is met, the subtree rooted at u is rebuilt as in the batch version of Algorithm 3. The proposed algorithm can be thought of as a combination of *Batch and Online* versions. The pseudo-code of the algorithm is outlined in Algorithm 8. We propose two heuristics for rebuilding the tree at node u. Those are described as follows:

Algorithm 8 Online Hierarchical Clustering

```
1:  procedure InsertPoint($(r, x)$)
2:      if $isNULL(r)$ then
3:          return x
4:      if $isLeaf(r)$ then
5:          $h_r, s_r \leftarrow$ SplittingRule($[leaves[T(r)] \cup x]$)
6:          $I$ is new internal node with $r, x$ as leaves and $h_r, s_r$ as splitting hyperplane, location
7:          replace $r$ with $I$ in parent of $r$
8:          return $I$
9:      if Rebuild Condition is satisfied then
10:         return MakeTree([leaves[T(r) ∪ x])
11:     $h_r, s_r$ be the splitting hyperplane, location for $r$.
12:     if $x \cdot h_r < s_r$ then  Insert $x$ to left child of $r$.
13:     else  Insert $x$ to right child of $r$.
14:     return $r$
```

Doubling Heuristic: If $|leaves(T[u])| = 2 * ptsAdded(u)$, where $ptsAdded(u)$ is the number of new points added since the last rebuild operation, then the subtree rooted at u is rebuilt.

Balancing Heuristic: If $|leaves(T[u.left])|$ is greater than twice of $|leaves(T[u.right])|$, where $u.left$, $u.right$ are the left and right children of u, respectively, then the subtree rooted at u is rebuilt. This guarantees a $(1/3, 2/3)$ balancing condition in the tree.

5.0.3 Results

We report results using `AEV` as SplittingRule for all experiments. Table 5.1 summarizes classification performance on open source datasets. For both the rebuild heuristics described above, `AEV` achieves at least 80% of the offline F1 score on all the open source datasets (also see Table 4.1).

For the `Comments` dataset, we compare `AEV` splitting rule with `Doubling Heuristic` with the `PERCH` algorithm of Kobren et al. (2017). For the `Clear` v/s (`Low-Quality`+ `Spam`) task, F1-score for both `AEV` and `PERCH` are comparable. However, for the (`Clear`+ `Low-Quality`) v/s `Spam` task `AEV` performs around 41% percent better than `PERCH`. We suspect this is because the `AEV` splitting rule is more suited for situations with high class overlap along a few critical dimensions, as can be expected of overlapping text categories, compared to `PERCH`. The `AEV` splitting rule also performs best for batch mode `Comments` task (see Table 4.1).

Dataset	Doubling Heuristic P	R	F1	Balancing Heuristic P	R	F1
MNIST	0.917	0.916	0.916	0.893	0.890	0.891
ALOI	0.826	0.807	0.816	0.807	0.795	0.801
CoverType	0.880	0.877	0.878	0.882	0.876	0.879

TABLE 5.1: Precision(P), Recall(R) and F1-Score(F1) for the Online heuristics using AEV

Methods	Clear vs LQ + Spam P	R	F1	Clear + LQ vs Spam P	R	F1
AEV	0.753	0.853	0.80	0.685	0.686	0.685
PERCH	0.708	0.971	0.82	0.48	0.454	0.466

TABLE 5.2: Precision(P), Recall(R) and F1-Score(F1) on Comments Dataset for the Doubling Heurestic

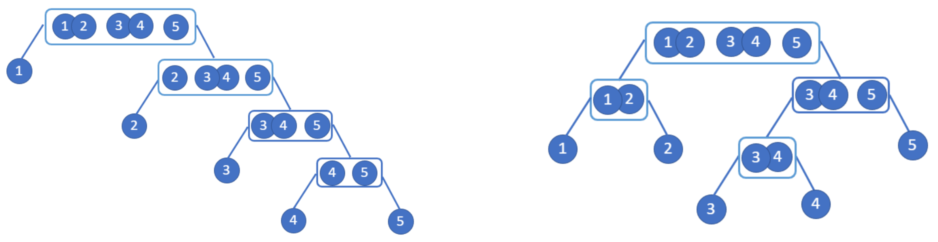

(a) Hierarchical Clustering with Additions (b) Desired Hierarchical Clustering

FIGURE 5.1: Online Hierarchical Clustering

5.0.4 Eigenvector Updates for Hierarchical Clustering

The proposed heuristics do not have any approximation guarantee for the hierarchical clustering produced. This is a major drawback as there is no bound on how bad the clustering can get. Since we use eigenvectors for a set of points (in a non-incremental manner) as a SPLITTINGRULE, the next logical step is to update the eigenvectors on the fly. Though there is some work done on streaming or online updates of eigenvectors, this is independent of the online hierarchical clustering. Also, most of the algorithms proposed for online eigenvector updates deal with only addition of data. We first demonstrate our inherent requirement of support for insertions and deletions.

In Figure 5.1, the root node shows points in the one dimensional space. The distance between the points as shown in the root are an indicator of the Euclidean distance between them. Suppose the points arrive in order as *1,2,3,4 and 5*. Figure 5.1(a) shows the clustering formed without any rebuilding. This hierarchical clustering has the following properties:

- The tree formed can be unbalanced, thus query time could be as bad as $O(n)$ where n is the number of points

- Points that are close in terms of distance can be far or separated in the tree.

These properties are undesirable as (i) we require a near real time querying, and (ii) separating point close in terms of distance incurs a high cost. Figure 5.1(b) shows a tree which does not have the aforementioned undesirable qualities. In this case, to go from Figure 5.1(a) to Figure 5.1(b), the following changes would be required:

1. Delete node *4* from the node containing *4 and 5*.
2. Insert node *4* to the node containing *3*.
3. Delete node *2* from the node containing *2,3,4 and 5*.
4. Insert node *2* to the node containing *1*.

Thus, it is clear from the above example that we need deletions of points as well for maintaining a hierarchical clustering tree with the desirable properties. The deletion performed here is not a deletion from the dataset but deleting from one subtree and inserting to another subtree. However, the insert operations can be performed by updating the eigenvectors using the algorithm proposed by Mitliagkas et al. (2013). But, there is no work which focuses on focuses on eigenvector updations for dynamic streams which have insertion and deletion operations. Therefore, identifying the online eigenvector updates for a dynamic stream as a subproblem, we propose an algorithm for the same in the next chapter.

5.0.5 Conclusion

We present two heuristics for the online hierarchical clustering which show little quality degradation as compared to their batch counterparts. The heurisitics proposed in this chapter satisfy all requirements of being unsupervised, hierarchical and incrementally modifiable with near real-time classification. The drawback is that we do not have any theoretical bounds on the quality of the solutions. To obtain a theoretical bound, we identify the sub-problem of online eigenvector updates. This could potentially be used to obtain a bound on the cost of the online hierarchical clustering algorithm. In the next chapter, we focus on the online eigenvector update problem for dynamic streams.

Online Eigenvector Updates

6.0.1 Introduction

A natural sub-problem that came to notice while tackling the online hierarchical clustering problem was that of *Online Eigenvector Estimation*. Though algorithms exist to maintain eigenvectors in setting where data can only be added such as the one by Mitliagkas et al. (2013), our model inherently needed support for deletions as well as explained in Section 5.0.4. No algorithm existed to update the eigenvector on insertions as well as deletions (dynamic stream).

The problem of online eigenvector estimation in streams where points may be added or deleted has various use-cases. One use case is in social networks where users may join and leave, or comments, images, videos or other media may be added or deleted. Another use case is in sensors where erroneous data may require deletion. In such cases, it is of utmost importance to delete the data to get better accuracy. This directly translates to deletion of a data point in the stream.

We first outline some naive heuristics for online hierarchical clustering and empirically demonstrate that there is very little degradation in the quality of the solution. We propose an algorithm for the online eigenvector updation problem for the *orthogonal dual spiked covariance model* as described in Section 6.0.2, give some theoretical guarantees and demonstrate the empirical performance of the proposed algorithm.

6.0.2 Problem Formulation and Notation

We consider a dynamic streaming model, where at each time step t, we receive a point (x_t, op), where $x_t \in \mathbb{R}^p$ and op is an operation. This operation can either be an **insert** operation or a **delete** operation. Furthermore, any vector that is not explicitly stored can never be revisited. Our aim is to compute the top eigenvector of the data.

We assume a probabilistic generative model for which data is sampled from at each step t. The model is comprised of two orthogonal spikes. Specifically, we assume that the data to be inserted is sampled at each time step as:

$$\mathbf{x_t} = \mathbf{u}z_t + \mathbf{w_t} \tag{6.1}$$

or

$$\mathbf{x_t} = \mathbf{v}z_t + \mathbf{y_t} \tag{6.2}$$

where $u \in \mathbb{R}^{p\times 1}$ is a fixed vector (i.e., the eigenvector), $z_t \in \mathbb{R}^{k\times 1}$ is a multivariate normal random variable, i.e.,

$$z_t \sim \mathcal{N}(0, 1)$$

vector $\mathbf{w_t} \in \mathbb{R}^{p\times 1}$ is the noise vector for the model 6.1 and is also sampled from a multivariate normal distribution, i.e.,

$$\mathbf{w_t} \sim \mathcal{N}(0_{p\times 1}, \sigma_1^2 I_{p\times p})$$

vector $\mathbf{y_t} \in \mathbb{R}^{p\times 1}$ is the noise vector for the model 6.2 and is also sampled from a multivariate normal distribution, i.e.,

$$\mathbf{y_t} \sim \mathcal{N}(0_{p\times 1}, \sigma_2^2 I_{p\times p})$$

Furthermore $\mathbf{u} \perp \mathbf{v}$, i.e, $\mathbf{u}^T\mathbf{v} = \mathbf{v}^T\mathbf{u} = 0$, and all $z_t, \mathbf{w_t}, \mathbf{y_t}$ are mutually independent.

The points to be deleted only come from the points inserted from the model given in Equation 6.2. We also assume that the points and operations come in batches, that is, a batch can either be data from Model 6.1 which is to be inserted, or data can be from Model 6.2 which has to either to be inserted or deleted.

The aim of this work is to provide guarantees for a dynamic streaming algorithm that requires $O(p)$ memory.

We shall denote matrices by capital letters (e.g. A), vectors by lower case bold letters (e.g. $\mathbf{x}$). $||\mathbf{x}||_q$ denotes the $q-$norm of $\mathbf{x}$, $||A||$ or $||A||_2$ denotes the spectral norm of A and $||A||_F$ denotes the Frobenius norm of A. $< \mathbf{x}, \mathbf{y} >$ denotes the dot product of vectors $\mathbf{x}$ and $\mathbf{y}$. In proofs the constant C is used loosely and may vary at every statement.

Algorithm 9 Largest Eigenvector updates on Insertion

```
1: procedure INSERT($B, \mathbf{q}, \lambda$)
2:     $\mathbf{s} \leftarrow 0$
3:     for $\mathbf{x_t} \in B$ do
4:         $\mathbf{y} \leftarrow < \mathbf{q}, \mathbf{x_t} > \mathbf{x_t}$
5:         $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{y}$
6:         $\lambda \leftarrow \lambda + < \mathbf{q}, \mathbf{y} >$
7:     $\mathbf{s} \leftarrow \frac{1}{B}\mathbf{s}$
8:     $\mathbf{q} \leftarrow \frac{\mathbf{s}}{||\mathbf{s}||_2}$
9:     return $\mathbf{q}, \lambda$
```

Algorithm 10 Largest Eigenvector updates on Deletion

```
1: procedure DELETE($B, \mathbf{q}, \lambda$)
2:     $\mathbf{s} \leftarrow 0$
3:     $\lambda_t \leftarrow 0$
4:     for $\mathbf{x_t} \in B$ do
5:         $\mathbf{y} \leftarrow < \mathbf{q}, \mathbf{x_t} > \mathbf{x_t}$
6:         $\mathbf{s} \leftarrow \mathbf{s} + \mathbf{y}$
7:         $\lambda_t \leftarrow \lambda_t + < \mathbf{q}, \mathbf{y} >$
8:     $\mathbf{s} \leftarrow \lambda\mathbf{q} - \mathbf{s}$
9:     $\mathbf{q} \leftarrow \frac{\mathbf{s}}{||\mathbf{s}||_2}$
10:    $\lambda \leftarrow \lambda - \lambda_t$
11:    return $\mathbf{q}, \lambda$
```

6.0.3 Proposed Algorithm

6.0.4 Theoretical Guarantees

We first quote some results from Mitliagkas et al. (2013).

Lemma 6.1. *For a batch of insertions where the points come from Model 6.1, with probability $1 - C/T$, for C a universal constant, we have*

$$\left|\left|\frac{1}{B}\sum \mathbf{x_t}\mathbf{x_t}^T - uu^t - \sigma_1^2 I\right|\right|_2 \leq \epsilon$$

Lemma 6.2. *For a batch of points which come from Model 6.2 With probability $1 - C/T$, for C a universal constant, we have:*

$$\left|\left|\frac{1}{B}\sum \mathbf{x_t}\mathbf{x_t}^T - vv^t - \sigma_2^2 I\right|\right|_2 \leq \epsilon$$

Lemma 6.1 and Lemma 6.2 show that for large enough batch size B, the empirical covariance matrices are close to the actual covariance matrix. That is, for a batch of points from Model 6.1, $\frac{1}{B}\sum_t x_t x_t^T$ is close to $uu^T + \sigma_1^2 I$ and for a batch of points from Model 6.2, $\frac{1}{B}\sum_t x_t x_t^T$ is close to $vv^T + \sigma_2^2 I$

For the rest of the chapter, let us assume that $\mathbf{q}_\tau = \sqrt{\alpha_\tau}\mathbf{u} + \sqrt{\gamma_\tau}\mathbf{v} + \sqrt{\delta_\tau}\mathbf{g}_\tau$ where $\mathbf{g}_\tau$ is orthogonal to both $\mathbf{u}$, $\mathbf{v}$, and $\alpha_\tau + \gamma_\tau + \delta_\tau = 1$.

It is also worthwhile to note that

$$\alpha_\tau = (\mathbf{u^T q}_\tau)^2 = \frac{(\mathbf{u^T s}_\tau)^2}{||\mathbf{s}_\tau||_2^2} \tag{6.3}$$

$$\gamma_\tau = (\mathbf{v^T q}_\tau)^2 = \frac{(\mathbf{v^T s}_\tau)^2}{||\mathbf{s}_\tau||_2^2} \tag{6.4}$$

$$\delta_\tau = (\mathbf{g}_\tau^\mathbf{T} \mathbf{q}_\tau)^2 = \frac{(\mathbf{g}_\tau^\mathbf{T} \mathbf{s}_\tau)^2}{||\mathbf{s}_\tau||_2^2} \tag{6.5}$$

6.0.4.1 Insertions

We now characterize the behaviour of our algorithm on insertion of a batch that takes place at the $\tau + 1$-th iteration.

Lemma 6.3. *On insertion of a batch of points sampled from Model 6.1, the components along* $\mathbf{u}$, $\mathbf{v}$ *and* $\mathbf{g}_\tau$ *change as:*

$$\alpha_{\tau+1} = \frac{\alpha_\tau(1+\sigma_1^2+\epsilon)^2}{(1+\sigma_1^2+\epsilon)^2 - (1-\alpha_\tau)(1+2(\sigma_1^2+\epsilon))} \tag{6.6}$$

$$\gamma_{\tau+1} = \frac{\gamma_\tau(\sigma_1^2+\epsilon)^2}{(\sigma_1^2+\epsilon)^2 + \alpha_\tau(1+2(\sigma_1^2+\epsilon))} \tag{6.7}$$

$$\delta_{\tau+1} = \frac{\delta_\tau(\sigma_1^2+\epsilon)^2}{(\sigma_1^2+\epsilon)^2 + \alpha_\tau(1+2(\sigma_1^2+\epsilon))} \tag{6.8}$$

Proof. Let us assume that B is the $\tau + 1 - th$ batch, and all points in this batch are from the Model 6.1.

$$\begin{aligned}
\frac{1}{B}\sum_{t\in B} \mathbf{x_t x_t}^T \mathbf{q}_\tau &= (\mathbf{uu}^T + \sigma_1^2 I + E)\mathbf{q}_\tau \\
&= \sqrt{\alpha_\tau}\mathbf{u} + (\sigma_1^2+\epsilon)\mathbf{q}_\tau \\
\mathbf{s_{\tau+1}} &= \sqrt{\alpha_\tau}\mathbf{u} + (\sigma_1^2+\epsilon)\mathbf{q}_\tau \\
&= \sqrt{\alpha_\tau}\mathbf{u}(1+(\sigma_1^2+\epsilon)) + (\sqrt{\gamma_\tau}\mathbf{v} + \sqrt{\delta_\tau}\mathbf{g}_\tau)(\sigma_1^2+\epsilon) \\
||\mathbf{s_{\tau+1}}||^2 &= \alpha_\tau(1+\sigma_1^2+\epsilon)^2 + \gamma_\tau(\sigma_1^2+\epsilon)^2 + \delta_\tau(\sigma_1^2+\epsilon)^2
\end{aligned}$$

$||\mathbf{s}_{\tau+\mathbf{1}}||^2$ can be written in two ways.

$$||\mathbf{s}_{\tau+\mathbf{1}}||_2^2 = (\sigma_1^2 + \epsilon)^2 + \alpha_\tau(1 + 2\sigma_1^2 + 2\epsilon)$$

and

$$||\mathbf{s}_{\tau+\mathbf{1}}||_2^2 = (1 + \sigma_1^2 + \epsilon)^2 - (1 - \alpha_\tau)(1 + 2\sigma_1^2 + 2\epsilon)$$

Using Equation 6.3, Equation 6.4 and Equation 6.5,

$$\begin{aligned}
\alpha_{\tau+1} &= \frac{\alpha_\tau(1 + \sigma_1^2 + \epsilon)^2}{(1 + \sigma_1^2 + \epsilon)^2 - (1 - \alpha_\tau)(1 + 2\sigma_1^2 + 2\epsilon)} \\
\gamma_{\tau+1} &= \frac{\gamma_\tau(\sigma_1^2 + \epsilon)^2}{(\sigma_1^2 + \epsilon)^2 + \alpha_\tau(1 + 2\sigma_1^2 + 2\epsilon)} \\
\delta_{\tau+1} &= \frac{\delta_\tau(\sigma_1^2 + \epsilon)^2}{(\sigma_1^2 + \epsilon)^2 + \alpha_\tau(1 + 2\sigma_1^2 + 2\epsilon)}
\end{aligned}$$

□

Upon insertion of a batch of points from Model 6.1, we observe that the component along **u** increases, while the components along **v** and $\mathbf{g}_\tau$ decreases. As the number of batches from Model 6.1 increases, the eigenvector estimated should align more with **u** and, hence, the component α_τ should increase, and γ_τ, δ_τ should decrease.

Lemma 6.4. *On insertion of a batch of points sampled from Model 6.2, the components along* **u**, **v** *and* $\mathbf{g}_\tau$ *change as:*

$$\alpha_{\tau+1} = \frac{\alpha_\tau(\sigma_2^2 + \epsilon)^2}{(\sigma_2^2 + \epsilon)^2 + \gamma_\tau(1 + 2(\sigma_2^2 + \epsilon))} \tag{6.9}$$

$$\gamma_{\tau+1} = \frac{\gamma_\tau(1 + \sigma_2^2 + \epsilon)^2}{(1 + \sigma_2^2 + \epsilon)^2 - (1 - \gamma_\tau)(1 + 2(\sigma_2^2 + \epsilon))} \tag{6.10}$$

$$\delta_{\tau+1} = \frac{\delta_\tau(\sigma_2^2 + \epsilon)^2}{(\sigma_2^2 + \epsilon)^2 + \gamma_\tau(1 + 2(\sigma_2^2 + \epsilon))} \tag{6.11}$$

Proof. Let us assume that B is the $\tau + 1 - th$ batch, and all points in this batch are from the Model 6.2.

$$\begin{aligned}
\frac{1}{B}\sum_{t \in B} \mathbf{x_t}\mathbf{x_t}^T \mathbf{q}_\tau &= (\mathbf{v}\mathbf{v}^T + \sigma_2^2 I + E)\mathbf{q}_\tau \\
&= \sqrt{\gamma_\tau}\mathbf{v} + (\sigma_2^2 + \epsilon)\mathbf{q}_\tau \\
\mathbf{s_{\tau+1}} &= \sqrt{\gamma_\tau}\mathbf{v} + (\sigma_2^2 + \epsilon)\mathbf{q}_\tau \\
&= \sqrt{\gamma_\tau}\mathbf{v}(1 + (\sigma_2^2 + \epsilon)) + (\sqrt{\alpha_\tau}\mathbf{u} + \sqrt{\delta_\tau}\mathbf{g}_\tau)(\sigma_2^2 + \epsilon) \\
||\mathbf{s_{\tau+1}}||^2 &= \gamma_\tau(1 + \sigma_2^2 + \epsilon)^2 + \alpha_\tau(\sigma_2^2 + \epsilon)^2 + \delta_\tau(\sigma_2^2 + \epsilon)^2
\end{aligned}$$

$||\mathbf{s_{\tau+1}}||^2$ can be written in two ways.

$$||\mathbf{s_{\tau+1}}||_2^2 = (\sigma_2^2 + \epsilon)^2 + \gamma_\tau(1 + 2\sigma_2^2 + 2\epsilon)$$

and

$$||\mathbf{s_{\tau+1}}||_2^2 = (1 + \sigma_2^2 + \epsilon)^2 - (1 - \gamma_\tau)(1 + 2\sigma_2^2 + 2\epsilon)$$

Using Equation 6.3, Equation 6.4 and Equation 6.5,

$$\begin{aligned}
\alpha_{\tau+1} &= \frac{\alpha_\tau(\sigma_2^2 + \epsilon)^2}{(\sigma_2^2 + \epsilon)^2 + \gamma_\tau(1 + 2(\sigma_2^2 + \epsilon))} \\
\gamma_{\tau+1} &= \frac{\gamma_\tau(1 + \sigma_2^2 + \epsilon)^2}{(1 + \sigma_2^2 + \epsilon)^2 - (1 - \gamma_\tau)(1 + 2(\sigma_2^2 + \epsilon))} \\
\delta_{\tau+1} &= \frac{\delta_\tau(\sigma_2^2 + \epsilon)^2}{(\sigma_2^2 + \epsilon)^2 + \gamma_\tau(1 + 2(\sigma_2^2 + \epsilon))}
\end{aligned}$$

□

Upon insertion of a batch of points from Model 6.2, we observe that the component along $\mathbf{v}$ increases, while the components along $\mathbf{u}$ and $\mathbf{g}_\tau$ decreases. As the number of batches from Model 6.2 increases, the eigenvector estimated should align more with $\mathbf{v}$ and, hence, the component γ_τ should increase, and α_τ, δ_τ should decrease.

6.0.4.2 Deletions

We characterize the behaviour of the proposed algorithm on deletion of a batch of points which happens at the $\tau + 1$-th iteration.

Lemma 6.5. *On deletion of a batch of points sampled from Model 6.2, the components along* $\mathbf{u}$*,* $\mathbf{v}$ *and* $\mathbf{g}_\tau$ *change as:*

$$\alpha_{\tau+1} = \frac{\alpha_\tau(\lambda_\tau - B(\sigma_2^2+\epsilon))^2}{(\lambda_\tau - B(\sigma_2^2+\epsilon))^2 - \gamma_\tau(2B\lambda_\tau - B^2(1+2\sigma_2^2+2\epsilon))} \tag{6.12}$$

$$\gamma_{\tau+1} = \frac{\gamma_\tau(\lambda_\tau - B(1+\sigma_2^2+\epsilon))^2}{(\lambda_\tau - B(1+\sigma_2^2+\epsilon))^2 + (1-\gamma_\tau)(B^2(1+2\sigma_2^2+2\epsilon) - 2B\lambda_\tau)} \tag{6.13}$$

$$\delta_{\tau+1} = \frac{\delta_\tau(\lambda_\tau - B(\sigma_2^2+\epsilon))^2}{(\lambda_\tau - B(\sigma_2^2+\epsilon))^2 - \gamma_\tau(2B\lambda_\tau - B^2(1+2\sigma_2^2+2\epsilon))} \tag{6.14}$$

Proof. If B consists of the points in the $\tau+1-th$ batch, and all points to be deleted are from Model 6.2,

$$\begin{aligned}\mathbf{y} &= B(\mathbf{v}\mathbf{v^T} + \sigma_2^2 I + E)\mathbf{q}_\tau \\ &= B(\sqrt{\gamma_\tau}\mathbf{v} + (\sigma_2^2+\epsilon)\mathbf{q}_\tau)\end{aligned}$$

Now, we can write,

$$\begin{aligned}\mathbf{s_{\tau+1}} &= \lambda_\tau \mathbf{q}_\tau - \mathbf{y} \\ &= (\lambda_\tau - B(\mathbf{v}\mathbf{v^T} + \sigma_2^2 I + E))\mathbf{q}_\tau \\ &= (\sqrt{\alpha_\tau}\mathbf{u} + \sqrt{\delta_\tau}\mathbf{g}_\tau)(\lambda_\tau - B(\sigma_2^2+\epsilon)) + \sqrt{\gamma_\tau}\mathbf{v}(\lambda_\tau - B(1+\sigma_2^2+\epsilon)) \\ ||\mathbf{s_{\tau+1}}||_2^2 &= (\alpha_\tau + \delta_\tau)(\lambda_\tau - B(\sigma_2^2+\epsilon))^2 + \gamma_\tau(\lambda_\tau - B(1+\sigma_2^2+\epsilon))^2\end{aligned}$$

Again, $||\mathbf{s_{\tau+1}}||_2^2$ can be written in two ways,

$$||\mathbf{s_{\tau+1}}||_2^2 = (\lambda_\tau - B(\sigma_2^2+\epsilon))^2 + \gamma_\tau(B^2(1+2\sigma_2^2+2\epsilon) - 2B\lambda_\tau)$$

and

$$||\mathbf{s_{\tau+1}}||_2^2 = (\lambda_\tau - B(1+\sigma_2^2+\epsilon))^2 + (1-\gamma_\tau)(B^2(1+2\sigma_2^2+2\epsilon) - 2B\lambda_\tau)$$

Using Equation 6.3, Equation 6.4 and Equation 6.5,

$$\begin{aligned}\alpha_{\tau+1} &= \frac{\alpha_\tau(\lambda_\tau - B(\sigma_2^2+\epsilon))^2}{(\lambda_\tau - B(\sigma_2^2+\epsilon))^2 - \gamma_\tau(2B\lambda_\tau - B^2(1+2\sigma_2^2+2\epsilon))} \\ \gamma_{\tau+1} &= \frac{\gamma_\tau(\lambda_\tau - B(1+\sigma_2^2+\epsilon))^2}{(\lambda_\tau - B(1+\sigma_2^2+\epsilon))^2 + (1-\gamma_\tau)(B^2(1+2\sigma_2^2+2\epsilon) - 2B\lambda_\tau)} \\ \delta_{\tau+1} &= \frac{\delta_\tau(\lambda_\tau - B(\sigma_2^2+\epsilon))^2}{(\lambda_\tau - B(\sigma_2^2+\epsilon))^2 - \gamma_\tau(2B\lambda_\tau - B^2(1+2\sigma_2^2+2\epsilon))}\end{aligned}$$

□

Upon deletion of a batch of points from Model 6.2, we observe that the component along **v** decreases, while the components along **v** and $\mathbf{g}_\tau$ increases. But, this trend is only observed when $2B\lambda_\tau > B^2(1+2\sigma_2^2+2\epsilon)$. λ_τ is an indication of the number of points in the dataset and is an indicator of the number of points in the data. As the number of batches from Model 6.1 increases, the eigenvector estimated should align more with **u** and, hence, the component α_τ should increase. Since we are deleting a component along **v**, we expect γ_τ to decrease and the components perpendicular to **v**, i.e., α_τ and δ_τ to increase.

Theorem 6.6. *For any batch of insertions or deletions,*

$$\frac{\alpha_{\tau+1}+\delta_{\tau+1}}{\alpha_\tau+\delta_\tau} \leq \frac{\alpha_{\tau+1}}{\alpha_\tau}$$

Proof. There can be atmost three different types of operations, i.e.,

Case 1: Insertion of a batch where data is from Model 6.1

Note: Equation 6.6 can also be written as

$$\alpha_{\tau+1} = \frac{\alpha_\tau(1+\sigma_1^2+\epsilon)^2}{(\sigma_1^2+\epsilon)^2+\alpha_\tau(1+2(\sigma_1^2+\epsilon))} \tag{6.15}$$

From Equation 6.15 and Equation 6.8,

$$\begin{aligned}
\frac{\delta_{\tau+1}}{\alpha_{\tau+1}} &= \frac{\frac{\delta_\tau(\sigma_1^2+\epsilon)^2}{(\sigma_1^2+\epsilon)^2+\alpha_\tau(1+2(\sigma_1^2+\epsilon))}}{\frac{\alpha_\tau(1+\sigma_1^2+\epsilon)^2}{(\sigma_1^2+\epsilon)^2+\alpha_\tau(1+2(\sigma_1^2+\epsilon))}} \\
&= \frac{\delta_\tau(\sigma_1^2+\epsilon)^2}{\alpha_\tau(1+\sigma_1^2+\epsilon)^2} \\
&\leq \frac{\delta_\tau}{\alpha_\tau} \\
\frac{\delta_{\tau+1}+\alpha_{\tau+1}}{\alpha_{\tau+1}} &\leq \frac{\delta_\tau+\alpha_\tau}{\alpha_\tau}
\end{aligned}$$

Case 2: Insertion of a batch where data is from Model 6.2

From Equation 6.9 and Equation 6.11,

$$\begin{aligned}
\frac{\delta_{\tau+1}}{\alpha_{\tau+1}} &= \frac{\frac{\delta_\tau(\sigma_2^2+\epsilon)^2}{(\sigma_2^2+\epsilon)^2+\gamma_\tau(1+2(\sigma_2^2+\epsilon))}}{\frac{\alpha_\tau(\sigma_2^2+\epsilon)^2}{(\sigma_2^2+\epsilon)^2+\gamma_\tau(1+2(\sigma_2^2+\epsilon))}} \\
\frac{\delta_{\tau+1}}{\alpha_{\tau+1}} &= \frac{\delta_\tau}{\alpha_\tau} \\
\frac{\delta_{\tau+1}+\alpha_{\tau+1}}{\alpha_{\tau+1}} &= \frac{\delta_\tau+\alpha_\tau}{\alpha_\tau}
\end{aligned}$$

Case 3: Deletion of a batch where data is from Model 6.2

From Equation 6.12 and Equation 6.14,

$$\begin{aligned}
\frac{\delta_{\tau+1}}{\alpha_{\tau+1}} &= \frac{\frac{\delta_\tau(\lambda_\tau-B(\sigma_2^2+\epsilon))^2}{(\lambda_\tau-B(\sigma_2^2+\epsilon))^2-\gamma_\tau(2B\lambda_\tau-B^2(1+2\sigma_2^2+2\epsilon))}}{\frac{\alpha_\tau(\lambda_\tau-B(\sigma_2^2+\epsilon))^2}{(\lambda_\tau-B(\sigma_2^2+\epsilon))^2-\gamma_\tau(2B\lambda_\tau-B^2(1+2\sigma_2^2+2\epsilon))}} \\
\frac{\delta_{\tau+1}}{\alpha_{\tau+1}} &= \frac{\delta_\tau}{\alpha_\tau} \\
\frac{\delta_{\tau+1}+\alpha_{\tau+1}}{\alpha_{\tau+1}} &= \frac{\delta_\tau+\alpha_\tau}{\alpha_\tau}
\end{aligned}$$

□

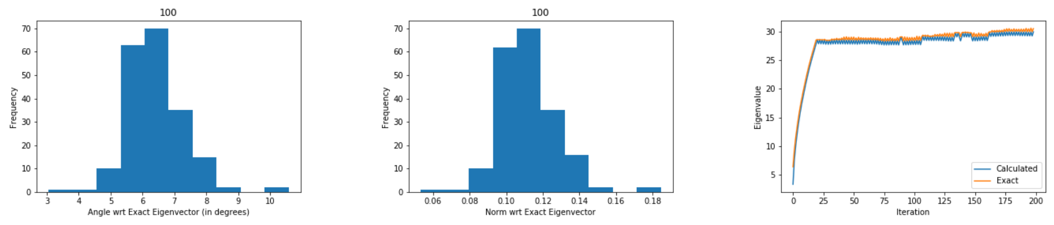
6.0.5 Results

We implemented the proposed algorithm in Python3.6 using the numpy library. We demonstrate the performance of the proposed algorithm on an open source dataset, namely, `MNIST`. It consists of images of handwritten digits. This indicates that the algorithm performs well on data other than those generated from the *orthogonal dual spiked covariance model.*

For every batch operation, we choose to perform either an insertion or a deletion. Results were obtained for the following parameters.

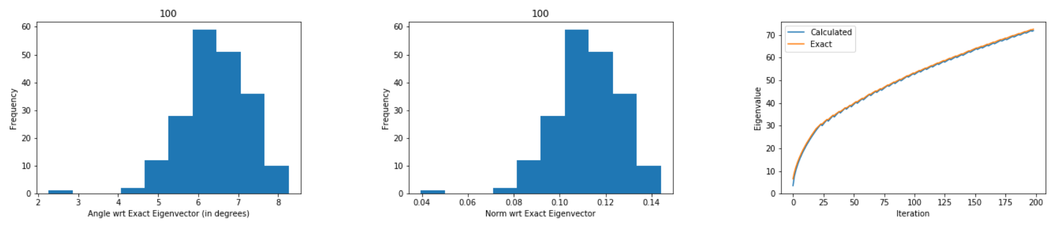
1. Batch sizes: 100, 200.
2. Probability of insertion: 0.5, 0.8.
3. Total number of points inserted or deleted: 20,000 and 40,000.

We measure the performance of our algorithm on the basis of i) angle of the computed eigenvector with respect to the exact eigenvector ii) norm of the computed eigenvector



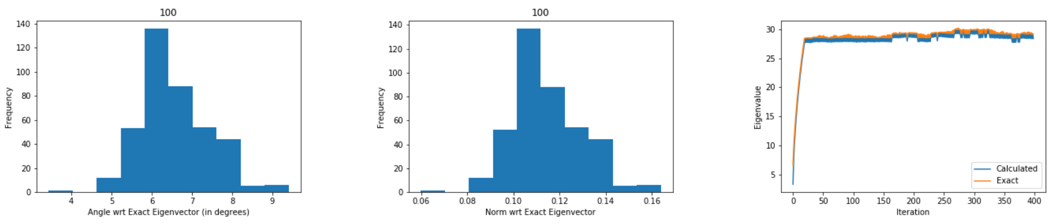
Angle Error (degrees) Norm Error Eigenvalues

50% Insertions, Points processed $= 20$k



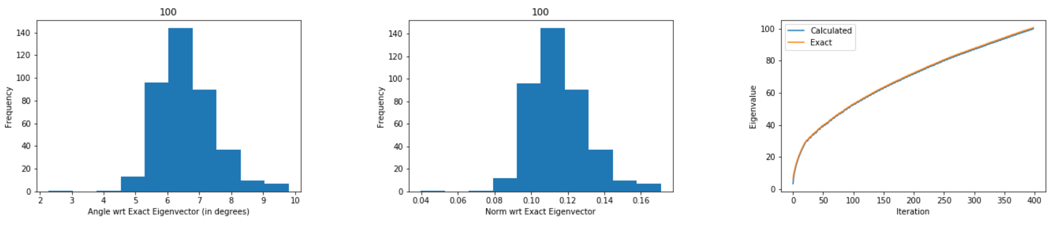
Angle Error (degrees) Norm Error Eigenvalues

80% Insertions, Points processed $= 20$k



Angle Error (degrees) Norm Error Eigenvalues

50% Insertions, Points processed $= 40$k



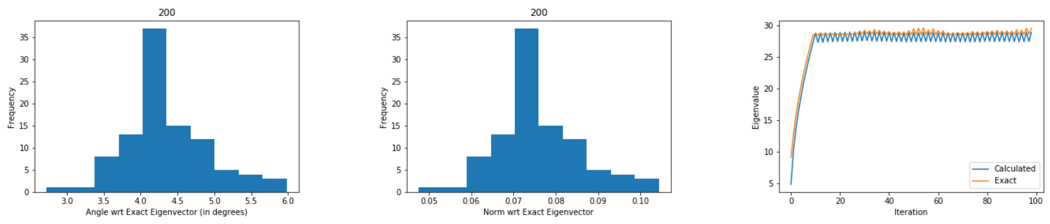
Angle Error (degrees) Norm Error Eigenvalues

80% Insertions, Points processed $= 40$k

FIGURE 6.1: Results on the MNIST Dataset with Batch Size $= 100$

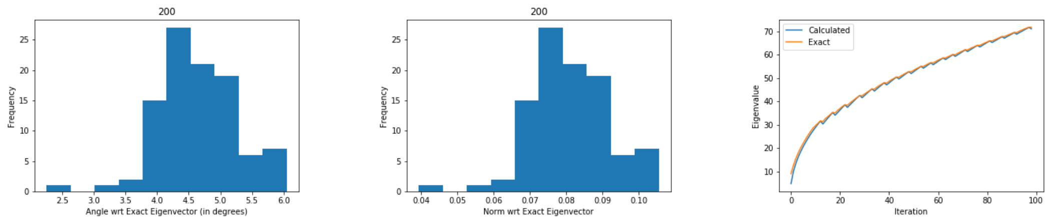
with respect to the exact eigenvector. We also track the eigenvalue after every batch of insertion or deletion.

As observed from Figure 6.1 and Figure 6.2, we observe that the proposed algorithm tracks the topmost eigenvector with little error. For batch size, $B = 100$ and $B = 200$, the angle error is bounded by 10 degrees and 6 degrees respectively. It is also evident that the eigenvalue of the data stream is approximated reasonably well. We can also observe that the error distributions appear almost gaussian in nature with mean close to 6 degrees for batch size $B = 100$, and 4.5 degrees for $B = 200$. For all plots, the errors shown are at each iteration. Plots for insertion and deletion are not shown separately.



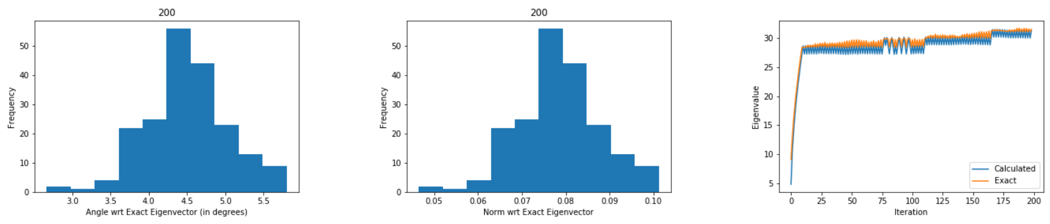
Angle Error (degrees) Norm Error Eigenvalues

50% Insertions, Points processed = 20k



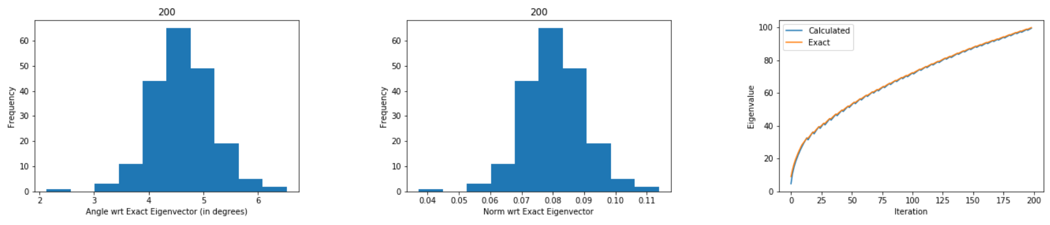
Angle Error (degrees) Norm Error Eigenvalues

80% Insertions, Points processed = 20k



Angle Error (degrees) Norm Error Eigenvalues

50% Insertions, Points processed = 40k



Angle Error (degrees) Norm Error Eigenvalues

80% Insertions, Points processed = 40k

FIGURE 6.2: Results on the MNIST Dataset with Batch Size = 200

6.0.6 Rank-k Updates

We also propose an algorithm for maintaining the top-k eigenvectors in a dyanmic stream as defined in Section 6.0.2. The pseudocode is presented in Algorithm 11 and Algorithm 12.

We also show some result for the error in the first and second eigenvector estimation. We process 20,000 operations in a batch size of 200 with probability of insert = 0.5 as shown in Figure 6.3 and probability of having an insert operation = 0.8 as shown in Figure 6.4. Though the error in estimation of the second largest eigenvector is higher than that of the largest eigenvector, the error is still reasonable. In Figure 6.3, we see that the error

Algorithm 11 Rank-k updates on Insertion

```
1: procedure INSERT($B, Q, \lambda$)
2:     $S \leftarrow 0_{k \times p}$
3:     for $\mathbf{x_t} \in B$ do
4:         $Y \leftarrow < Q, \mathbf{x_t} > \mathbf{x_t}$
5:         $S \leftarrow S + Y$
6:         $\lambda \leftarrow \lambda + < Q, Y >$
7:     $S \leftarrow \frac{1}{B} S$
8:     $Q, R \leftarrow QR(S)$
9:     return $Q, \lambda$
```

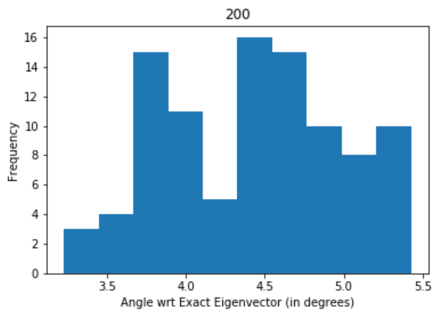
Algorithm 12 Rank-k updates on Deletion

```
1: procedure DELETE($B, Q, \lambda$)
2:     $S \leftarrow 0_{k \times p}$
3:     $\lambda_\tau \leftarrow 0$
4:     for $\mathbf{x_t} \in B$ do
5:         $Y \leftarrow < Q, \mathbf{x_t} > \mathbf{x_t}$
6:         $S \leftarrow S + Y$
7:         $\lambda_\mathbf{t} \leftarrow \lambda_\mathbf{t} + < Q, Y >$
8:     $S \leftarrow \lambda Q - S$
9:     $Q, R \leftarrow QR(S)$
10:    $\lambda \leftarrow \lambda - \lambda_\mathbf{t}$
11:    return $Q, \lambda$
```

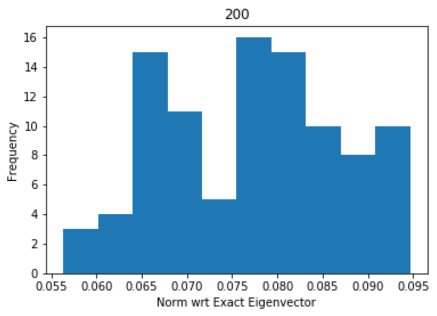
is bounded by a value around 20 degrees, but this occurs very infrequently. Similarly, for Figure 6.4, the frequency of occurence of high errors is quite rare. For all plots, the errors shown are at each iteration. Plots for insertion and deletion are not shown separately.

6.0.7 Conclusion

In this chapter, we presented an algorithm for online eigenvectors. The theoretical analysis of the proposed algorithm was performed on the *orthogonal dual spiked covariance model* described in Section 6.0.2. We demonstrate the performance of the algorithm on an open source dataset with various parameters and show that it performs well on real data as well.

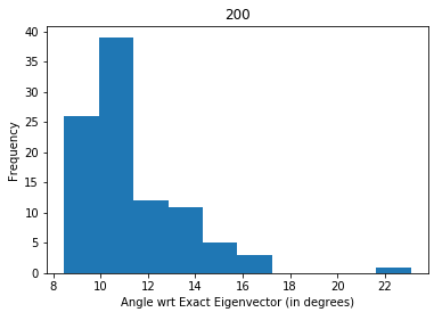


Error in degrees

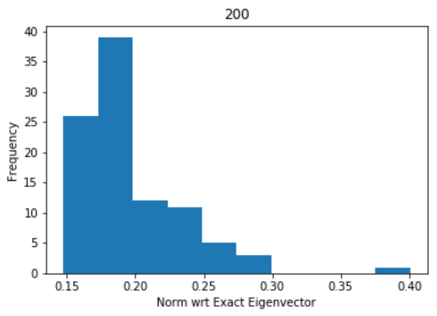


Norm Error

Errors in the largest eigenvector estimation

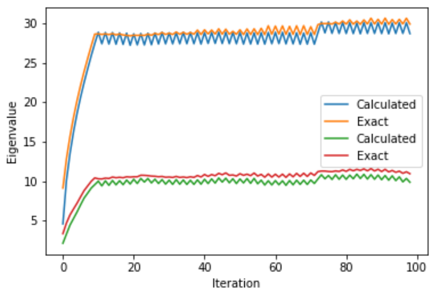


Error in degrees



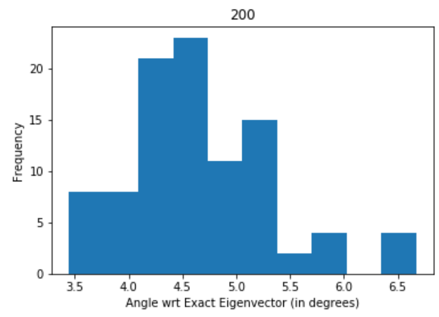
Norm Error

Errors in the second largest eigenvector estimation

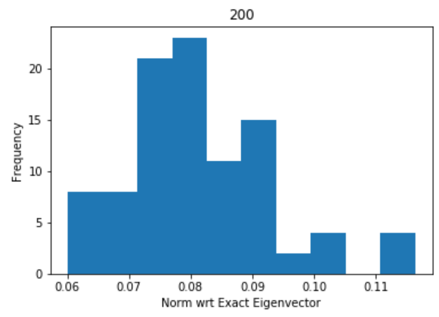


Eigenvalues

FIGURE 6.3: Results on the MNIST Dataset with 50% insertions for top-2 eigenvectors

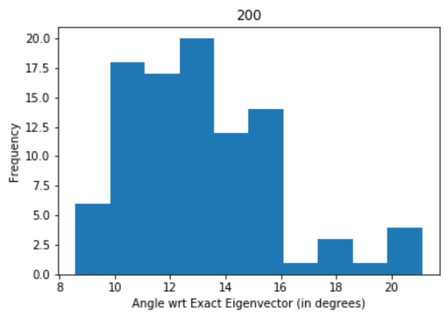


Angle Error (degrees)

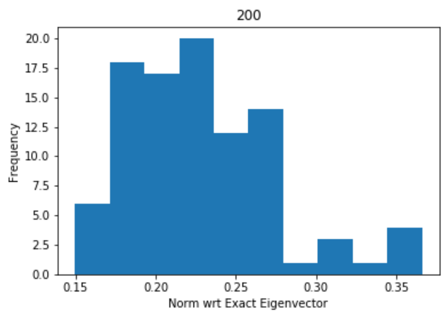


Norm Error

Errors in the largest eigenvector estimation

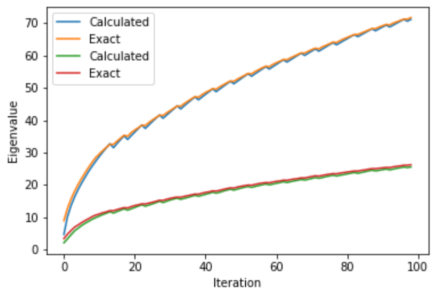


Angle Error (degrees)



Norm Error

Errors in the second largest eigenvector estimation



Eigenvalues

FIGURE 6.4: Results on the MNIST Dataset with 80% insertions for top-2 eigenvectors

Conclusion

In this thesis, we propose splitting rules for the hierarchical clustering. These splitting rules have a provable worst case guarantee and are efficient in practice. Our techniques have approximation guarantees for a specific objective. We demonstrate the usefulness of the partitions created in classification and anomaly detection tasks that require real-time response and have a variety of class labels. The techniques empirically outperform all reasonable baselines on large datasets, matching the classification guarantees while having a logarithmic query time.

We then propose two naive heuristics for the online hierarchical clustering problem and demonstrate their excellent empirical performance with minimal quality loss as compared to their batch counterparts. We also identify a natural sub-problem, namely, the online eigenvector updation problem. We propose an algorithm to update the largest eigenvector under insertion as well as deletion operations, show some theoretical guarantees and the excellent performance of the algorithm on the MNIST dataset. We also propose an algorithm to maintain the top-k eigenvectors under a dynamic stream of operations. We do not show any theoretical guarantees for the same.

7.0.1 Future Work

This work can be extended to give theoretical bounds for the online updations of the top-k eigenvectors, apply this to online hierarchical clustering and prove some theoretical bound on the quality of the solutions provided. A related area would be to look at distributed hierarchical clustering algorithms with provable worst case guarantees.

Bibliography

Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. 2015. Practical and Optimal LSH for Angular Distance. *CoRR* abs/1509.02897 (2015). arXiv:1509.02897 `http://arxiv.org/abs/1509.02897`

Sanjeev Arora, Satish Rao, and Umesh Vazirani. 2009. Expander flows, geometric embeddings and graph partitioning. *Journal of the ACM (JACM)* 56, 2 (2009), 5.

David Arthur and Sergei Vassilvitskii. 2007. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 1027–1035.

Mohammadhossein Bateni, Soheil Behnezhad, Mahsa Derakhshan, MohammadTaghi Hajiaghayi, Raimondas Kiveris, Silvio Lattanzi, and Vahab Mirrokni. 2017. Affinity Clustering: Hierarchical Clustering at Scale. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 6864–6874. `http://papers.nips.cc/paper/7262-affinity-clustering-hierarchical-clustering-at-scale.pdf`

Jon Louis Bentley. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM* 18, 9 (Sept. 1975), 509–517. `https://doi.org/10.1145/361002.361007`

Moses Charikar and Vaggos Chatziafratis. 2017. Approximate Hierarchical Clustering via Sparsest Cut and Spreading Metrics. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '17)*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 841–854. `http://dl.acm.org/citation.cfm?id=3039686.3039739`

Kenneth L Clarkson and David P Woodruff. 2009. Numerical linear algebra in the streaming model. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. ACM, 205–214.

Vincent Cohen-Addad, Varun Kanade, Frederik Mallmann-Trenn, and Claire Mathieu. 2017. Hierarchical Clustering: Objective Functions and Algorithms. *CoRR* abs/1704.02147 (2017). arXiv:1704.02147 `http://arxiv.org/abs/1704.02147`

Sanjoy Dasgupta. 2016. A Cost Function for Similarity-based Hierarchical Clustering. In *Proceedings of the Forty-eighth Annual ACM Symposium on Theory of Computing (STOC '16)*. ACM, New York, NY, USA, 118–127. `https://doi.org/10.1145/2897518.2897527`

Dua Dheeru and Efi Karra Taniskidou. 2017. UCI Machine Learning Repository. `http://archive.ics.uci.edu/ml`

Dan Garber, Elad Hazan, and Tengyu Ma. 2015. Online Learning of Eigenvectors. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Francis Bach and David Blei (Eds.), Vol. 37. PMLR, Lille, France, 560–568. `http://proceedings.mlr.press/v37/garberb15.html`

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. 2011. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review* 53, 2 (2011), 217–288.

Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC '98)*. ACM, New York, NY, USA, 604–613. `https://doi.org/10.1145/276698.276876`

Ari Kobren, Nicholas Monath, Akshay Krishnamurthy, and Andrew McCallum. 2017. A hierarchical algorithm for extreme clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 255–264.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

Tom Leighton and Satish Rao. 1999. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM (JACM)* 46, 6 (1999), 787–832.

Mark McCartin-Lim, Andrew McGregor, and Rui Wang. 2012. Approximate Principal Direction Trees. *CoRR* abs/1206.4668 (2012). arXiv:1206.4668 `http://arxiv.org/abs/1206.4668`

Frank McSherry. 2001. Spectral partitioning of random graphs. In *Proceedings 2001 IEEE International Conference on Cluster Computing*. IEEE, 529–537.

Ioannis Mitliagkas, Constantine Caramanis, and Prateek Jain. 2013. Memory Limited, Streaming PCA. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'13)*. Curran Associates Inc., USA, 2886–2894. `http://dl.acm.org/citation.cfm?id=2999792.2999934`

Nicholas Monath, Ari Kobren, Akshay Krishnamurthy, and Andrew McCallum. 2017. Gradient-based Hierarchical Clustering.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, Doha, Qatar, 1532–1543. `https://doi.org/10.3115/v1/D14-1162`

Anderson Rocha and Siome Klein Goldenstein. 2014. Multiclass from binary: Expanding one-versus-all, one-versus-one and ecoc-based approaches. *IEEE Transactions on Neural Networks and Learning Systems* 25, 2 (2014), 289–302.

Luca Trevisan. 2013. Lecture notes on expansion, sparsest cut, and spectral graph theory. , 81 pages. `https://people.eecs.berkeley.edu/~luca/books/expanders.pdf`