Article

An Explanatory CDF Manifold Algorithm for Large Telecom Datasets

Vladislav Vasilev * and Georgi Iliev

Posted Date: 10 March 2026

doi: 10.20944/preprints202603.0819.v1

Keywords: manifold learning; high dimensional CDF; non-parametric machine learning; binary searches; dictionary sort; graph algorithms

 doi:10.20944/preprints202603.0819.v1

Article

An Explanatory CDF Manifold Algorithm for Large Telecom Datasets

Vladislav Vasilev and Georgi Iliev *

TU Sofia
* Correspondence: gli@tu-sofia.bg

Abstract

In this paper we introduce the CDF manifold algorithm that operates on data sets where a single target dimension is strictly increasing given a minimum of two or more number of input dimension which is very common in telco data. The manifold can then be used to compute the closest upper and lower limit to a given new point as well as its CDF. Training takes $O(n.ln[n])$ steps in the best case and $O(n^{3/2})$ in the worst case. Look up takes $O(ln[n])$ steps in the best case and $O(n^{1/2})$ in the worst case. The asymptotic computational cost is proven with a theorem. We compared our manifold method versus a standard dense neural network and show the asymptotic advantages both in terms of speed and accuracy. We also comment of potentials speed gains through the use of reference points. In summary, the manifold is a non-parametric and explanatory method to find the tightest data driven upper and lower limit of the output dimension given a new unseen input. This makes it ideal for planning new site deployments where we would need to find actual measurements as base-line performance.

Keywords: manifold learning; high dimensional CDF; non-parametric machine learning; binary searches; dictionary sort; graph algorithms

Introduction

Manifold methods [1] are an active area of research that have found application as dimensionality reduction techniques [2], representation learning [3] or just data exploration. Broadly speaking the type of algorithms fall in a number of categories, namely probability manifolds [4], Neural Network (NN) manifolds [5], geometric manifold [6,7] and graph manifolds [8]. However the sheer number of possible manifolds that exist in nature and the possible ways to detect them is so innumerable that the general consensus on what algorithms is expected to work best comes from prior understanding of the physical system that created the manifold. Because of that, the algorithms in this paper are mostly based on the following observation: The physical laws that underline most Telecom data mandate a strictly increasing manifold.

For example, increasing only the SNR for even a single users and keeping all other variables the same is guaranteed to worsen the cell throughput. Similarly if the type of service even for a single user needs more guaranteed bit rate this is also guaranteed to worsen the cell throughput. Furthermore if we increase any combination of these variables then the cell throughput will decrease.

This means the variables and the cell throughput lay on a strictly increasing manifold. Let us call that manifold a CDF manifold because the CDF is always a strictly increasing manifold and there is a relation between computing the high dimensional CDF and computing the corresponding value of the telecom data that lays on such a manifold.

If we try to apply any of the standard manifold method, for example NN then we'll struggle to regularize it while the training costs for large data sets will be high. On the other hand it is practically impossible to force the NN to adhere to the strictly increasing property of the data. And because of it an attentive customer would ask why the NN is inferring that if the activity on a cell increases with the

 doi:10.20944/preprints202603.0819.v1

rest of the KPIs remain the same the cell throughput will improve. The simple reality of the situation is that there might be a better non-parametric method to model the data and to this end we propose the CDF manifold algorithm.

We call any telco data that has the strictly increasing property as having the CDF manifold property. Such data admits the specialised set of constructions we define in Sec. 1, where by Theorem 1 the manifold is constructed in $O(n^{3/2})$ steps in the worst case and in $O(n.ln[n])$ steps in the best case. The manifold allows two specialised additional constructions that build on previous work in [9] which enable binary searches that cost $O(ln[n])$ in the best case and $O(n^{1/2}ln[n])$ in the worst case. Furthermore, the manifold we propose in this paper has the important property that it also allows us to compute the CDF once the manifold is constructed as described in Sec. 2 which give the manifold its name. Last but not least we perform a comparative simulation between a NN and the CDF manifold in Sec. 3. Importantly the proposed manifold is an explanatory algorithm that can always pinpoint the data that produced its prediction while the NN in comparison is not.

1. Defining the Manifold, Properties and Sampling

To formalize this pattern, assume we have data in d dimensions $[x_1, x_2, ..., x_k...x_d]$ and a target single dimension y of the data such that if any x_k increases independently then y stays the same or increases, meaning $y = CDF_{manifold}[x_1, x_2...x_k, ...x_d]$. To simplify the notations and proofs we can consider the manifold in 2D which is also the smallest whole number of dimensions where the manifold method is valid. Therefore for the remainder of the paper consider two dimensional data $[x_1, x_2]$ and $y = CDF_{manif}[x_1, x_2]$.

Next assume we have 2 data points $[a_1, a_2]$ mapping to $y_a = CDF_{manif}[a_1, a_2]$ and $[b_1, b_2]$ mapping to $y_b = CDF_{manif}[b_1, b_2]$ such that $[a_1 < b_1, a_2 < b_2]$ which also means that $y_a < y_b$. Also assume we have a new input data point $[g_1, g_2]$ for which we want to compute y_g. Because we know the data lays on a CDF manifold we know that if $a_1 < g_1 < b_1$ and $a_2 < g_2 < b_2$ then $y_a < y_g < y_b$. Therefore if $[a, b]$ is the tightest for g in the data then we can compute $y_g = (y_a + y_b)/2$. Observe that this estimate is a data driven explanatory computation since we can explain y_a, y_b which is not true for NN for example. This means that if the customer ask why we made a given inference we can point out the actual data measurements we used while with the NN we wont be able to give a definitive answer.

The draw back of this CDF estimate is that we must have as much data as possible to make sure that the interval is as tight as possible. As a result the look up algorithm must be extremely efficient computationally and using as little memory as possible. For instance for large data even $O(n^2)$ complexity becomes infeasible and so we'll set out to look for an algorithm that is more efficient.

Next, a point b is considered Above Right (AR) of point a if $a_1 < b_1$ and $a_2 < b_2$. Similarly, a point b is considered Below Left (BL) of point a if $a_1 > b_1$ and $a_2 > b_2$. Examples of this are given in Figure 1. To further simplify the algorithms observe that computing the below left is equivalent to computing above right but we just need to multiply each data dimension x_k other than the target dimension y by (-1). That is why for the remainder of the paper we only consider the algorithms in the context of computing the upper limit (UL) in the AR.

 doi:10.20944/preprints202603.0819.v1

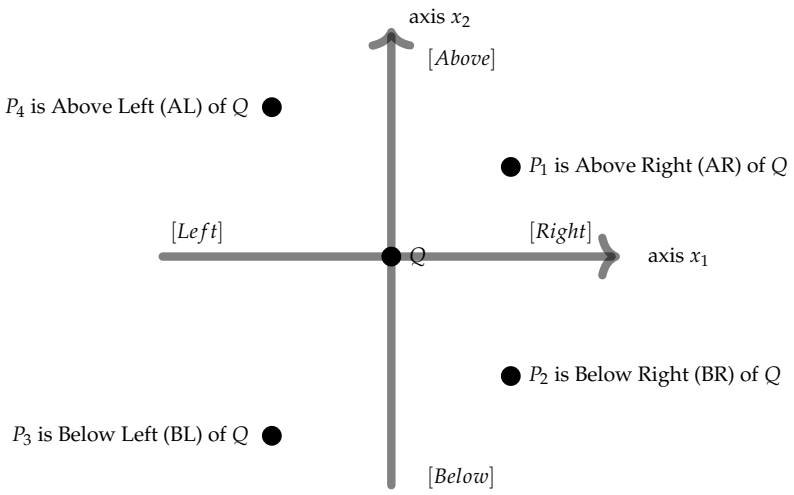

Figure 1. Defining above right and below left of a point.

Next in Figure 2 on the right we show that the AR preserves the upper limit. If a target point T is AR of sample point P then the upper limit of T is also AR of P.

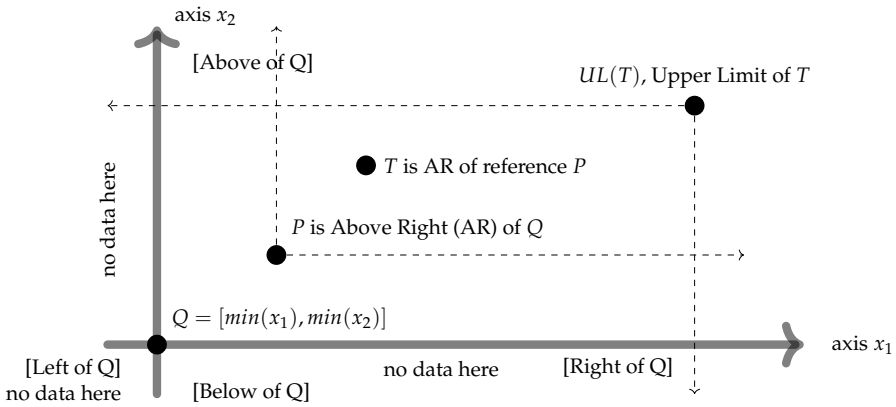

Figure 2. If target point T is AR of sample point P then Upper Limit of T is also AR of P.

Using the definitions and results of Figure 1 we can see that if we've found of a point R that is AR of target point T then no point that is AR of R like S can ever be a better upper limit to T as shown in Figure 3. In other words the BL and AR of a reference point R partitions the data space such that no point BL of R can have an upper limit in the AR of R.

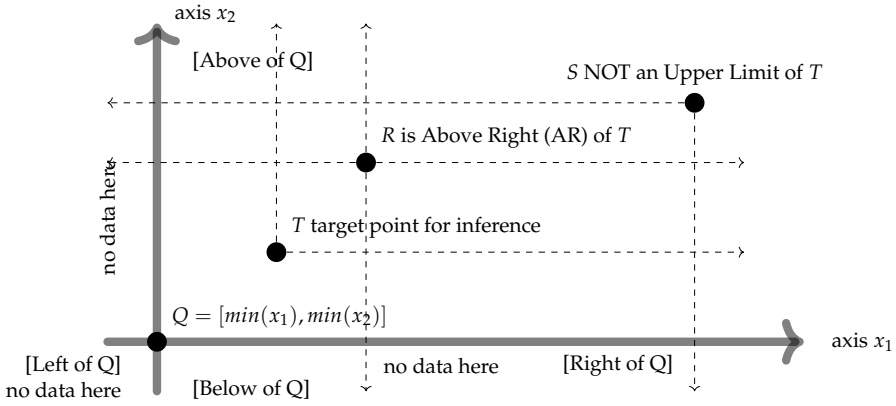

Figure 3. If a reference point R is AR of a target point T then the Upper Limit of T is not any point AR of R. In this example S is not an upper limit of T because R is a better upper limit.

 doi:10.20944/preprints202603.0819.v1

Next consider Figure 4. We define the points F_1, F_2, F_3 to form on a long front meaning no F_i is AR of F_j for all pairs $[i, j]$. Assume we start the binary search at F_1 and T is NOT AR of F_1. Which would be the next check point? In the worst case we might need to check F_2 then F_3 and in this case we'll do a lineae search to F_3 at which point we've found the target point. This proves by example that the AR and the BL functions do not eliminate with a single check a portion of the search data in general which hinders its used for binary searches.

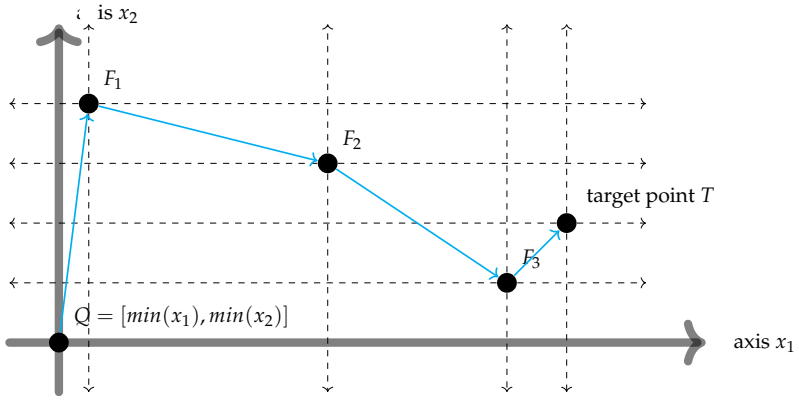

Figure 4. The AR and BL functions do not eliminate a portion of the search set in general which hinders its use for binary searches.

Fortunately there are cases where the AR and BL do reduce the search space with a single check as depicted in Figure 5. The first structure that support binary searches is given in Figure 5 where all points L are such that for any pair $[i, j]$ either $L_i \in AR(L_j)$ or $L_i \in BL(L_j)$. We call such points a long lines and we can model their relationship using the Index Invariant Graph (IIG) defined in [9] and [10]. The incidence matrix I_L defined by the AR operations is clearly a triangular matrix and as proven in [9] the dual hypergraph defined by that matrix is isomorphic to the original graph. In this context the dual isomorphism means that the AR operations uniquely describe the location of each data point. As an example consider in Figure 5 how we'd look for the target T among the L points. First we could check if T is AR of L_2. As a result we know that T is not AR of any point AR of L_2 which is L_3 in this case. This means we've eliminated halve of the data and we can consider the subset that is BL of L_2 which would be L_1.

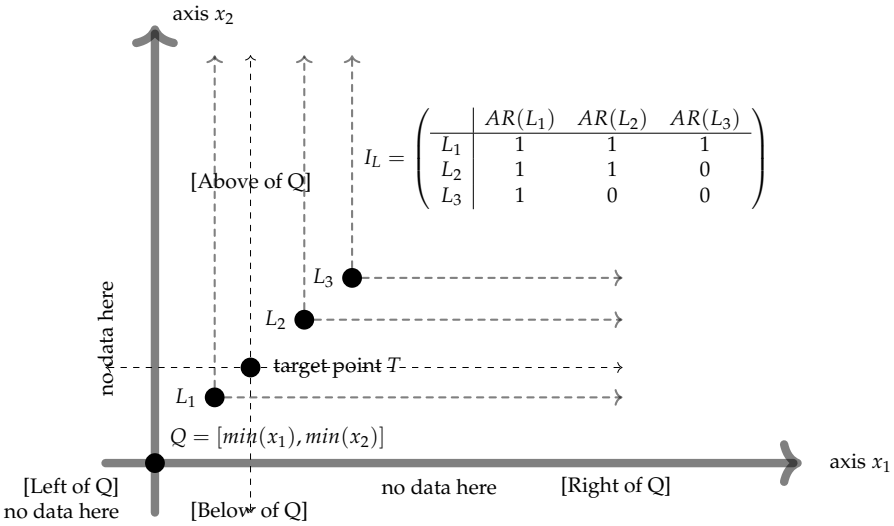

Figure 5. The structure where all data points are such that $\forall i \neq 1, \exists j \Rightarrow L_i \in AR(L_j)$ that we called a long line can be expressed as an Index Invariant Graph. Due to the dual isomorphism of the hypergraph the long lines allows binary search.

 doi:10.20944/preprints202603.0819.v1

Next in Figure 6 we apply a similar logic to a different structure. Let us call a long front a structure of points such that all pairs $[i,j]$ are such that $F_i \notin AR(F_j), F_i \notin BL(F_j)$. Set x_1 as the binary search dimension and x_2 as any other data dimension . Then define the construction of points C such that C_i is a translation of F_i among the data dimension x_2 to the minimum value among all data while the binary search dimension x_1 is kept or more explicitly $C_i = [F_i(x_1), \{min_i F_i(x_2)]\}]$. Observe that the incidence matrix I_F is all zeros, while the incidence matrix I_C is again modelled with the IIG. As a result we can treat the C points as L. That being said the F points still cannot be treated as L points. Instead a binary search on the C points will return just the closes upper value $best(x_1)$ along the binary search dimension x_1. Once we have the best upper limit $best(x_1)$ across x_1 we must run another binary search among a different data dimension for example x_3 starting at $best(x_1)$. We repeat the searches until there are no more dimensions or no more points left to search.

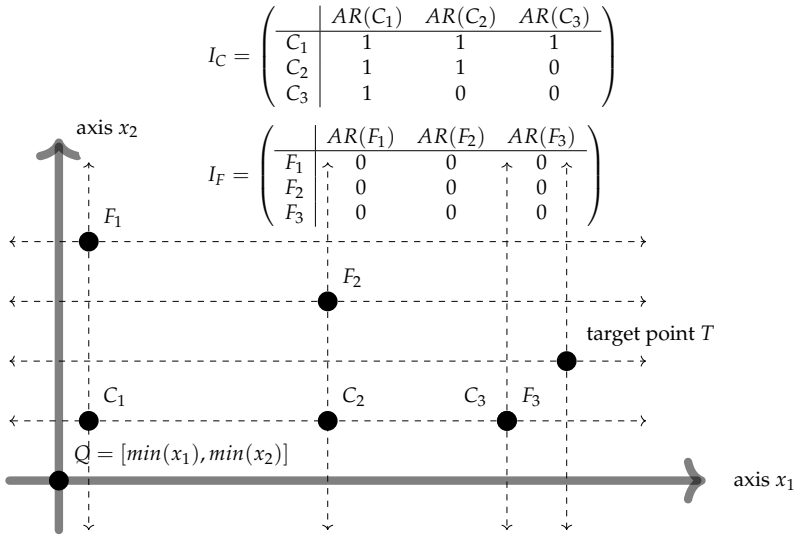

Figure 6. The structure where all data points are such that $\forall[i,j], L_i \notin AR(L_j), L_i \notin BL(L_j)$ that we called a long front cannot be expressed as an Index Invariant Graph. However with the additional construction of the C points we can model the front with the graph Due to the dual isomorphism of the hypergraph the modified long front allows binary search.

The next question we consider is how do we detect the long fronts and lines? Let $data$ is a matrix with n rows and d dimensions. Using Matlab/Octave notation $data(:,k)$ is a single data dimension and $data(i,:)$ would be a single data instance as a row vector. In this context a regular sort would then be applied to $data(:,k)$ while a dictionary sort would be applied to $data$. A dictionary sort of this type is implemented in [11]. Therefore a dictionary sort would order the data in such a way that if $data(i,:)$ appears before $data(j,:)$ then $data(i,k) \leq data(j,k)$. As a result we can scan the data from i to j and find the next point after i that is part of a long line L after i but before j. Or more generally if we do a linear scan we are guaranteed to find the longest line that stars with the first row of the data. Importantly the same linear scan will also give us the longest front that starts with the first row of the dictionary.

In short to detect a single line or front we just start at the first row in the sorted dictionary $data(1,:)$ and scan the data linearly until the end $i = 1:n$. We maintain two separate queues for the front q_F or line q_L where initially $q_F = data(1,:)$ and $q_L = data(1,:)$. Once we encounter a point i that is in a line with the last point in the line queue we add it to the queue $q_L = [q_L; data(i,:)]$ and mark it as the new end of the line. Similarly if point i is in a front with the last point in the front queue we add it to the queue $q_F = [q_F; data(i,:)]$ and mark it as the new end of the front. By the end of the scan either queues will have a long front or a line meaning a single IIG detection takes $O(n)$ iterations.

There is a way to speed up the expensive linear scan $O(n)$ of a single IIG detection by observing another property of the dictionary sort. This property is best explained with a numeric example and

 doi:10.20944/preprints202603.0819.v1

so let us have the following dictionary sorted data points $d_1 = [1,1], d_2 = [1,2], d_3 = [2,1], d_4 = [2,2], d_5 = [3,1], d_6 = [3.2]$. In this order $d_1 \to d_2$ form a line and between $d_3 \to d_3$ we have an inversion to a front. Similarly $d_3 \to d_4$ are a line and $d_4 \to d_5$ is an inversion to a front. Therefore if we start the detection at d_1 we'd expect the longest line to end at the first inversion while the front points would be likely within all the inversion. Of course this approach is just a heuristics and in the worst case we'd still need to perform a linear scan.

The next question we consider is what is the shortest line or front that exists in any data set? For starters any data set containing at least 2 points will have the 2 points in a long line of length 2 or a long front of length 2. Let us denote this with $ANY[2p] = OR[2L, 2F]$. If the data has 3 points we'd express this as $ANY[3p] = ANY[2p] + ANY[1p] = OR[2L, 2F] + ANY[1p]$. For these functions we can prove that:

Theorem 1.

$$ANY[n^2 p] = OR\Big[nL, nF\Big] + ANY\Big[(n^2 - n)p\Big] \tag{1}$$

meaning that in a data set of size n the shortest long front or long line contains $n^{1/2}$ points.

Proof of Theorem 1 . We will prove Theorem 1 by induction. To the start of the induction we already stated that Eq. 1 is true for any 2 data point. Therefore assume it is true up to some number n. For the induction step we need to prove it is true for $n+1$ of more explicitly that:

$$ANY\Big[(n+1)^2 p\Big] = OR\Big[(n+1)L, (n+1)F\Big] + ANY\Big[((n+1)^2 - (n+1))p\Big] = \tag{2}$$

Observe that $ANY\Big[(n+1)^2 p\Big] = ANY\Big[n^2 p\Big] + ANY\Big[((n+1)^2 - n)p\Big]$ and so we can substitute Eq. 1 to get:

$$ANY\Big[(n+1)^2 p\Big] = OR\Big[nL, nF\Big] + ANY\Big[((n+1)^2 - n)p\Big] = \tag{3}$$

In short Eq. 3 tells us that the bigger data set will contain at least a front or a line of size n and $((n+1)^2 - n)$ that are in some configuration around them. This is depicted in Figure 7 and Figure 8.

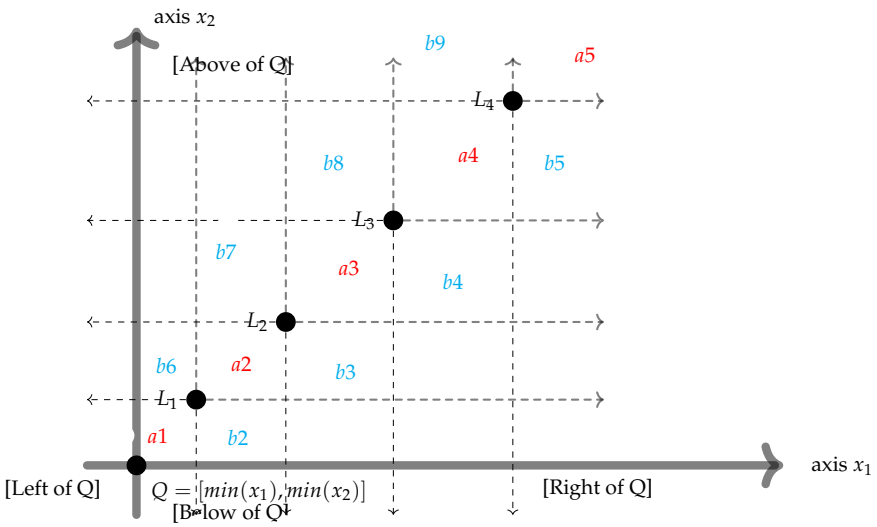

Figure 7. Areas a indicate locations where we cannot add any points without going over the long line limit. Areas in b are positions that don't increase the long line limit but do increase the long front count for some points.

 doi:10.20944/preprints202603.0819.v1

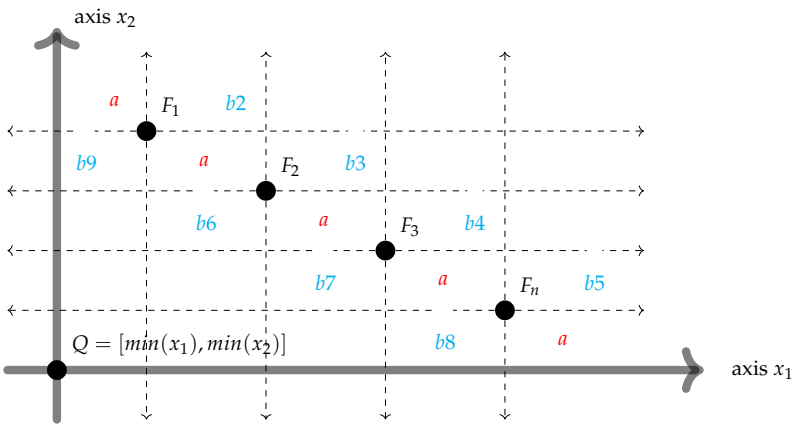

Figure 8. Areas a indicate locations where we cannot add any points without going over the long front limit. Areas in b are positions that don't increase the long front limit but do increase the long line count for some points.

Figure 7 and Figure 8 the red areas a indicate areas where we cannot fit any more points of we'll end up with $OR\Big[(n+1)L,(n+1)F\Big]$ meaning all the $((n+1)^2-n)$ points must be somewhere if the blue areas marked with b. Clearly if we fit all $((n+1)^2-n)$ within the same area we'll produce at least one $OR\Big[(n+1)L,(n+1)F\Big]$. Therefore at best we can try splitting the point evenly across the n line or front points. As a result we'll end up with at least:

$$\frac{((n+1)^2-n)}{n} = \frac{n^2+2n+1-n}{n} = n+1+\frac{1}{n} \tag{4}$$

many points which will produce at least one $OR\Big[(n+1)L,(n+1)F\Big]$ by the pigeonhole principle. As a result we've proven Eq. 1.

□

2. Computing the CDF

Recalling Figure 3 on the left we can also infer that if S was an UL to T then there exist no point R that is both AR of T and BL of S. In other words the $UL(T)$ is always such that there is no point that is $AR(T)$ and $BL(UL(T))$ that will contribute to the $CDF(T)$.

Therefore consider the graph $G = \{V, E\}$, where the vertex set $V = \{d_1, d_2 ... d_k .. d_n\}$ is the data and the edge set $E = \{(d_1, UL(d_1)), ...(d_k, UL(d_k))...(d_n, UL(d_n))\}$ are all the pairs d_k to its upper limit $UL(d_k)$. Next select a data point d_k that has a number of other point as in upper limit meaning $d_k = UL(d_w), d_k = UL(d_u)$... etcetera. These points d_w, d_u... are not AR of one another because if they were then they would not be upper limits to d_k too. As a result $CDF(d_k) = CDF(d_w) + CDF(d_u)$... meaning we need to just sum up the CDF values of the edges that contain d_k to compute $CDF(d_k)$ in Eq. 5:

$$CDF(d_k) = \sum_{(d_k, d_w) \in E} CDF(d_w) \tag{5}$$

Applying Eq. 5 to all n data points should be straight forward using a modified breadth-first search [12].

3. Numerical Comparison

Figure 9 shows an example where the AR and BL were computed for a single point in 3D since the manifold only has meaning where we have at least 2 input and 1 output dimension.

 doi:10.20944/preprints202603.0819.v1

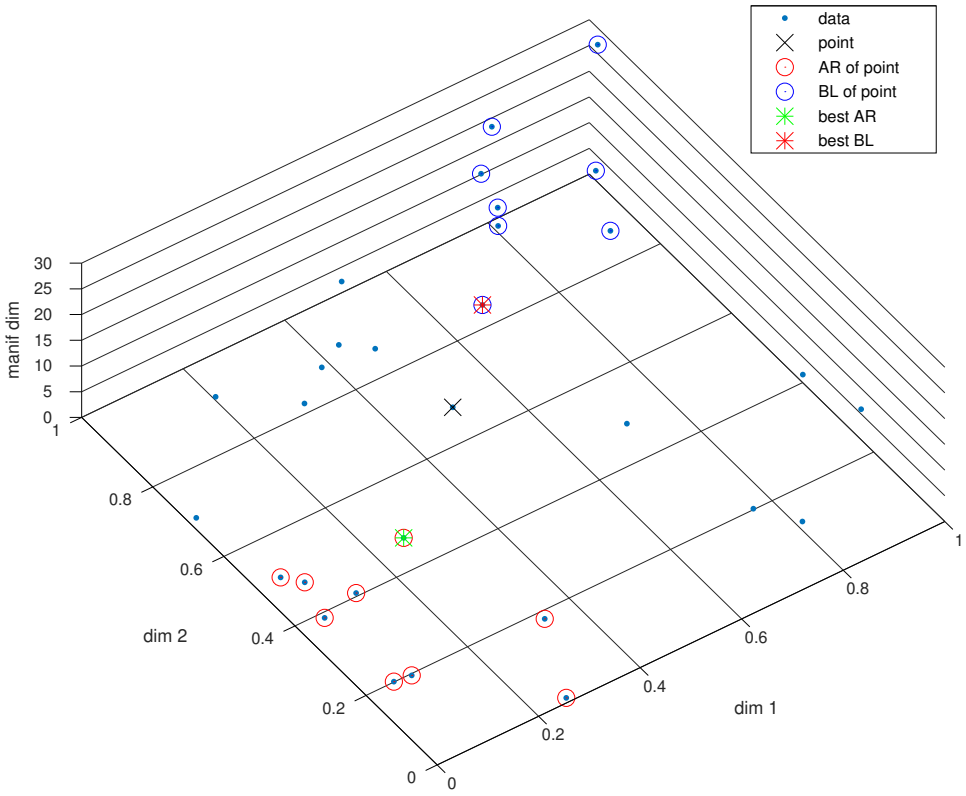

Figure 9. Example AR and BL for a point.

In Figure 9 observe that the AR and BL set might be a huge part of the whole set meaning computing the optimal limits is difficult Figure 10 shows a side view the same example as Figure 9. In Figure 10 observe that the limits might not be very tight, but the result would still be reasonable due to the manifold property.

For the remaining evaluations in this section consider a common telco problem where the base station has gathered 3 KPIs namely the UE activity, mean CQI and mean UE throughput on hourly bases. For a given UE activity and CQI we'd like to be compute the mean UE throughput. Observe that increasing the activity separately from the CQI will decrease the throughput. Similarly increasing the CQI separately from the activity will increase the throughput. Increasing both the CQI and decreasing the activity will increase the throughput. Therefore let the first dimension $x_1 = CQI$ while the second dimension is the minus of the activity $x_2 = (-1) * activity$. The target variable $y = throughput$ is the mean throughput. Therefore x_1, x_2, y have the CDF manifold.

A straightforward method to compute the mapping would be to directly apply a NN. For a comparison we'll run the CDF manifold method proposed in this paper. To start the comparison observe that the computational cost of a single NN training even with Strassen's multiplication is $O(n^{2.7})$ while the CDF manifold costs in the worst case $O(n^{3/2})$ meaning the manifold runs more than $O(n)$ times faster. In fact depending on the data the manifold might cost only $O(n.ln[n])$ which is an improvement of at least $O(n^2)$. Furthermore the manifold is an explanatory model where we know exactly which 2 data points produced the prediction, while the NN acts as black box that does not have a clear reason for any given prediction.

 doi:10.20944/preprints202603.0819.v1

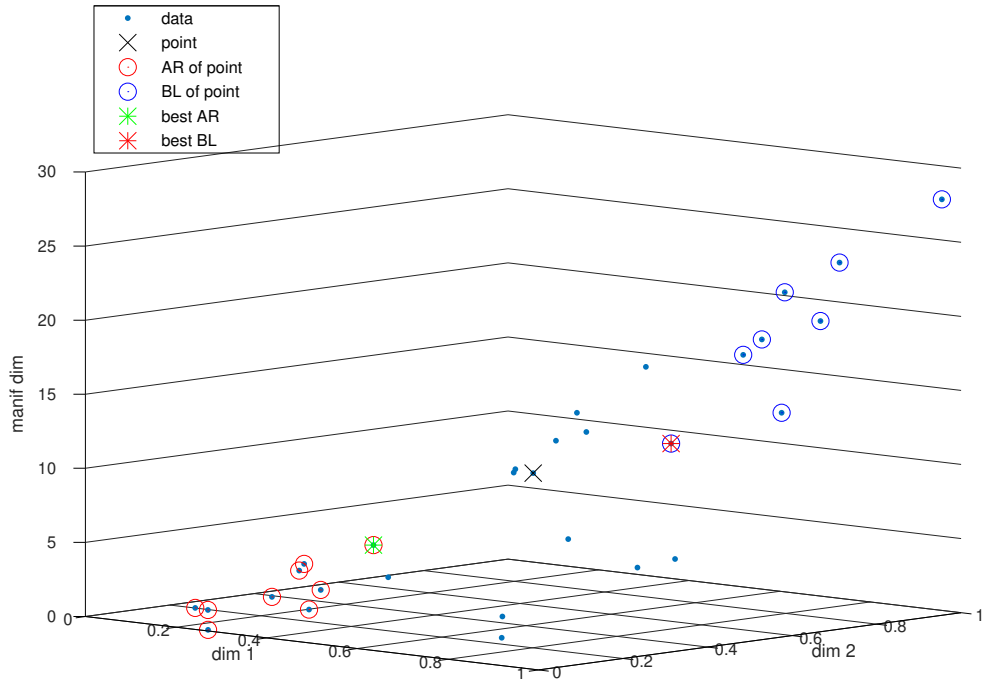

Figure 10. Side view of Figure 9.

Next, the input data x_1 and x_2 is sampled from a standard normal distribution using $rand(*)$ in Octave. The true mapping between the data dimensions x_1 and x_2 is assumed a sum of sigmoid functions $\sigma[*]$ or more specifically $y = \sigma[10(x_1+1)] + \sigma[15(x_1+2)] + \sigma[5(x_1-1)] + \sigma[2(x_1+1)] + \sigma[1(x_1+3)] + \sigma[3(x_1-3)] + \sigma[4(x_1-2)] + \sigma[(x_2-1)] + \sigma[30(x_2-1)] + \sigma[30(x_2-2)]$. Note that because a sum of strictly increasing functions is strictly increasing then this mapping has the CDF manifold property. This elaborate mapping from x to y was chosen to make sure we observe a lot of errors made by both models.

The NN has a single layer with 16 neurons and was trained for 400 iterations using Octave's $fminunc(*)$ function applied to the forward-backward propagation algorithm. To compare the NN to the manifold we compute the Root Mean Square Error (RMSE) and a separate swap metric.

The swap metric counts the number of time the prediction does not adhere to the strictly increasing property. Meaning, for a data prediction point t we find the indexes of all training points $I_{ar}[x \in AR(t)]$ that are AR of t, then we count how many of them had y values that were not AR of the prediction y_p or more explicitly $y(I_{ar}) \notin AR(y_p)$.

We start the simulation with a sample of 40 data points, then compute the error measures and then add 40 new sampled data points to the original set. We repeat sample and addition until we get to a sum total of 800 points. The reason behind this is to simulate how KPIs get added over time as the base station obtains more and more real-life measurement. Additionally, the data is split into training and validation sets using a 50/50 ratio. The results are given in Figure 11 where the Y-axis is on a logarithmic scale because the measures have different scales and the X-axis is the number of input data for each estimation.

 doi:10.20944/preprints202603.0819.v1

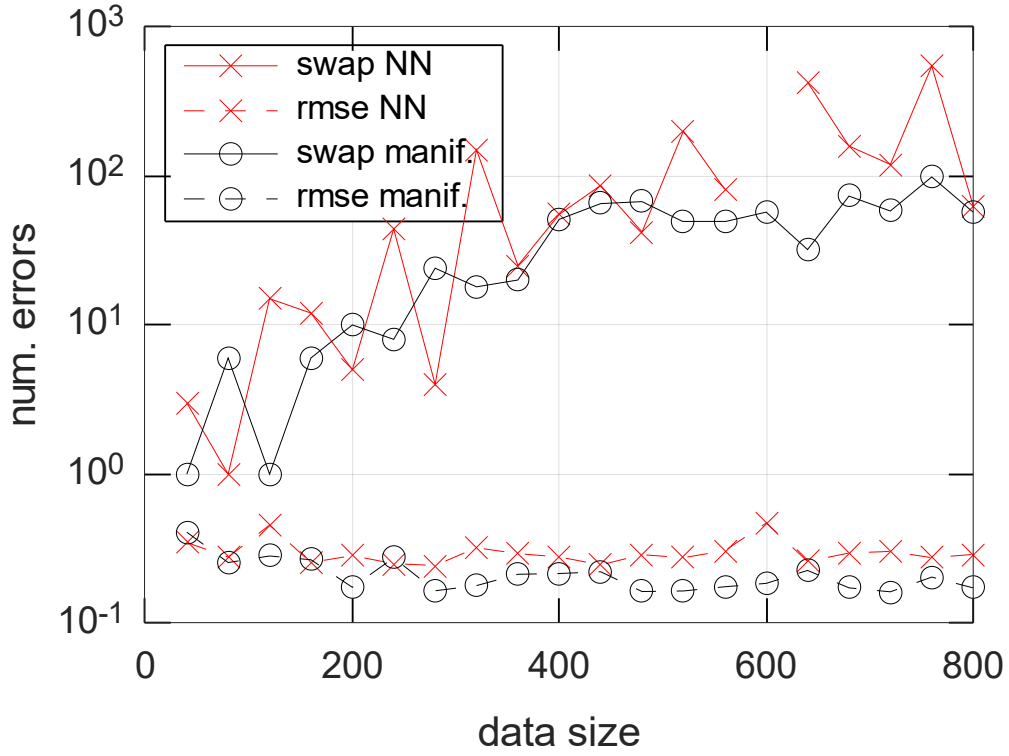

Figure 11. The change in RMSE and number of errors made by the NN versus the manifold as a function of the size of the data. The data size increases by an in incremental addition of 40 new points instead of sampling an entirely new data set, which simulates a monthly addition of new base station measurements.

In Figure 11 we see that the RMSE gradually decreases for both models as we get more and more data. The RMSE of the NN flattens after we reach around 200 sample while the manifold experiences improved performance with each new data point.

Both the NN and the manifold can experience swaps in the strictly increasing data property, but the manifold swaps are more reliable and in many cases at least 1 order of magnitude better than the NN.

Interestingly for data size 600 the NN had 0 swaps which does not guarantee it will not swap an unseen input at all, because its difficult to make sure the continuous NN mapping if strictly increasing everywhere. On the other side with a large enough data set the manifold will make less and less swaps while the RMSE will keep decreasing.

Therefore as future work it would be interesting to see if there is an ensemble model that can take advantage of both the NN and the manifold.

4. Conclusions

In this paper we introduced the CDF manifold that on average can be constructed in $O(n.ln(n))$ steps with the worst case costing $O(n^{3/2})$. Searching with a single new instance is expected to costs $O(ln(n))$ steps and $O(n^{1/2}ln(n))$ in the worst case.

The manifold also relies on the dictionary algorithms of this [11] package. The same manifold can also be used to compute the CDF of either a data points or a new data point as described in Sec. 2. Both the construction and the search algorithms can be parallelised.

The algorithm is expected to perform best with telecom data where the KPIs have physical system meaning that keep it bound within certain limits which increases the likelihood of containing singular long fronts and long lines. In addition to that telco data is highly likely to be strictly increasing which satisfy the prerequisites of the algorithm.

 doi:10.20944/preprints202603.0819.v1

We also did a simulation study to compare the performance of a NN and the CDF manifold. The result showed that for large data sets the computational complexity, the achievable accuracy and the explanatory characteristic of the search algorithms point in favour of the manifold.

Future work consist of considering various reference point approaches because we saw how the inversion points heuristic can help speed up construction. Another future work would be considering hybrid models between the CDF manifold and a neural network for example.

Author Contributions: Conceptualization, V.V. and G.I.; methodology, V.V.; validation, V.V.; formal analysis, V.V.; investigation, V.V.; writing—original draft preparation, V.V.; writing—review and editing, G.I.; visualization, V.V. and G.I.; supervision, G.I.; project administration, G.I.; funding acquisition, G.I. All authors have read and agreed to the published version of the manuscript.

Funding: This study is financed by the European Union—NextGenerationEU, through the National Recovery and Resilience Plan of the Republic of Bulgaria, project No. BG-RRP-2.004-0005.

Data Availability Statement: Data is contained within the article; further inquiries can be directed to the corresponding author.

Acknowledgments: Special thanks to Cyril Murphy who greatly enabled and encouraged the investigation of the patterns investigated in this paper.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Meilă, M.; Zhang, H. Manifold learning: What, how, and why. *Annual Review of Statistics and Its Application* **2024**, *11*, 393–417.
2. Mordohai, P.; Medioni, G. Dimensionality estimation, manifold learning and function approximation using tensor voting. *The Journal of Machine Learning Research* **2010**, *11*, 411–450.
3. Hamilton, W.L.; Ying, R.; Leskovec, J. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584* **2017**.
4. Thakare, R.; Akugri, K.M. Manifold Decoders: A Framework for Generative Modeling from Nonlinear Embeddings. *arXiv preprint arXiv:2510.13622* **2025**.
5. Guo, J.; Wen, C.K.; Jin, S.; Li, G.Y. Overview of deep learning-based CSI feedback in massive MIMO systems. *IEEE Transactions on Communications* **2022**, *70*, 8017–8045.
6. Fei, Y.; Liu, Y.; Jia, C.; Li, Z.; Wei, X.; Chen, M. A survey of geometric optimization for deep learning: from Euclidean space to Riemannian manifold. *ACM Computing Surveys* **2025**, *57*, 1–37.
7. Lang, R.J. A computational algorithm for origami design. In Proceedings of the Proceedings of the twelfth annual symposium on Computational geometry, 1996, pp. 98–105.
8. Chen, F.; Cheung, G.; Zhang, X. Manifold graph signal restoration using gradient graph Laplacian regularizer. *IEEE Transactions on Signal Processing* **2024**, *72*, 744–761.
9. Vasilev, V. Chromatic Polynomial Heuristics for Connectivity Prediction in Wireless Sensor Networks. In Proceedings of the ICEST 2016, Ohrid, Macedonia, 28-30 June 2016.
10. Vasilev, V.; Leguay, J.; Paris, S.; Maggi, L.; Debbah, M. Predicting QoE factors with machine learning. In Proceedings of the 2018 IEEE International Conference on Communications (ICC). IEEE, 2018, pp. 1–6.
11. Vasilev, V. Folded sheet of paper. Bitbucket, 2024. Self-published.
12. T. Cormen, C. Leiserson, R.R. *Introduction To Algorithms*; The MIT Press, 2000.