A Report On

DTN Routing as a Machine Learning Classification Problem

Prepared for

Dr Virendra Singh Shekhawat
Instructor in Charge

BITS Pilani

Prepared by

Mayank Jain

2019A7PS0141P

Birla Institute of Technology and Science Pilani, Rajasthan - 333031
May 2021

Acknowledgements

I would like to thank Prof. Sudhir Kumar Barai, Director-Birla Institute of Technology and Science, Pilani campus for providing me the opportunity to work on this report.

I consider myself very fortunate to work under Dr Virendra Singh Shekhawat, Instructor-in-Charge of my project for providing me this great opportunity to work on a topic gaining more and more importance as we speak. I would also like to thank him for the immense guidance and support he provided, while I was preparing this report.

Table of Contents

1. Introduction

1. What are DTNs?
    - Delay tolerant networks (DTNs) are a field of research focused on networks in which link connectivity may be frequently disrupted.
2. Where are DTNs used?
    - Mars exploration rover and CubeSats or drone swarms that perform image collection of a planet's surface
    - Both consist of many smaller nodes that transfer the data to bigger relay nodes.
    - The network topology continuously changes due to nodes moving out of range as well as loss of connectivity due to other factors such as environmental interference and other errors.
    - In both the cases, some nodes are expected to fail in routing due to connection disruptions, damage etc.
    - So, we can't rely on all nodes in the network to be known at any moment for routing.

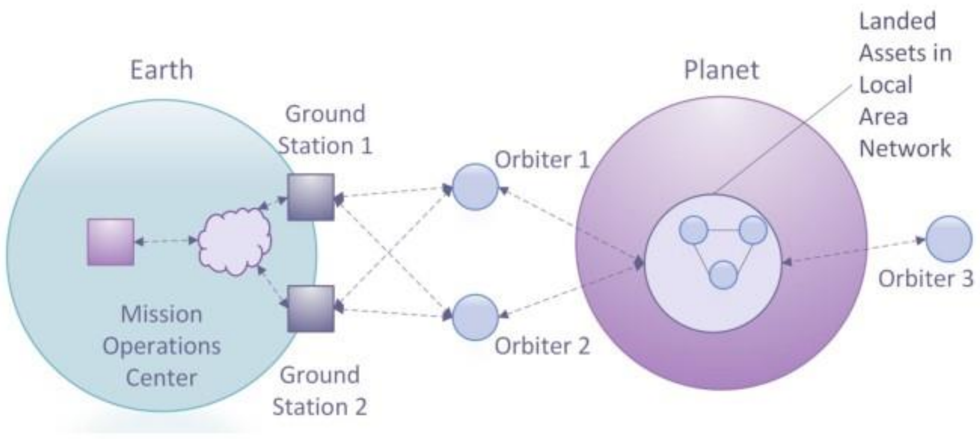


Fig. 1. Deep Space Scenario with Rovers and Orbiter Relay [4]

3. For a successful routing, we must address three issues

   a. While the relay nodes exhibit very predictable behavior, the smaller "worker" nodes may move with either random or predictable patterns as they perform autonomous tasks (image collection, spectrometry, radiometry etc.)

   b. Link asymmetry between forward and return links – leading to data being sent faster than the acknowledgments being received.

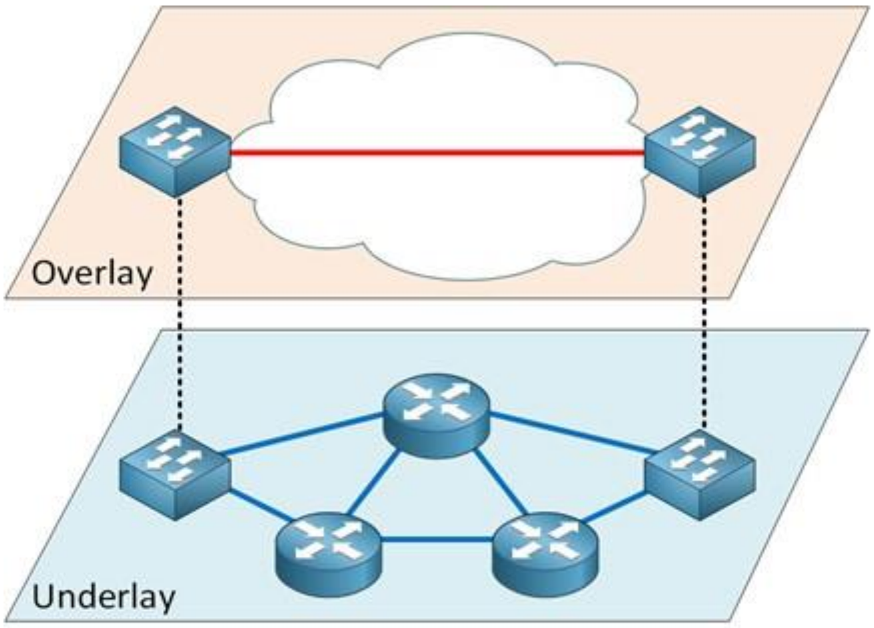
   c. Multiple protocol stacks being used in the path from source to destination creates a need for a common interface layer between the nodes.

1. DTN Architecture and Bundle Protocol

The above issues are addressed by Bundle Protocol and the DTN Architecture

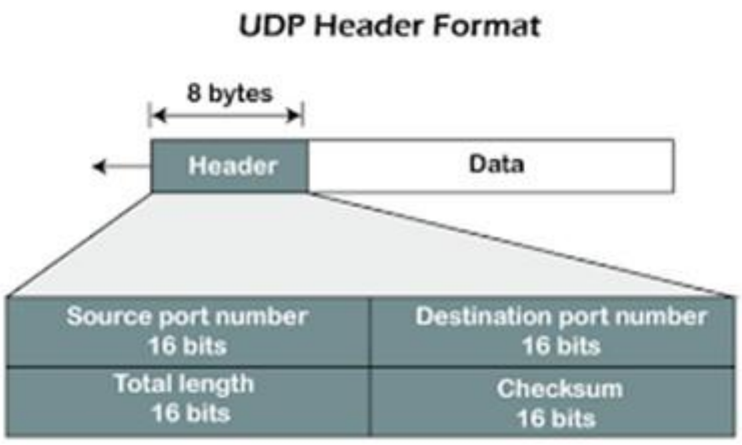
1. **Bundle Protocol –**

   a. It is an overlay protocol which means it creates a network layer between datalink, transport, and physical layers.
   b. Store & Forward based protocol – meaning the data is stored during disruptions and transmitted once the connectivity is restored.
   c. The overlay removes the lower-level details and simplifies it allowing nodes to use variety of suitable Internet protocols.



2. DTN architecture

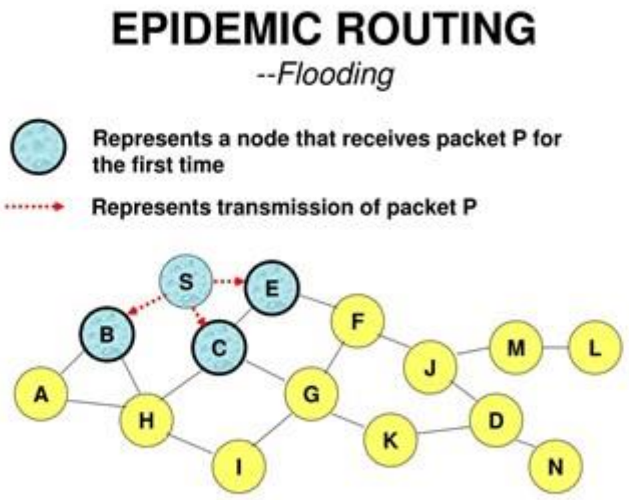
a. Allows for Internet Protocol Neighborhood Discovery (IPND) which is an opportunistic discovery mechanism.
b. IPND allows nodes to announce themselves to previously unknown nodes and exchange connectivity information as UDP Datagrams.



2. Related Works

1. Opportunistic Routing (OR) for DTNs

a. Most existing methods are based on epidemic routing.
b. Epidemic routing is flooding-based in nature, as nodes continuously replicate and transmit messages to newly discovered contacts that do not already possess a copy of the message.
a. It is very resource hungry as it makes no attempt to eliminate replications that would be unlikely to improve the delivery probability of messages.


EPIDEMIC ROUTING
--Flooding
Represents a node that receives packet P for the first time
Represents transmission of packet P
S
E
B
C
F
J
M
L
A
H
G
D
K
I
N

b. Most OR protocols focus on a way to determine the best nodes to copy a message to, meaning nodes which have the greatest chance of encountering the destination node.

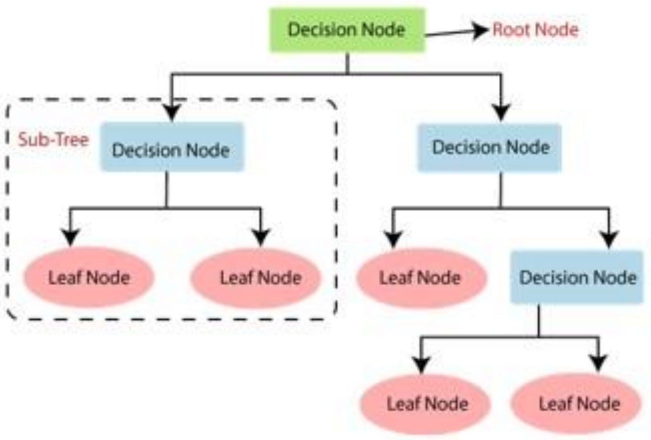
c. E.g., PRoPHET routing protocol is based on past encounters and takes probability into account to determine the best nodes.

2. Machine Learning Techniques applied to routing in DTNs

a. Decision tree-based classifiers are applied to make improved routing decisions for epidemic routing by classifying nodes using an attribute vector and a derived classification label.

b. Other methods use Bayesian classifiers with attributes such as network regions, a time-based index and message destination.

c. Both methods use stored network traffic history as samples to train their classifiers.



Decision Tree-based Classifier

2. Algorithm Development

We solve the routing problem as a Machine Learning classification task.

Why do we use Machine Learning?

1. It is adaptable to many conditions.
   - New nodes constantly entering/leaving the network due to different nodes following different operating patterns over different lengths of time.
2. It allows to detect patterns of disruption/traffic which may be difficult to detect normally.
3. Many ML algorithms can be used like –
   - Supervised Learning
   - Unsupervised Learning
   - Reinforcement Learning

Since the data is labelled, we can use supervised learning.

The model essentially determines the best nodes to forward the message to, thus reducing overhead too.

Here we classify the nodes which are in the optimum route to the destination node according to the trained model.

1. We assume that the behavior of each node follows a predictable periodicity which is influenced by various parameters like the set of neighboring nodes, duration of the contact period, data rate, buffer capacity etc.
2. We call this period an epoch and divide it into time slices.
3. For a message, we are classifying whether the message has been forwarded to the ith node where $i \in [0,\text{numNodes})$
4. So, this is a single multi-label classification problem split into multiple binary classification problems. This can lead to performance issues.
5. Also, it classifies all labels separately, without considering the interdependence between outputs.
6. Machine learning with Ensemble Classifier Chains (ECC) solves the performance issue of multilabel classification and takes interdependence into account.
7. We select five well-known classifiers (XGBoost, AdaBoost, Decision Tree, Gaussian Naïve Bayes, and K-Nearest Neighbors), to determine which would provide the best performance.

8. Our classifier takes following input features/attributes (X)⟶
   (tsIdx,fromHost,toHost,region_x,region_y,deliveryStatus)
9. And returns output labels (Y) ⟶ (d0-d9)

for each entry indicating if the message visited i'th node or not.

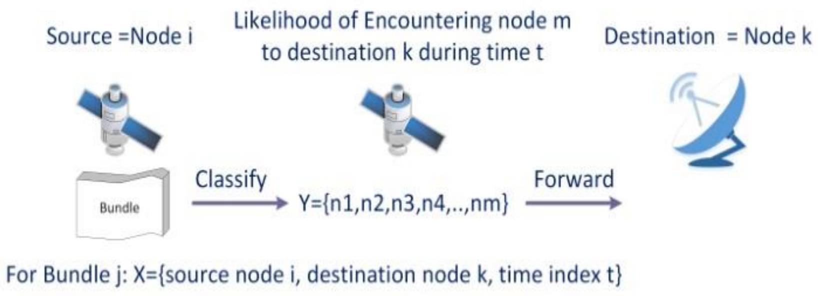

Source =Node i
Likelihood of Encountering node m to destination k during time t
Destination = Node k
Bundle
Classify
Y={n1,n2,n3,n4,..,nm}
Forward
For Bundle j: X={source node i, destination node k, time index t}

Fig. 4. Example Classifier Attribute Vector and Prediction

The features have the following meanings.

Feature	Meaning
tsIdx	Time index in the epoch
fromHost	Source node
toHost	Destination node
region_x	Region index for source node
region_y	Region index for destination node
deliveryStatus	Whether the message is delivered or not

3. Implementation

Emulation Tool Chain

- DTN-2 is used as a basis for the software development.
- ONE Simulator (The Opportunistic Network Environment) is used.
- Mobility traces from ZebraNet (based on data from zebra movements) were also used for providing more realistic movement patterns.
- Excel was used to organize the data in database tables for efficient access to the training data.

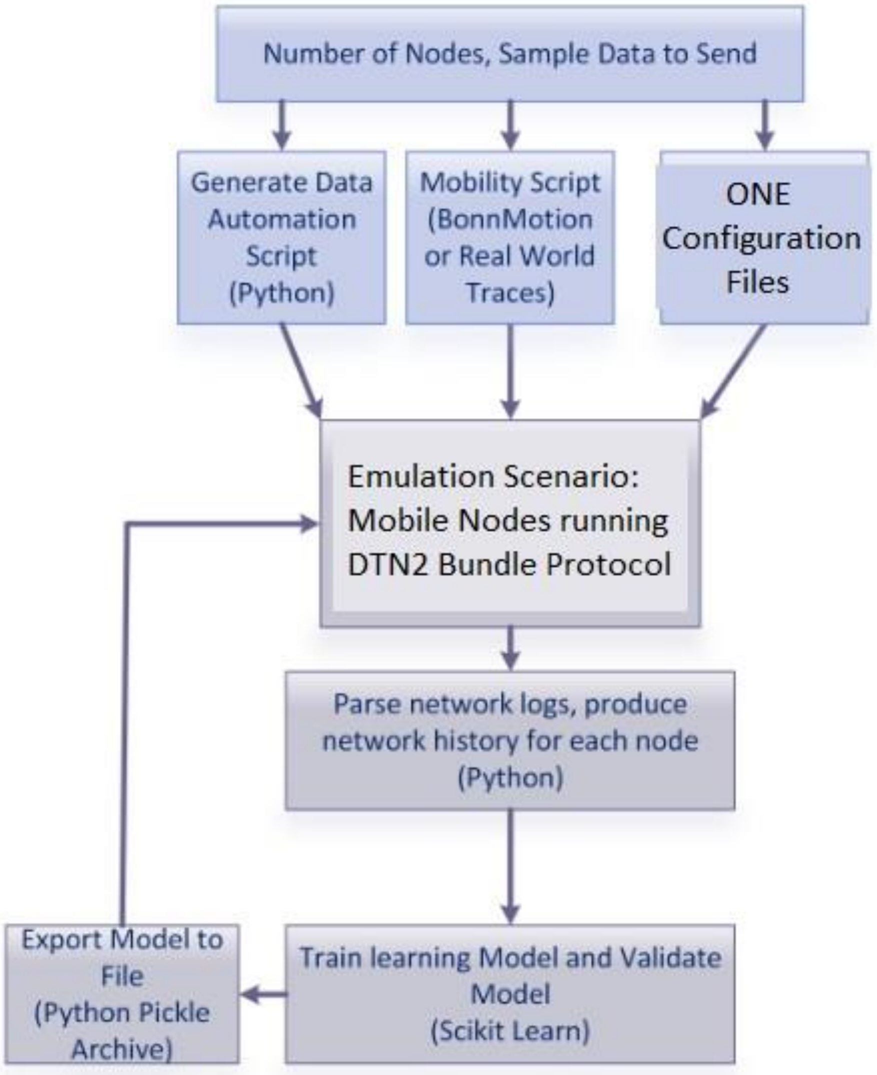

Number of Nodes, Sample Data to Send
Generate Data Automation Script (Python)
Mobility Script (BonnMotion or Real World Traces)
ONE Configuration Files
Emulation Scenario: Mobile Nodes running DTN2 Bundle Protocol
Parse network logs, produce network history for each node (Python)
Train learning Model and Validate Model (Scikit Learn)
Export Model to File (Python Pickle Archive)

Fig. 5. Emulation Tool Chain

Core Program Implementation Details

1. trace2one.py

The implementation works as follows -

a. First, we extract the locations from the UTM file and for each entry we calculate
    a. distance moved and the direction angle relative to the last location (by get_delta_xy method)
    b. the change in angle (by get_delta_direction method)

b. We then initialize the simulation time and the dimensions of the simulation area and a standard offset.

c. Now for each node, we generate the node locations from the direction and distance lists in step 1.

d. We also deal with the movements at the boundary (reflections).

e. One important detail is that ONE only accepts the entries separated by a constant value (the standard offset). So, after running the script we sort the node locations by time.

2. Preprocessing & Feature Building

1. Node Locations Dataset

Goal - To assign a region code to each node based on its locations throughout the simulation.

a. To determine the best number of clusters for k means clustering, we used K-Elbow visualizer and ran it from limits 2 to the total number of nodes.

b. Now with the optimum number of clusters, we created a new column with the region code for node - by taking the mode of all region codes associated with each node across all epochs.

2. Message Delivery Report Dataset

Goal - To generate columns with the columns for the input features and output labels –

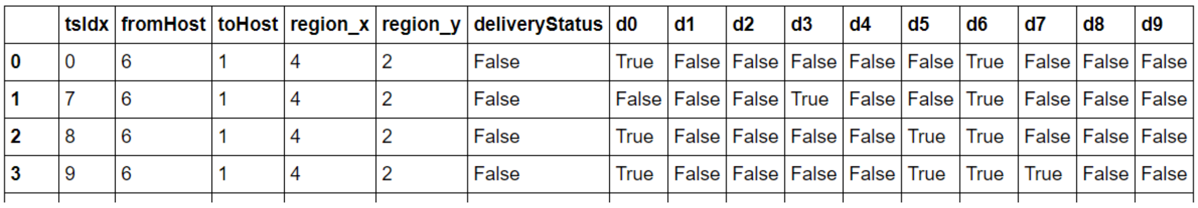
Our classifier takes input features (X)→

(tsIdx, fromHost, toHost, region_x, region_y, deliveryStatus)

And returns output labels → (d0-d9)

for each entry indicating if the message visited the i'th node or not.

a. After running the simulation, import the ONE reports AllNodesDeliveryReport for the message details that the message was forwarded to and LocationSnapshotReport for the region index code.
b. The size of epoch is decided arbitrarily which is then used to assign a timeslice index to all data entries.
c. Remove the irrelevant columns like time, hopcount, size etc.
d. All entires are grouped by 'Id,tsIdx,fromHost,toHost' to find all the nodes that the message was forwarded to in an epoch.
e. Unpack the path column entries to suitable format.
f. Create new column deliveryStatus indicating if the message got delivered or not.
g. Now, join the two databases to create the "regioncode" columns associated with the message source and delivery nodes.
h. Generate the columns for the output labels.
i. Finally, our dataset looks like –



	tsIdx	fromHost	toHost	region_x	region_y	deliveryStatus	d0	d1	d2	d3	d4	d5	d6	d7	d8	d9
0	0	6	1	4	2	False	True	False	False	False	False	False	True	False	False	False
1	7	6	1	4	2	False	False	False	False	True	False	False	True	False	False	False
2	8	6	1	4	2	False	True	False	False	False	False	True	True	False	False	False
3	9	6	1	4	2	False	True	False	False	False	False	True	True	True	False	False

j. Now, split the data into test and training sets and scale appropriately.

3. Training

A. Classifier Optimization –

1. **Decision Tree**, **Naïve Bayes**- No parameters to optimize.
2. **K Nearest Neighbors** - 'k' has to be optimized, which came out as 1.
3. **XGBoost**

   Now, the goal is to optimize the parameters for XGBoost classifier.

   a. We use tree booster since it always outperforms linear booster.
   b. First, we tried the GridSearch method to search for the optimum parameters, but it was less performant than the "hyperopt" library since

GridSearch only checks the values at the defined step size whereas hyperopt has functions which check values uniformly in the given range.

c. So finally, we use hyperopt library for tuning the parameters.
d. Since hyperopt does not allow for multi-label classifications natively, we find the optimum parameters for each label separately through binary classifications for each label.
e. The parameters were found by trial and error by first trying the range of the values where they generally fall in, and then adjusting according to the results on these.
f. Then we run the complete program within the range of all the optimum parameters combined to find the best parameters.
g. This process took many iterations since it was done manually.

4. **AdaBoost**

We use the hyperopt library as done above for XGBoost.

B. Evaluation -

The multi label classification techniques used are -

1. **OneVsRest** - Uses multiple independent classifiers assuming all labels as mutually exclusive.
2. **Chain Classifiers** - Takes into account the correlation between labels by stacking the classifiers.
3. **Ensemble Chain Classifiers** - Removes the dependence on the order of the chains by randomizing it.
4. **Label Powerset**

Here one important detail is that after prediction the model gives continuous values from 0 to 1(if ith node is visited or not).

But visited should be a Boolean value, so for all values with greater than 0.5, the message is assumed forwarded to the node and not forwarded for less than 0.5.

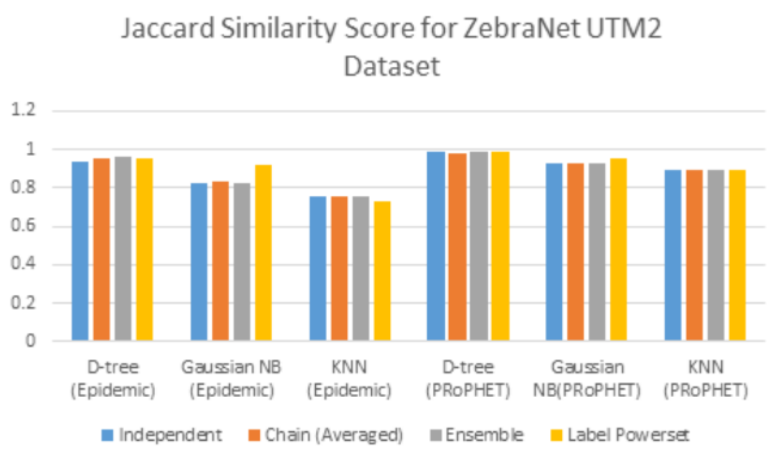
For all the above four techniques and five classifiers, we calculated the following metrics -

1. **Jaccard Similarity Score**
2. **f1 Score**
3. **Hamming Loss**
4. **Zero-One Loss**

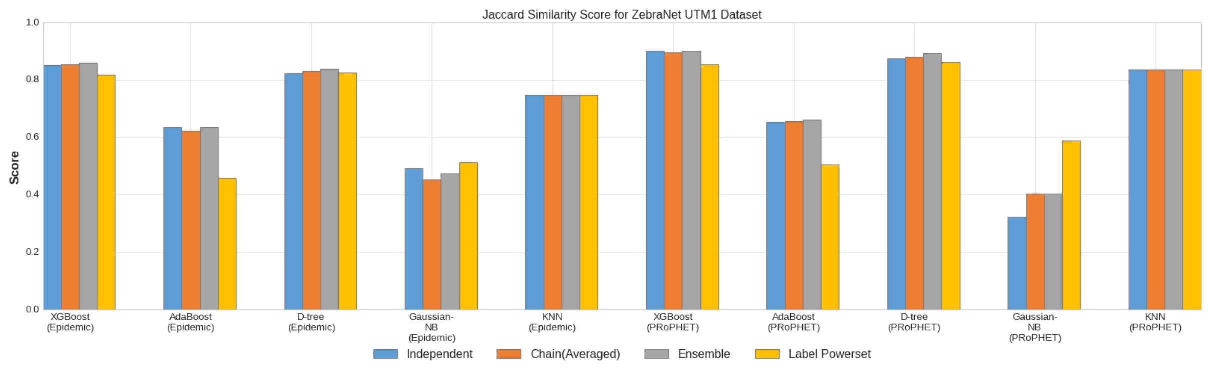
4. Metric wise Results

1. Jaccard similarity score

- Also called multi-label accuracy, it is the size of the intersection of two label sets (the predictions and true labels) divided by the size of the union of the two label sets.
- Higher score→ More accurate classification



Original Implementation

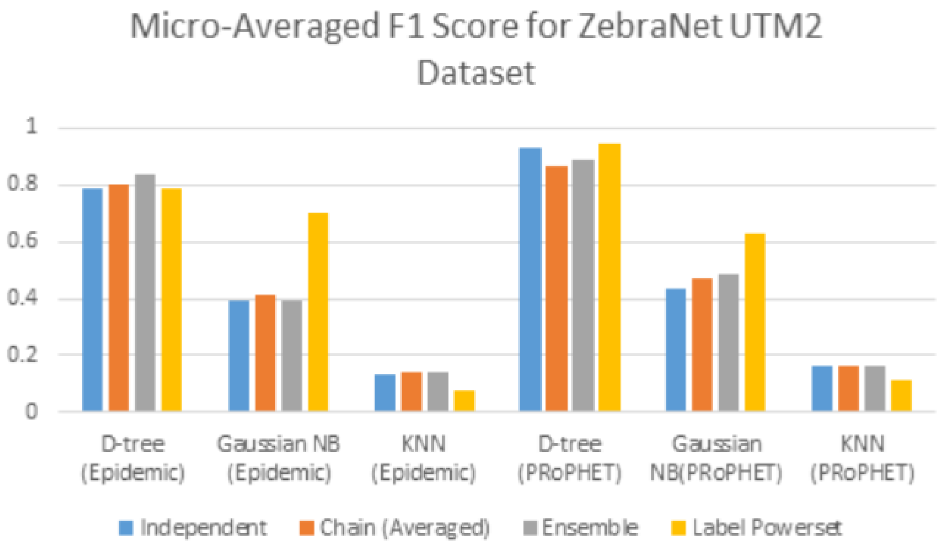


Our Implementation

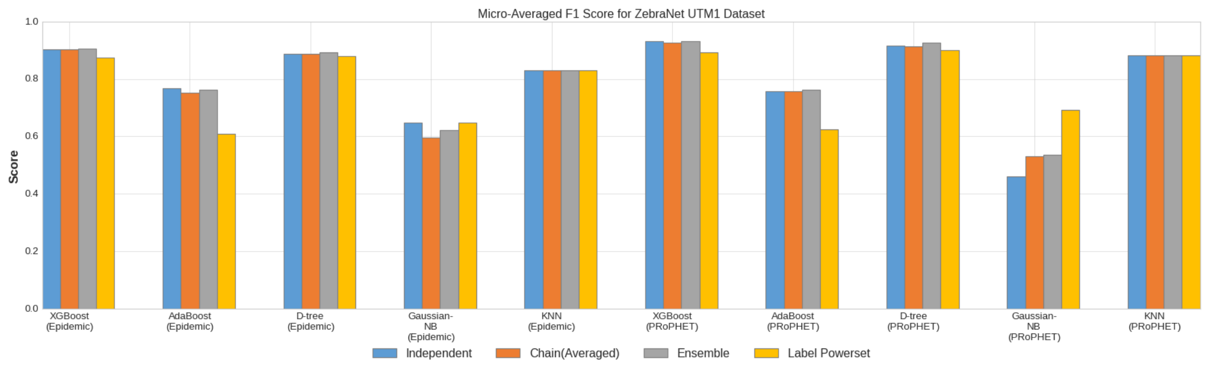
a. XGBoost is the best performing classifier.
b. PRoPHET router performs better than Epidemic router.
c. Gaussian NB is less performant than expected.

2. f1 score

- A weighted average of the precision and recall.
- Precision and recall are calculated by counting the total true positives tp, true negatives tn, false negatives fn and false positives fp for examples classified as label l.
- Higher score→ More accurate classification



Original Implementation

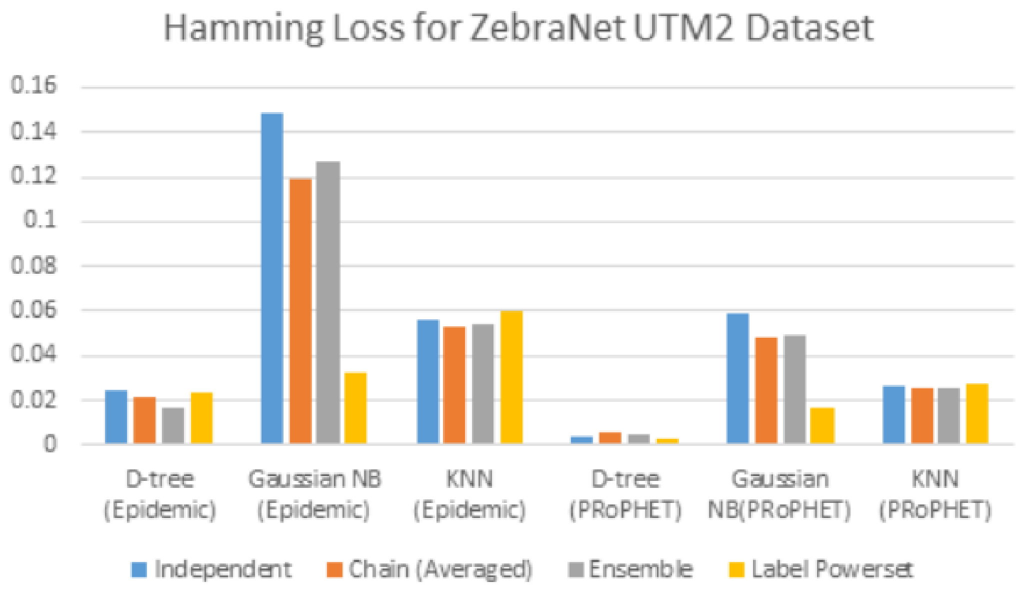


Our Implementation

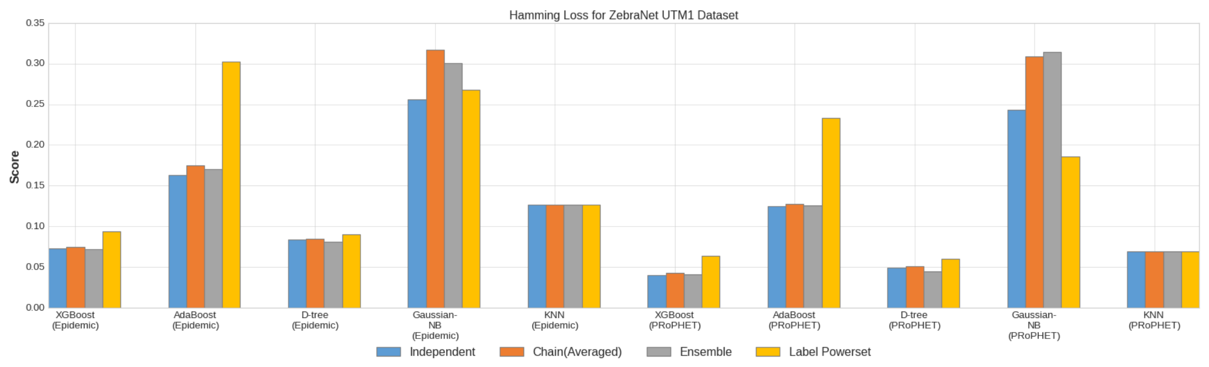
a. XGBoost is the best performing classifier.
b. Epidemic router performs marginally better than PRoPHET router.
c. KNN deviates from the original result, giving a much better score than expected for both routers.

3. Hamming Loss

- Calculates the fraction of labels incorrectly classified.
- Lower value → Lesser misclassifications, better results



Original Implementation

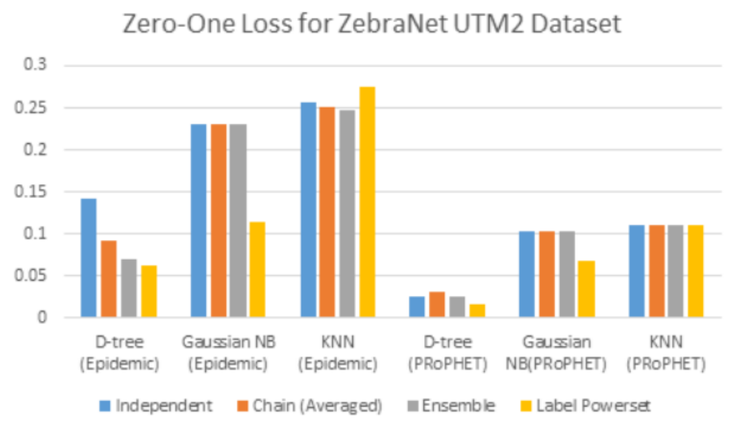


Our Implementation

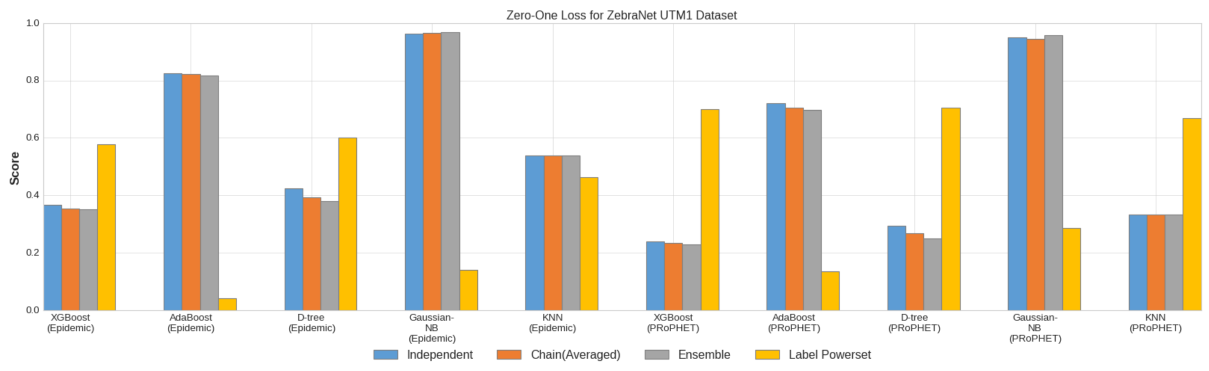
a. XGBoost is the best performing classifier.
b. PRoPHET router performs better than Epidemic router.
c. Label Powerset is an outlier in case of Gaussian NB, while the rest follow the same trend.

4. Zero-One loss

- Similar to Hamming Loss but considers the entire prediction incorrect if any label in the prediction is incorrect.
- Lower value → Lesser misclassifications, better results



Original Implementation



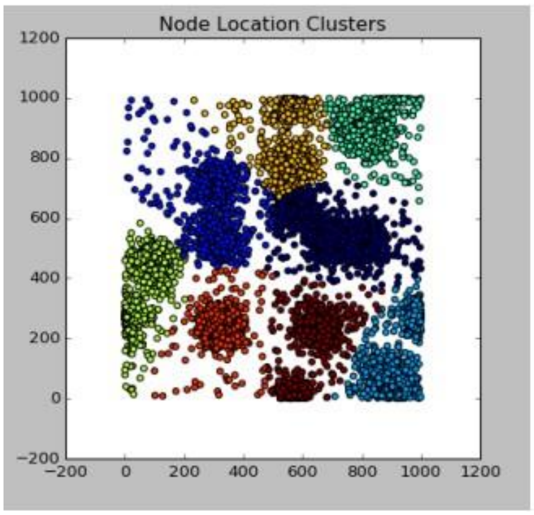
Our Implementation

a. XGBoost is the best performing classifier.
b. ProPHET router performs better than Epidemic router.
c. Label Powerset is an outlier in all cases.
d. KNN is more performant and Gaussian NB is significantly less performant than expected.
e. All scores are much higher than expected.

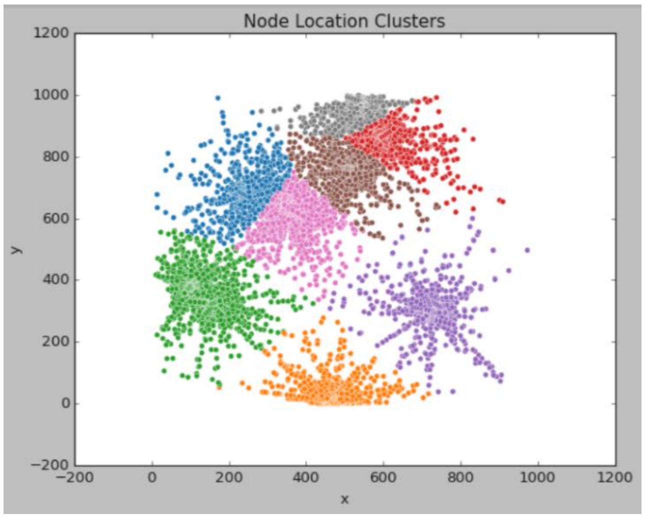
5. Results

1. We have extended the results of the paper for XGBoost and AdaBoost Classifiers. Their performance compared to the previous best classifier is XGBoost > Decision > AdaBoost.
2. XGBoost is the best performing classifier on all 4 metrics with an improved performance compared to the previous best performer Decision Tree Classifier.
    a. XGBoost is an implementation of gradient boosted decision trees and has a significant improvement over Decision Tree on tweaking the parameters.
3. Overall, PRoPHET Router gives better results compared to Epidemic Router.
    a. Its physical significance can be that in case of Epidemic Router, due to some messages being forwarded to unnecessary nodes, they occupy the buffer space hence not allowing some of the actual vital forwards which are dropped due to congested buffer space.
    b. This is not an issue in PRoPHET which optimizes those forwards based on probability.
4. The best performing base classifier (DT) followed the original result, but there were some deviations with Gaussian NB and KNN.
5. The deviations could be due to the following –
    a. Different Network Emulators using different Bundle Protocol implementations – Original uses IBR-DTN in CORE (Common Open Research Emulator) while ours uses DTN-2 in ONE (The Opportunistic Network Environment).
    b. Differences in network parameters used like Buffer Size, Waiting Time etc.
    c. Deviations persist even after optimizing the classifiers. This can be due to the different ZebraNet UTM datasets used for movement traces (original uses UTM2, ours uses UTM1) and the node movement data extracted from them. For ex.

d. The node locations in our implementation are more concentrated as compared to the node locations in the paper which are more evenly distributed throughout.



Original implementation



Our implementation

6. Conclusions

1. XGBoost paired with PRoPHET Router gave the best performance overall.
2. Our results suggest that machine learning classification is a viable method to predict network traffic and determine the most likely nodes to be encountered on a given path which can be used to make more informed routing decisions, reducing overhead in epidemic-based routing approaches.

7. Scope for future work

1. For assigning regions to the nodes during clustering, their locations are taken during as a whole during the entire epoch(as a snapshot of an epoch). This method of clustering is very static in nature and there are other clustering methods more suited for a time-series data such as this as in some other works.[1][2]
2. Challenges in DTN Routing involve the optimization of-
    a. Routing
    b. Reducing the Routing Overhead
    c. Congestion Control of the messages on the nodes.
3. Deep Learning and Reinforcement Learning based methods can be explored to improve routing and congestion control in DTNs.[8]
4. In Deep Learning, methods using Neural Networks(NN) and Artificial Neural Networks(ANNs) [6][7] can be explored which make use of factors like
    i. Time Difference between Recent Connection and Last Connection
    ii. Frequency of Calls among others
5. In reinforcement learning, the feedback factors can be inconsistent due to timeliness of reward/feedback. Methods outlined by [3][4] use Q-learning-based DTN routing protocol(QDTR) in which the reward function makes use of sinks and features like
    i. relative distance to the sink
    ii. node density of an area
    iii. residual energy of an encountered node.
6. Double Q-Learning Routing(DQLR)[5] selects the next hop in a distributed way. DQLR decouples the selection and evaluation with two value functions i.e., the double Q-Learning functions. Every message in every node of the network is associated with these two functions.

7. References

1. R. Langone, R. Mall, and J. A. K. Suykens, “Clustering data over time using kernel spectral clustering with memory,” in 2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), Dec 2014, pp.
2. K. S. Xu, M. Kliger, and A. O. Hero, Iii, “Adaptive evolutionary clustering,” Data Min. Knowl. Discov., vol. 28, no. 2, pp. 304–336, Mar. 2014.[Online].Available: http://dx.doi.org/10.1007/s10618-012-0302-x
3. Hu, Tiansi, and Yunsi Fei. "An adaptive and energy-efficient routing protocol based on machine learning for underwater delay tolerant networks." In 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, pp. 381-384. IEEE, 2010.
4. Dudukovich, Rachel, Alan Hylton, and Christos Papachristou. "A machine learning concept for DTN routing." In 2017 IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE), pp. 110-115. IEEE, 2017
5. Yuan, Fan, Jaogao Wu, Hongyu Zhou, and Linfeng Liu. "A Double Q-Learning Routing in Delay Tolerant Networks." In ICC 2019-2019 IEEE International Conference on Communications (ICC), pp. 1-6. IEEE, 2019
6. Segundo, Fabio Rafael, Eraldo Silveira e Silva, and Jean-Marie Farines. "A DTN routing strategy based on neural networks for urban bus transportation system." Journal of Network and Computer Applications 64 (2016): 216-228
7. Singh, Ajay Vikram, Vandana Juyal, and Ravish Saggar. "Trust based intelligent routing algorithm for delay tolerant network using artificial neural network." Wireless Networks 23, no. 3 (2017): 693-702
8. R. Amirthavalli, S. Thanga Ramya, “Machine Learning in Delay Tolerant Networks: Algorithms, Strategies, and Applications” in International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-9 Issue-1S, November 2019
9. ONE Simulator configuration with eclipse - http://www.iosrjournals.org/iosr-jece/papers/Conf.15010/Volume%201/S.ECE-154.pdf
10. Mailing List archives - http://theonekb.pythonanywhere.com/
11. Configuration files - http://delay-tolerant-networks.blogspot.com/p/one-tutorial.html
12. Parametrized simulation - http://delay-tolerant-networks.blogspot.com/p/parametrized-simulations.html
13. http://one-simuator-for-beginners.blogspot.com/2013/08/one-simulator-introduction.html
14. http://delay-tolerant-networks.blogspot.com/
15. http://www.iosrjournals.org/iosr-jece/papers/Conf.15010/Volume%201/S.ECE-154.pdf