Kernel Function Learning for Graph Structure Formation (Graph Anomaly Detection)

Aniruddha Basak
aniruddha.basak@sv.cmu.edu
Carnegie Mellon University
Moffett Field, CA

Pragya Tripathi
pragya.tripathi@sv.cmu.edu
Carnegie Mellon University
Moffett Field, CA
Team 10, C1 Climate Change

Meng Wu
meng.wu@sv.cmu.edu
Carnegie Mellon University
Moffett Field, CA
Team 10, C1 Climate Change

ABSTRACT

Graph anomaly detection is often used in practice as structural information is added in a graph along with local information about every node. In real world applications, constructing a graph from data is the first task and needs to be done before any analysis. For the purpose of anomaly detection, it is important that the node relationships or edges are created in such a way that anomalous changes in graph structures can be captured well by algorithms that can detect node relationship changes. There are anomaly detection algorithms like CAD which can successfully identify anomalous nodes from series of graphs. However, constructing a graph from multivariate data is a challenge. In this paper, we develop and evaluate machine learning methods for predicting kernel bandwidth for converting unstructured data set into graph-structured data set.

1. INTRODUCTION (MENG)

There are diverse ways to analyze unstructured data sets, and a common technique to summarize such data is through graphs. Graph-structured data appears in many modern applications like social networks, health care, transportation networks and computer graphics. Although a graph can easily represent relations, discovering knowledge from graph-structured data poses a general problem for learning from structured data.

When analyzing large and complex graph-structured data sets, knowing what stands out in the data is more important and interesting than learning about its general structure. Finding anomalies in graphs is necessary to solve diverse problems such as security, finance, and law enforcement.

The algorithm of detecting anomalous nodes in graphs, which is called CAD (Commute-time based Anomaly detection in Dynamic graphs), has been provided. Compared to some studies that designed to detect when anomalous changes in graph structure occur, CAD algorithm could identify changing node relationships that contribute to such anomalous changes in graph structure.[5]

This project aims to provide a general, comprehensive way to convert unstructured data set into graph-structured data set using machine learning methods. The motivation for our work comes from the problem of paper *Localizing anomalous changes in time-evolving graphs* of not having a proper mechanism to select a kernel function for forming graphs that contain anomalies that CAD can detect.

DOI: 10.1145/1235

2. BACKGROUND (MENG)

The project is based on CAD algorithm, introduced by paper *Localizing anomalous changes in time-evolving graphs*. We will introduce CAD algorithm and a previous way of forming dynamic graph based on unstructured data sets in this section.

2.1 CAD algorithm

Multiple methods are presented to detect anomalies in dynamic graphs. However, their common limitation is that they are designed for event detection, which means, they detect a set of nodes including nodes that cause anomalous changes in node relationships as well as non-anomalous nodes that either are affected by structure changes or cause slight changes in graph structure.[5]

CAD successfully avoids those false alarms. The aim of CAD algorithm is to identify changes in node relationships that are responsible for anomaly changes in graph structure.[5] Although CAD should be black box to us, but better understanding of CAD could help our project.

The input of CAD is a time series of graphs $G_t, t = 1, \ldots, n$. Each G_t is a weighted undirected graph with a set of nodes $V = (v_1, \ldots, v_n)$ and a set of edges $E = (e(1,1), \ldots, e(n,n))$ of size n^2, and $e(i,j)$ indicates the edge between nodes vi and v_j. Given a series of graph instances G_t and $G_t + 1$, CAD will output the set of anomalous edges $E_t \subset E$ with edge scores whose change in weights contributes to structural differences between G_t and $G_t + 1$.[5]

CAD localizes anomalous changes in graph structure via combines information regarding changes of commute time distance and changes in edge weight. The definition of commute time distance between any pair of nodes is the average time needed to traverse the distance between these two nodes via random walks.[5] CAD computes the edge score based on the following formula.

$$\Delta E(e(i,j)) = |A_t + 1(i,j) - A_t(i,j)| \times |c_t + 1(i,j) - c_t(i,j)| \quad (1)$$

with $c_t(i,j)$ and $c_t + 1(i,j)$ separately indicating commute time distance between node i and node j of time slice t and time slice t+1, $A_t(i,j)$ and $A_t+1(i,j)$ indicating edge weight between node i and node j of time slice t and time slice t+1.

2.2 Previous Work in Graph Formation

According to the method of forming graphs introduced by paper *Localizing anomalous changes in time-evolving graphs*, the data set consists of four groups of clusters that are generated from 2-dimensional Gaussian mixture distribution. A adjacency matrix P is constructed where $P(i,j) = exp(-d(i,j))$ with $d(i,j)$ corresponding to Euclidean distance between node i and node j in the data set. And adjacency matrix P is considered as the graph instance of time slice t.

To generate graph instance of t+1, we add two types of noises to matrix P. Adjacency matrix Q is considered as white noises, which will not change graph structure. Random matrix R that is expected to contribute to changing graph structure is constructed, where $R(i,j)$ is given by

$$R(i,j) = \begin{cases} 0 & \text{with probability } p = 0.95 \\ u(i,j) & \text{with probability } p = 0.05 \end{cases}$$

with $u(i,j)$ is a random number between 0 and 1. We consider graph instance of time slice t $G_t = P$ and graph instance of time slice t+1 $G_t + 1 = P + Q + (R + R')/2$. Thus, edges in P that are combined with non-zero $R(i,j)$ will be regarded as anomalous edges

Figure 1 and Figure 2 shows the adjacency matrix without and with noise. The color indicated the degree of edge weight

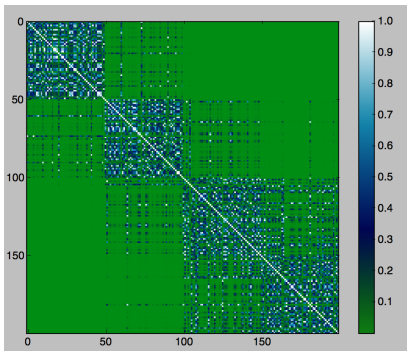

Figure 1: Graph from random realizations from 4-component Gaussian mixture before adding noises

The way of forming graphs introduced by paper *Localizing anomalous changes in time-evolving graphs* has a limitation: although it could effectively add anomalous changes that CAD can detect to the graph,there is no effective way to regenerate a graph using a new kernel function with the same structure change as previous graphs . Since an essential step of our project is to keep generating new graphs with updated kernel function until the current kernel function is close to ground truth kernel function, noises are supposed to be added to data sets before generating graphs instead of that noises are added to existed graphs.

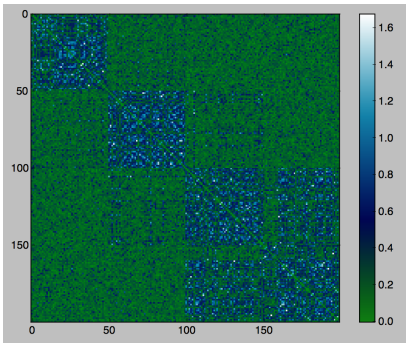

Figure 2: Graph from random realizations from 4-component Gaussian mixture after adding noises

2.3 Previous work on kernel estimation (PRAGYA)

Upon literature review, we found a lot of interesting novel techniques used for kernel estimation. However, none seemed to replicate what we were trying to do in the project. Through paper called *Very fast optimal bandwidth selection for univariate kernel density estimation*, we found a novel technique for exact approximation of kernel but it could only be applied in a univariate Gaussian distribution.[4] Another research done at *University of Michigan* proposed increasing robustness to non-parametric data using M-estimator, an estimator based on decreasing loss function.[1] Though it seems quite close to our project, the main focus in the paper was its sole application to non-parametric data. However, the paper provides some interesting comparisons of different kernel functions which we take into consideration in Section 4.2.

3. ALGORITHM (MENG)

A graph is constructed based on unstructured data set and relative kernel function. To find out the kernel bandwidth for constructing graphs, our program will keep regenerating graphs with updated kernel function.

3.1 Graph Preparation

Prior to generating graphs, two data sets in the time slice t and time slice t+1 are needed. Synthesizing a time series of data sets is the first step.

Let D_t be a data set that consists of four clusters generated based on multivariate Gaussian distribution function. Each cluster are loosely connected, while among each cluster, nodes are tightly coupled. Define node set V = $(v_1, v_2, \ldots, v_n)$. To simulate each node has two features, each node in D_t is assigned two values, which are node's X coordinates and Y coordinates.

In the time slice t+1, we add two types of noise, white noise and normal noise to D_t, simply via adding value of noise to value of each node in D_t. The purpose is to synthesize data set of time slice t+1 $D_t + 1$. White noise is

generated based on multivariate Gaussian distribution with

$$\mu = \begin{vmatrix} 0 & 0 \end{vmatrix}$$

and

$$cov = \begin{vmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{vmatrix}$$

Node set in $D_t + 1$ remains the same as in D_t. Node value might change.

While white noise is not expected to detect by CAD, it is used to validate if CAD can ignore irrelevant nodes that are not responsible for changes for graph structure. Normal noises are responsible for causing changes in graph structure and they are located in the center of the four cluster. We keep track of nodes combined with normal noises and they are regarded as ground truth anomalous nodes S and S $\subset V$.

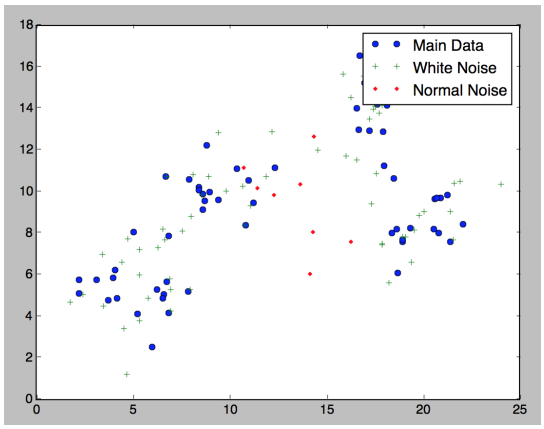

Figure 3: Random realizations from 4-component Gaussian mixture

Let G_t and $G_t + 1$ be graphs of time slice t and time slice t+1. G_t and $G_t + 1$ are weighted undirected graphs generated from node set in D_t and node set in $D_t + 1$separately. Define the edge set $E = (e_(1, 1), \ldots, e_(n, n))$ of size n^2, with $e_(i, j)$ denoting the edge between nodes v_i and v_j. Edge strength between a node i and a node j is given by radial basis function kernel $exp\frac{-||p_i - p_j||^2}{2\sigma^2}$ with $||p_i - p_j||^2$ corresponding to Euclidean distance between node i and node j. X value and Y value of each node need to be normalized before calculating the Euclidean distance to make sure they have the same scale. Ground truth anomalous edges and their relative edge scores are recorded. The anomaly score for each node $v_i \in V$ for a transition from t to t+1 is given by $\Sigma E_t(e(i, j), i \in (1, \ldots, n), j \in (1, \ldots, n))$ They are regarded as the standard to verify if CAD returns expected anomalous edges.

3.2 Learning Engine

CAD can effectively identify anomalous changes with a time-series of informative graphs. Since graphs generated from improper kernel function cannot provide useful information about the unstructured data sets, it is not likely that CAD can find out ground truth anomalous nodes given these graphs.

Suppose ground truth kernel function is unknown. To get the kernel bandwidth, we start from a randomly-chosen kernel function σ_0. Graphs G_t and $G_t + 1$ are generated based on σ_0. CAD detects anomalous edges from G_t and $G_t + 1$. There might exist difference between ground truth anomalous edges and anomalous edges returned by CAD because σ_0 is not ground truth σ. Loss function could be utilized to measure the similarity between ground truth anomalous edges and anomalous edges returned by CAD.

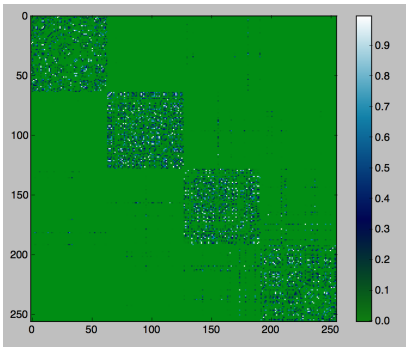

Figure 4: Adjacency matrix for G_t

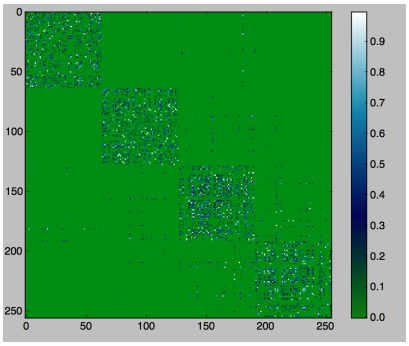

Figure 5: Adjacency matrix for $G_t + 1$

3.2.1 Loss Function

Loss function is utilized to measure the degree of fit between ground truth anomalous edges and anomalous edges returned by CAD. There are two ways we could use to form loss function. We could take the loss function to be the result of comparing ranks of ground truth anomalous edges and ranks of anomalous edges returned by CAD.

$$LossFunction_1 = \Sigma(\Delta R_G(i, j) - \Delta R(i - j))^2 \quad (2)$$

where $\Delta R_G(i, j)$ denotes ground truth edge ranks and $\Delta R(i, j)$ denotes edge ranks returned by CAD

We could also compute the loss function by comparing edges score of ground truth anomalous edges and edge score of anomalous edges returned by CAD.

$$LossFunction_2 = \Sigma(\Delta E_G(i, j) - \Delta E(i, j))^2 \quad (3)$$

where $\Delta E_G(i, j)$ denotes ground truth edge scores and $\Delta E(i, j)$ denotes edge scores returned by CAD

3.2.2 Loss Function Optimization

We pick the loss function that incorporates anomaly scores because the anomaly scores are computed using edges formed by the kernel. When loss function approaches to 0, it indicates high similarity between ground truth anomalous edges

and anomalous edges returned by CAD, which reflects that we are close to ground truth σ. Multiple ways could be applied to minimize loss function, such as gradient descent and genetic algorithm. We select gradient descent for simplicity first and compare it later with other techniques. The basic working flow of gradient descent to approach the ground truth σ is shown below:

- Randomly pick a value between a given range and set it as the value of σ, which might far from the ground truth σ
- Generate graphs G_t based on D_t and radial base function kernel, $exp\frac{-||p_i-p_j||^2}{2\sigma^2}$ with $||p_i - p_j||^2$ corresponding to Euclidean distance between node i and node j in D_t.
- Generate graphs G_t+1 based on D_t+1 and radial base function kernel, $exp\frac{-||p_i-p_j||^2}{2\sigma^2}$ with $||p_i - p_j||^2$ corresponding to Euclidean distance between node i and node j in $D_t + 1$.
- Using loss function to measure the similarity between ground truth anomalous edges and anomalous edges returned by CAD.
- Update σ based on $\sigma = \sigma$ - learning rate*lossfunction
- Back to step 2
- If loss function is lower than a threshold δ, the gradient decent loop breaks.

4. EXPERIMENTS (PRAGYA)

In this section, we present results from several experiments conducted with synthetic data. The data has been generated from the process described in Section 3.1. The goal behind these experiments are to:

1. Prove that the initial concept of using machine learning to learn kernel function works
2. Ensure the proposed method is robust enough to handle different types of kernel functions.

The experiment settings are described on each section below. We have additional results from optimizing various features of our learning engine like loss minimization function and kernel function used for graph formation in later sub-sections.

4.1 Experiments with Synthetic data

The major goal behind experimenting with Synthetic data is to ensure that the learning engine is performing as expected and generating speedier results. Synthetic data allows us to run the experiments in a controlled environment and expedite run time for CAD and learning engine so that multiple restarts can be done fairly quickly.

The experiments were run in multivariate synthetic data of length 64. Learning rate(η) for gradient descent was set to 0.2. Threshold(δ) and embedding dimension(d), the input variables for CAD have been set to $\delta = 0.1$, and $d = 3$. The ground truth σ is $\frac{1}{\sqrt{2}}$ which causes the kernel function to become $\exp -||p_i - p_j||^2$. The cluster environment used has been described in the Appendix.

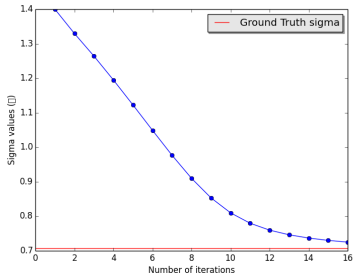

Figure 6: A program run with initial σ 1.4 and $\eta = 0.2$

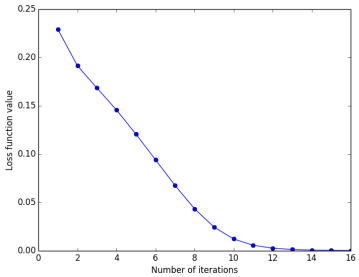

Figure 7: Loss function values for Figure 6

For Figure 6, the initial σ is 1.4. As it can be seen from the figure, the σ at each iteration slowly decreases and comes quite close to ground truth σ value. Similarly, the corresponding loss function in Figure 7 eventually converges to zero. The stopping criteria is met on 16^{th} iteration and the engine predicts kernel bandwidth to be 0.725. While it is easy to postulate that our method works, it must be considered that the initial σ was higher than the ground truth. For Figure 8, the initial σ starts with a lower value, 0.4, and while keeping all the other variables the same, we found that the σ value increases as required and comes close to the ground truth. Similarly, its corresponding loss function in Figure 9 converges to 0 in 10 iterations. This is an assurance that gradient descent is working.

Next, we do multiple random restarts with wide range of sigma values. Through Figure 10, we find that not all random restarts result in values closer to ground truth. However, through the box plot, we find that the average of σ values at subsequent iteration do converge to ground truth. Looking at their corresponding loss function values in Figure 11, all of the loss function values eventually try to converge to 0. For a random restart with initial $\sigma = 1.4$, even it doesn't immediately converge to ground truth, the decreasing loss function insinuates that it will eventually converge. We acknowledge, however, that Figure 10 doesn't have significantly high initial σ value. We have such results kept separately in Figure 12. The reason behind our choice is to avoid appending difficult to read plots with extreme values. The σ values in Figure 12 and their corresponding loss function values in Figure 13 show that the changes are not significant when the initial σ value is higher. While the values are changing, given the program has run for 100 iterations, the convergence is not satisfactory and optimization

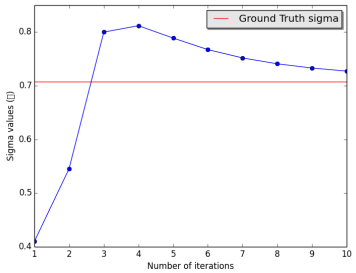

Figure 8: A program run with initial σ 0.4 and $\eta = 0.2$

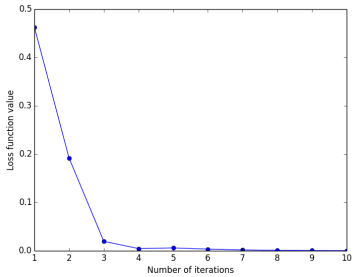

Figure 9: Loss function values for Figure 8

must be done to ensure robustness of our algorithm. We further discuss ways to improve in section below.

4.2 Experiments with Optimization

From the above experiments, it is quite clear that our proposed method only performs well with smaller initialization of σ. We need to take time as an important aspect to be optimized. Code optimization is outside of the scope of this paper. Although the necessary code changes were made in improving run time and there could be many other optimization that could be done, we mainly focus on improving loss minimization technique as it can affect run time significantly.

Besides optimization, we have known so far that our proposed method works only well while using same kernel function for finding ground truth kernel and learning kernel. However, we need to understand what happens when the ground truth kernel is different than the learning kernel and the correct kernel bandwidth is unknown. In subsequent subsections, we analyze our methods and optimize two important facets in it: loss minimization technique and kernel function.

4.2.1 Loss function minimization

Our initial reason for picking Gradient Descent as our loss function minimization technique was its simplicity and ability to minimize the loss function gradually. Although Gradient Descent gave us expected results, it is inadequate in terms of making our algorithm robust. Its chances of quickly converging to ground truth are highly dependent on the initial σ value and choice of learning rate. It is also prone to being stuck in local minima in the search space. Besides that the learning rate used in the descent step often guides it and its tuning is very important on how Gradient Descent

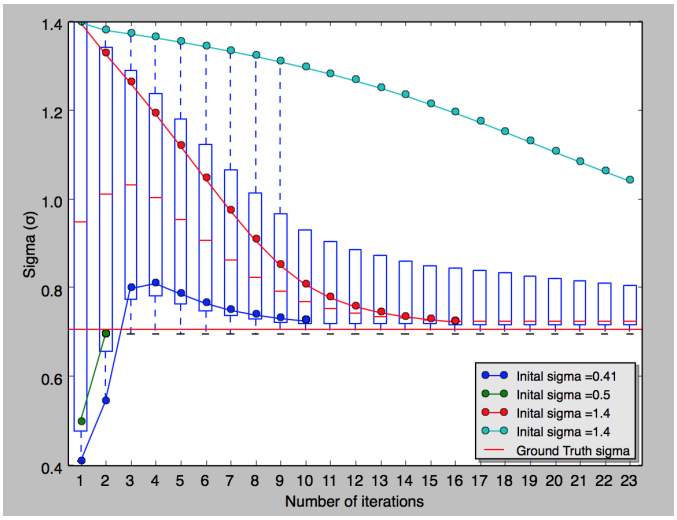

Figure 10: A box plot of σ values for multiple restarts with $\eta = 0.2$

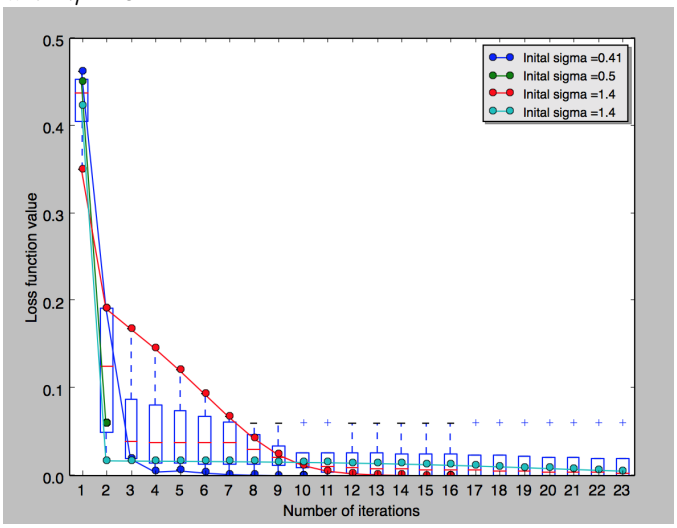

Figure 11: A box plot of corresponding loss function values for Figure 10

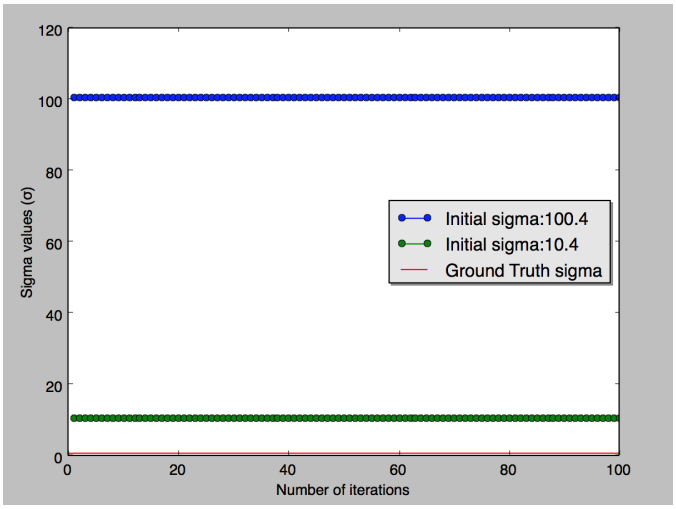

Figure 12: A box plot of σ values for higher initial σ values with $\eta = 0.2$

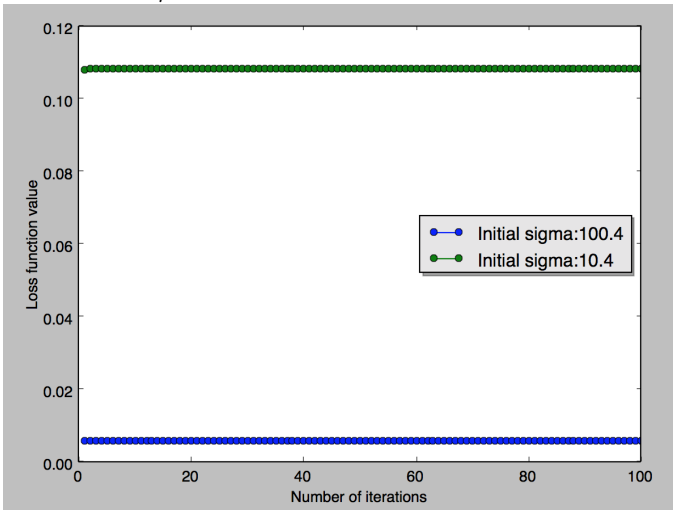

Figure 13: A box plot of corresponding loss function values for Figure 12

Table 1: Comparison of three loss minimization techniques with initial $\sigma = 1.4$, ground truth $\sigma = 0.707$

Variations	Gradient Descent	Nelder-Mead	Genetic Algorithm
Number of runs	16 iterations	6 iterations with 9 function evaluations	3 generations with population size 4
Time taken	55.17 m	20.63 m	16.64 m
Final σ value	0.725	0.702	0.7

performs. Since the learning rate is used to prevent overfitting, adjusting learning rate may not entirely solve the main problem of adding ability to handle any initialization value of σ as seen in Figure 12 and Figure 13. [2]

To replace Gradient Descent, we use two diverse yet interesting methods to minimize our loss function: Genetic Algorithm and Nelder-Mead algorithm. Genetic Algorithms are evolutionary methods based algorithm used for optimization problems and Nelder-Mead methods are used to maximize or minimize objective function through heuristic search methods. Genetic Algorithm's main advantage is to be able to find optimal solution (global minima) quickly[6] while Nelder-Mead algorithm, through use of its heuristic search, doesn't use a learning rate and extrapolates behavior of loss function for different values of given parameter[3] to eventually minimize the loss function.

For Genetic Algorithms, we are not able to use crossover techniques since we are optimizing only one parameter, σ. The mutation rate is set to 0.25 and Gaussian mutation technique is used. The population size is set to 4 and the algorithm is tuned to be able to minimize the loss function. For Nelder-Mead algorithm, error tolerance is set to 1e-4. The algorithm is fairly straightforward to use and doesn't require more parameters. The evaluation function used for both algorithms receives a new σ value generated through the respective minimization technique, uses it to form graphs and run CAD. Once CAD gives new anomaly scores and anomalous nodes, the anomalous nodes are compared against ground truth. If they match, the evaluation function always returns 0. Otherwise, loss function is used as usual to rate the performance of σ. The experiments are performed using similar synthetic data and CAD logistics described in Section 4.1.

Table 1 presents comparison of time, final σ value and number of runs comparison for all three algorithms with initial value of 1.4. Table 2 presents similar comparison but with a higher σ value, 100. Based on these results, Genetic Algorithms are quicker in terms of coming up with a σ value closer to ground truth value, while Nelder-Mead converges closest to ground-truth but not as fast as Genetic Algorithms. As known through previous experiments as well as these, Gradient Descent is incredibly slow and is no where close to convergence when initial σ is high.

Figure 14 presents interesting comparison of Gradient Descent and Nelder-Mead. It shows how Nelder-Mead gives accurate σ at the end of its run. Nelder-Mead takes huge steps when the loss function does not decrease. It makes

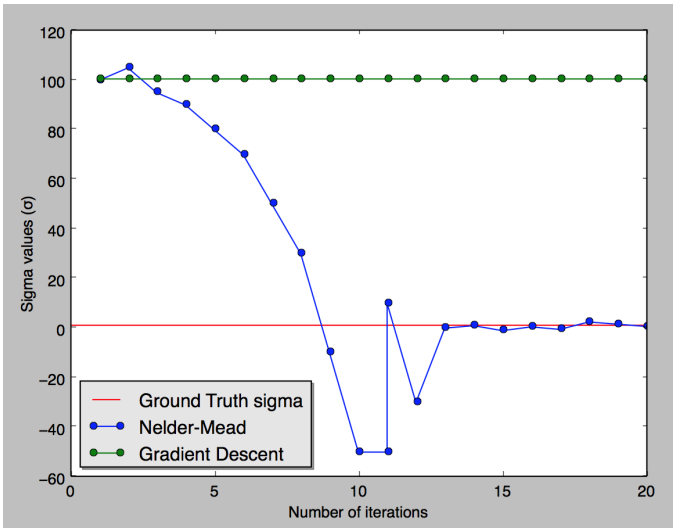

Figure 14: Comparison of σ values for Nelder-Mead with Gradient Descent

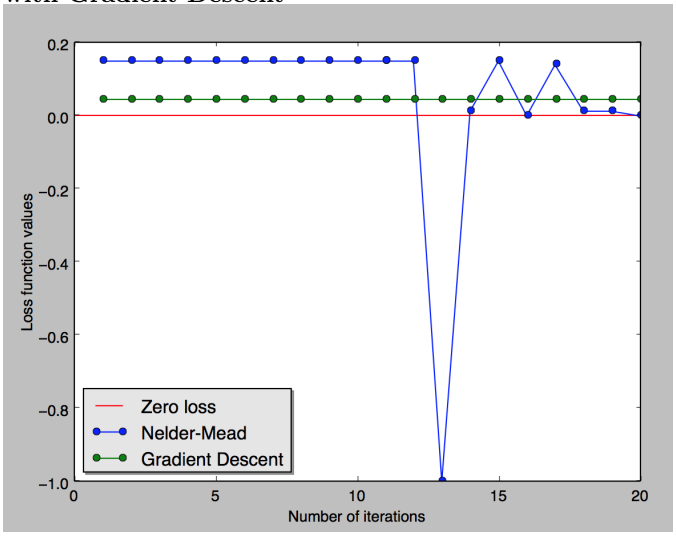

Figure 15: Corresponding loss function values for Figure 14

Table 2: Comparison of three loss minimization techniques with initial σ = 100, ground truth σ = 0.707

Variations	Gradient Descent	Nelder-Mead	Genetic Algorithm
Number of runs	100 iterations (manually stopped)	18 iterations with 48 function evaluations	3 generations with population size 4
Time taken	346.83 m	69.10 m	18.52 m
Final σ value	100.399	0.690	1.041

Table 3: Comparison of three loss minimization techniques with initial σ = 1.4, No ground truth σ

Variations	Gradient Descent	Nelder-Mead	Genetic Algorithm
Number of runs	100 iterations (manually stopped)	10 iterations with 20 function evaluations	6 generations with population size 4
Time taken	342.25 m	34.65 m	28.53 m
Final σ value	1.621	1.646	0.328

dramatic shifts in choosing the new σ value with the sole goal of minimizing loss. In Figure 15, there is an interesting dip of loss function value. It was caused by Nelder-Mead picking σ value to be 0. Sigma value cannot be zero as it is a division factor in radial basis kernel function. Therefore, our algorithm has been tuned to treat it as loss function value of -1.

4.2.2 *Kernel function*

So far all the experiments described above assume that radial basis kernel function is used as both ground-truth and learning kernel function. However, such assumption easily weakens our goal of creating an algorithm that can predict any kernel function. Thus, we use experiment with a different kernel function for ground truth and use radial basis kernel for the learning engine to understand how the algorithm can work. This means that there isn't a ground truth kernel bandwidth we can compare against to make sure the algorithm is working. Therefore, we have run this combination of kernel functions for all three loss minimization techniques we have used so far: gradient descent, Genetic algorithm and Nelder-Mead algorithm.

We have picked cosine kernel function for our ground truth because it has no kernel bandwidth. It is simply a dot product of two vectors divided by their normalized product. (cite)

$$cosine\_kernel = \frac{x.y^T}{||x||.||y||}$$

It results in smaller edge weights than radial basis kernel function as the result is not wrapped around an exponent. Therefore, CAD parameter for tolerance,δ, had to be tuned to 0.01 to make sure it captures the anomalous changes introduced in the ground truth phase of algorithm. The initial value of σ is taken to be 1.4 for learning engine which still uses radial basis kernel. The experiments are performed using similar synthetic data and logistics described in Section 4.1 except δ change.

Table 3 has results from using different kernel functions. Since the ground truth is unknown, the comparison of all three loss minimization technique helps to understand what the solution should be. Gradient Descent and Nelder-Mead both give similar values, but different than Genetic Algorithm. Even though Gradient descent and Nelder-Mead are both known to be stuck in local minima sometimes and Genetic Algorithms always finds global minima. Based on Nelder-Mead's convergence in experiments described in section 4.2.2, we have a good reason to believe that optimal σ

value is around that.

5. CONCLUSION AND FUTURE WORK (PRAGYA)

While graph formation is a challenging task, the results from our experiments corroborate that kernel bandwidth can be predicted by machine learning methods for graph formation. However, the use of learning methods requires careful analysis of loss minimization technique being used and kernel functions used for graph structure formation.

Since all of our experiments are based on synthetic data, this work needs to be extended further to be applied to real world data set. Due to problems in CAD and lack of time, we weren't able to get proper results. Furthermore, to decrease run-time and make graph formation speedier, cluster based system like Spark can be leveraged. Spark is already being used in CAD but can be extended to be used in our algorithm.

While we began this project with intention of using it to detect climate change, there are many diverse business and science work which may find anomaly detection through graph structures helpful. We hope our findings can shed some light in their quest of using machine learning techniques for graph formation.

6. ACKNOWLEDGEMENTS (PRAGYA)

We would like to thank Kamalika Das, Vinodh Paramesh and Dr. Ole Mengshoel for their immense help and support for this project.

7. REFERENCES

[1] JooSeuk Kim and Clayton D Scott. Robust kernel density estimation. *Journal of Machine Learning Research*, 13(Sep):2529–2565, 2012.

[2] Wouter M Koolen, Tim Van Erven, and Peter Grünwald. Learning the learning rate for prediction with expert advice. In *Advances in Neural Information Processing Systems*, pages 2294–2302, 2014.

[3] Ken IM McKinnon. Convergence of the nelder–mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158, 1998.

[4] Vikas Chandrakant Raykar and Ramani Duraiswami. Very fast optimal bandwidth selection for univariate kernel density estimation. 2006.

[5] Kumar Sricharan and Kamalika Das. Localizing anomalous changes in time-evolving graphs. *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*.

[6] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

APPENDIX

A. ENVIRONMENTS

We ran most of our experiments in a machine in cluster environment. Here are some details about the machine:

- HP proliant g6 360 server model
- 24 GB RAM
- 16 core
- OS Red Hat Enterprise Version 6.4
- Spark 2.1
- Python Anaconda 2.7
- Java 7

B. CAD CONFIGURATIONS

Besides the variables mentioned in Section 4.1, CAD program has few more variables that need to be set for it work properly. CAD program has a tolerance value(tol) used in various solvers. We have used tol = 1e-2. It has another variable to have minimum number of partitions($minP$) in Spark dataframe. We have used $minP = 10000$.

C. TECHNOLOGIES USED

- Python 2.7.10
  - Scipy 0.13.0b1
  - Numpy 1.8.0rc1
  - pyevolve 0.6
- Spark 2.0
- CAD (by Vinodh Paramesh and Aniruddha Basak)