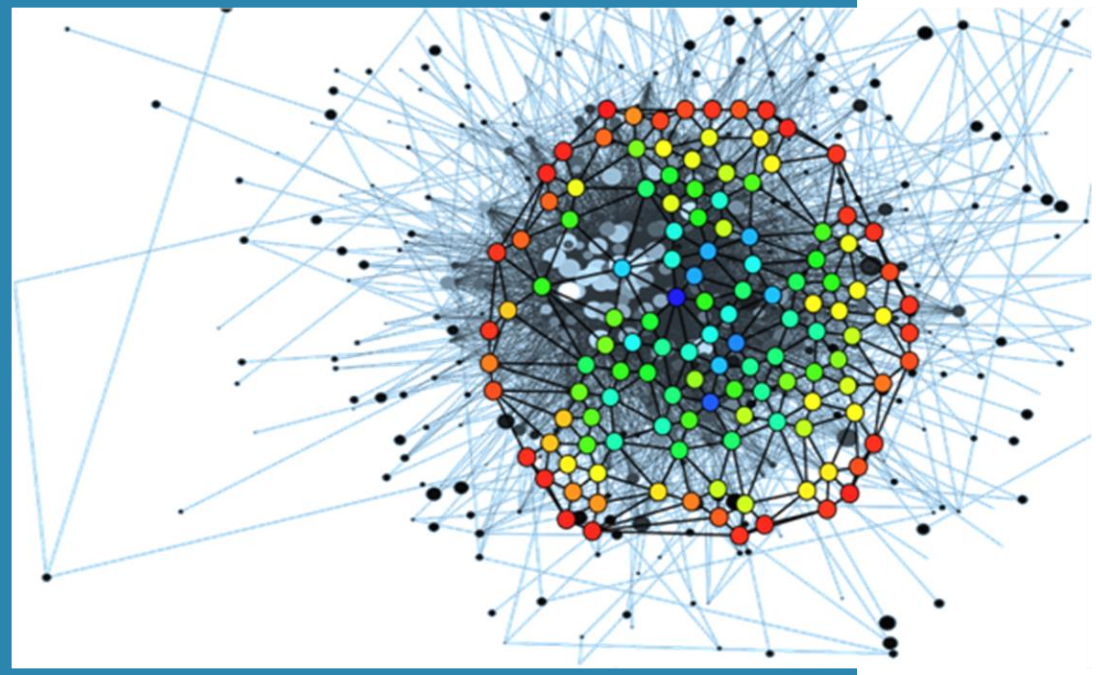
Delay Tolerant Network Routing as a Machine Learning Classification Problem

-Mayank Jain(2019A7PS0141P)



by Unknown Author is licensed under

Contents

1. Introduction
2. Related Works
3. Algorithm Development
4. Implementation
5. Evaluation
6. Results
7. Conclusion
8. Scope for future work
9. References

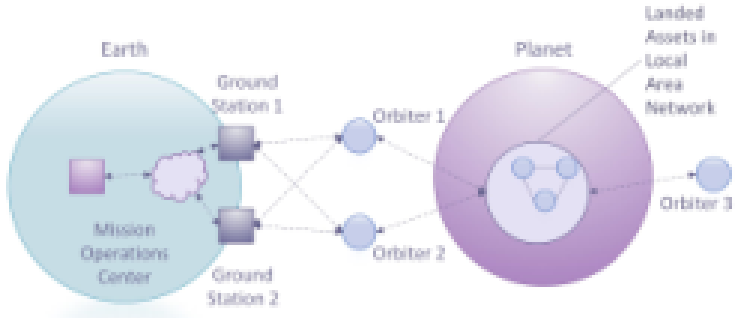
Introduction

1. What are DTNs?

- Delay tolerant networking (DTN) is 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.



Deep Space Scenario with Rovers and Orbiter Relay

Introduction

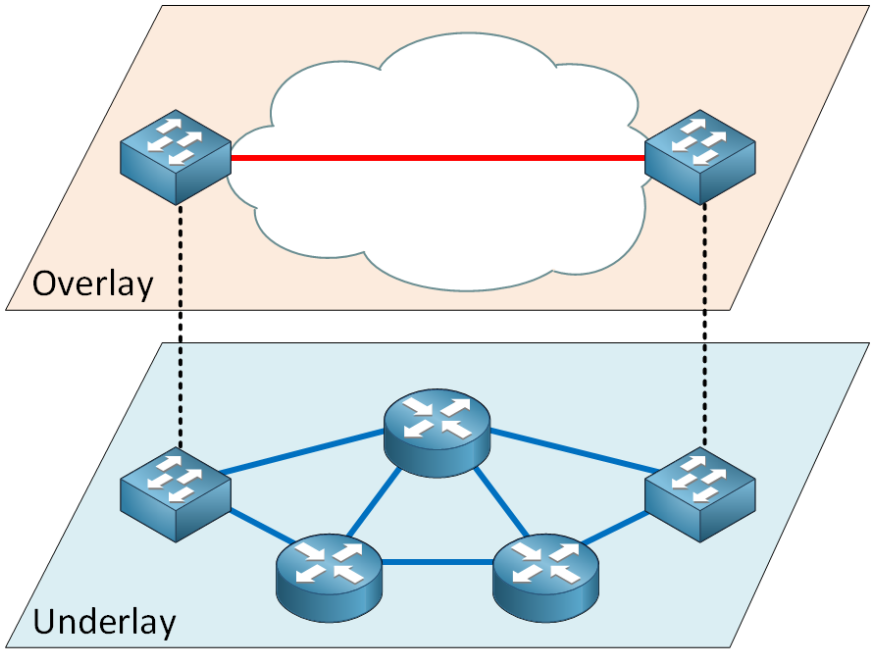
For a successful routing, we must address three issues-

1. 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.)
2. Link asymmetry between forward and return links – leading to data being sent faster than the acknowledgments being received.
3. Multiple protocol stacks being used in the path from source to destination creates a need for a common layer between the nodes.

Basics of DTN architecture and Bundle Protocol

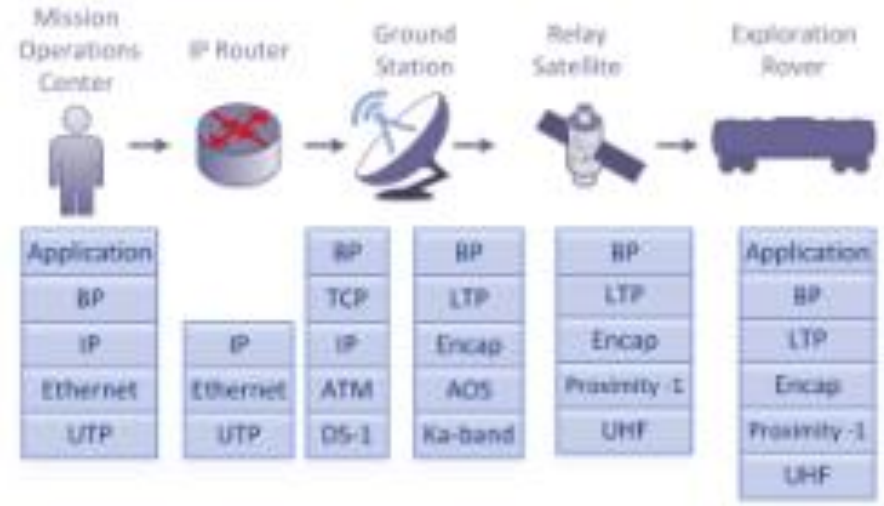
- These issues are addressed by Bundle Protocol and the DTN Architecture

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

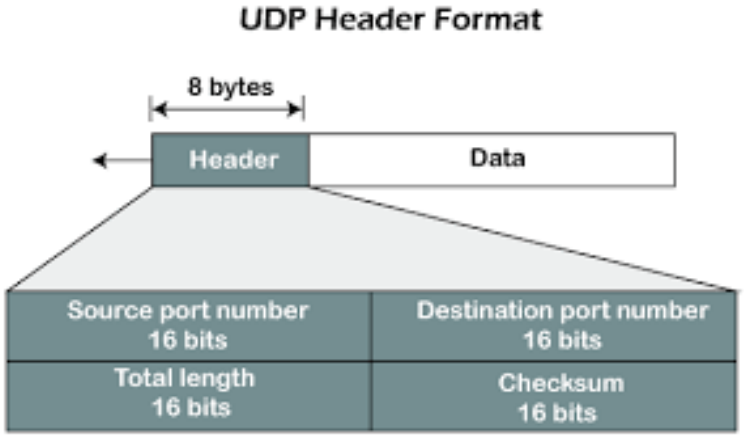


Basics of DTN architecture and Bundle Protocol

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



Example Mission Operations Center to Deep Space Rover Protocol Stack



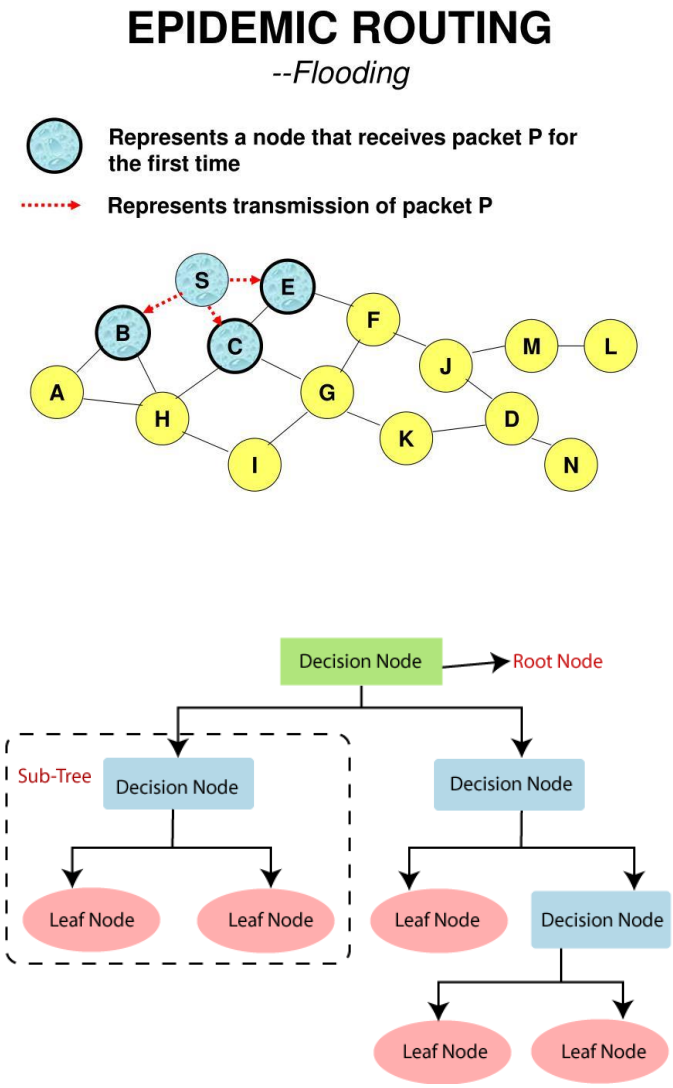
UDP Datagram

Related Work

1. Opportunistic Routing(OR) for DTNs
   - Most existing methods are based on epidemic routing
   - 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.
   - It is very resource hungry as it makes no attempt to eliminate replications that would be unlikely to improve the delivery probability of messages.
   - 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.
   - E.g. PRoPHET routing protocol is based on past encounters and takes probability into account to determine the best node.

Related Work

2. Machine Learning Techniques applied to routing in DTNs
   - 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.
   - Other methods use Bayesian classifiers with attributes such as network regions, a time-based index and message destination.
   - Both methods use stored network traffic history as samples to train their classifiers.



Decision Tree-based Classifier

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

Algorithm Development

- We don’t use Reinforcement Learning as it has many drawbacks in this scenario –
  - Time as a metric is inconclusive here as
    - While Poor routing choices lead to delay, delay times due to environmental conditions are out of control of learner
    - Feedback can be inconsistent (TCP/IP acknowledgments also depend on conditions out of control of learner).
  - As the timeliness of feedback is important in the performance, we use other methods.

- The goal is essentially to determine the future network state based on the history of the network.
- Though several factors determine the state of the network nodes such as the set of neighboring nodes, duration of the contact period, data rate, buffer capacity etc. , we can assume that they follow some predictable periodicity in behavior
- We call this period an epoch and divide it further into time slices
- Each node will have some pattern of mobility and data generation within the epoch that will likely repeat itself over time.

Algorithm Development

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

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

1. For a message, we are classifying whether the message has been forwarded to the ith node where i ∈ [0,numNodes)
2. So, this is a single multi-label classification problem split into multiple binary classification problems.
3. 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.

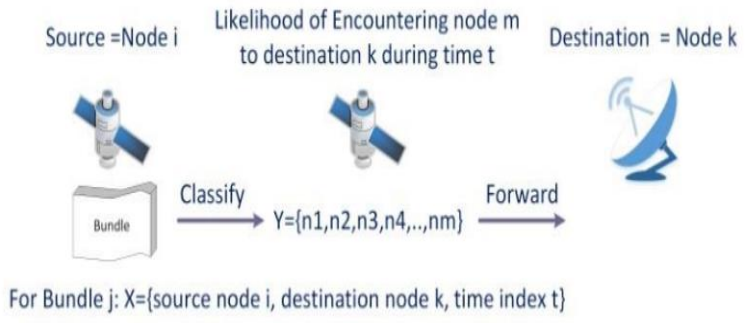


Fig. 4. Example Classifier Attribute Vector and Prediction

Algorithm Development

1. Our classifier takes following input features (X) →
2. (tsIdx, fromHost, toHost, region_x, region_y, deliveryStatus)
3. And returns output labels(Y) → (d0-d9) [in case of 10 nodes]
4. for each entry indicating if the message visited i'th node or not.

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

The final dataset looks like this

	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

Algorithm Development

- We select five well-known classifiers (XGBoost, AdaBoost, Naïve Bayes, Decision Tree and K-Nearest Neighbors), to determine which would provide the best performance.
- The input to our classifier is based on an attribute vector X consisting of the time index in the epoch, the source node, the destination node, and if the message was delivered or not (1 or 0).
- Only the X attributes (time, source and destination) are given to the model and it will output a prediction for the most likely set of nodes a message will be forwarded to.
- The label data Y, or output of the classifier, is the set of nodes that the message was forwarded to. This is encoded as an n-bit string where n is the number of nodes in the network. If the message has visited node i, then the bit in position i is set, it is zero otherwise.
- The performance of the classifier is evaluated by comparing the actual output Y of the test set to the output of the prediction.

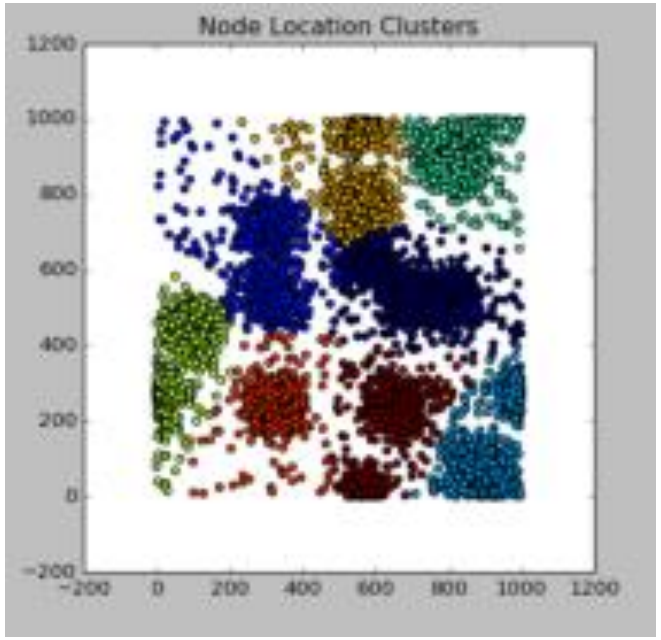
Algorithm Development

- This method divides the routing classification problem into n separate problems, with one classifier that produces a binary output indicating a node is or is not a member of the set of nodes along a given route. This is a multi-label classification approach.
- A single classification problem with multiple outputs is transformed into multiple classification problems. This can lead to performance issues.
- Also, it classifies all labels separately, without taking into account the interdependence between outputs.
- Machine learning along with Ensemble Classifier Chains(ECC) solves the performance issue of multilabel classification and takes interdependence into account.

- Evaluation metric - We first begin to evaluate different node attributes by taking into account node location.
- We use the K-means clustering algorithm to determine regions in which nodes frequently visit, which can then be used as another attribute to our classifier.
- Rather than simply dividing the area into equal partitions, the K-means clustering algorithm will provide a data-driven approach to grouping node locations.

Implementation

1. After clustering, the region is assigned back to the database entry corresponding to a given node at a given time based on the known location coordinates at that time.
2. Clustering was performed without taking time into consideration(the whole simulation as one).



Assigning node location regions with K-means Clustering

1. The classifier is trained with historical values for each message sent in a test emulation consisting of attributes X and the forwarded node string corresponding to each message.
2. Classifier Chains(CC) handles the problem of multi label classification by extending the binary labels of the previous classifiers to the current one. So, the performance here heavily depends upon the order of the labels selected.
3. Ensemble Classifier Chains(ECC) uses multiple randomly ordered classifiers, so the impact of order selection is reduced.

Implementation

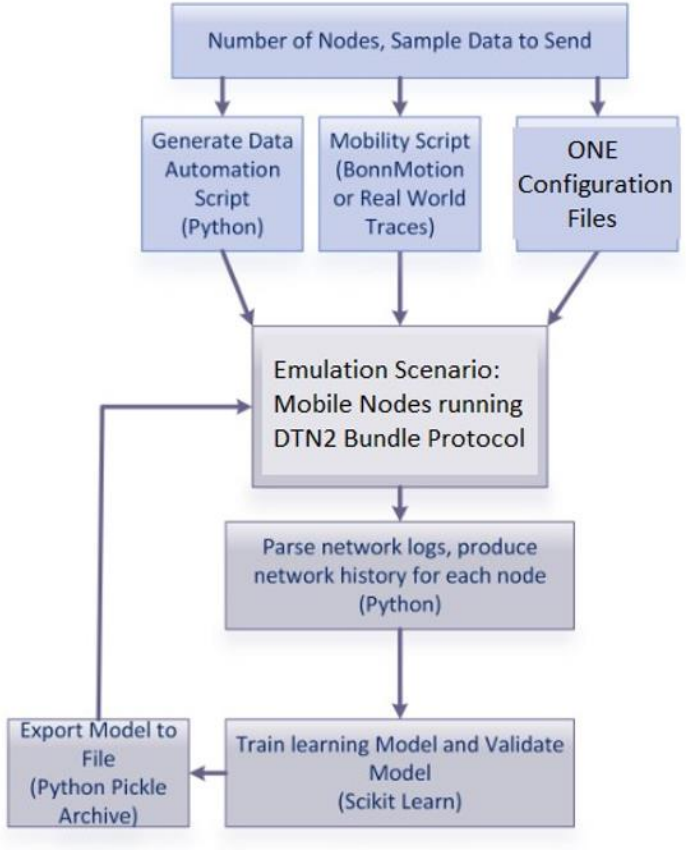


Fig. 5. Emulation Tool Chain

- DTN-2 is used as a basis for the software development.
- ONE Simulator(The Opportunistic Network Environment) is used.
- Nodes are moved in the emulator using mobility scripts based on the NS-2 network simulator script and BonnMotion mobility scenario generator.
- 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.

Implementation – 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 step1.

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.

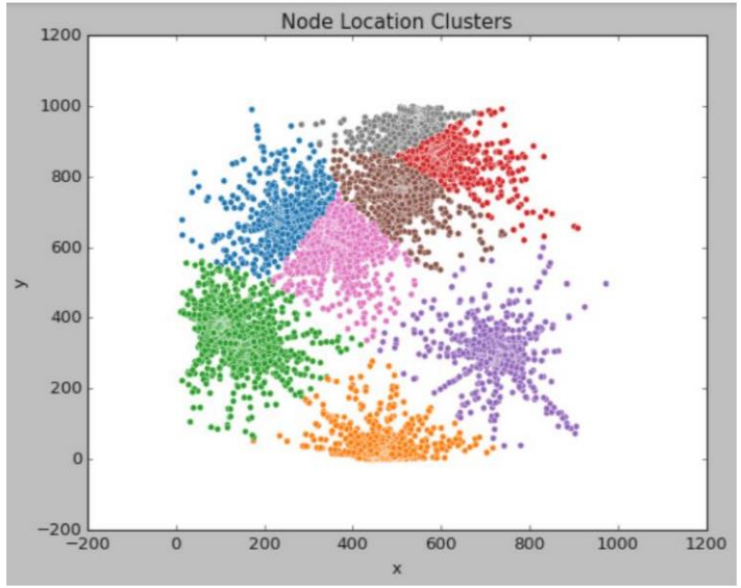
Implementation

```
def get_delta_xy(filename):
    '''Reads through filename and extracts the delta movement for x, y'''

    file = open(filename, 'r')
    dist_list = []
    direct_list = []
    ln = 0
    for line in file.readlines():
        ln += 1
        x1, y1 = extract_position(line)
        if ln > 1:
            distance = math.sqrt((x1-x0)**2+(y1-y0)**2)
            dist_list.append(distance)
            direction = math.atan2(y1-y0, x1-x0)
            direct_list.append(direction)
        x0, y0 = x1, y1  # save x & y
    file.close()
    return dist_list, direct_list
```

```
def get_delta_direction(direct_list):
    'Calculate the delta direction'
    dd = []
    for i in range(len(direct_list)):
        if i > 0:
            dd.append(direct_list[i]-direct_list[i-1])
    return dd
```

Preprocessing and Feature Extraction



Assigning node location regions with K-means Clustering

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.

3. Training - 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 & AdaBoost - We use tree booster since it always outperforms linear booster.

* 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.

* So finally, we use hyperopt library for tuning the parameters. Since it doesn't allow for multi-label classifications natively, we find the optimum parameters for each label separately through binary classifications for each label.

* 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. Then we run the complete program within the range of all the optimum parameters combined to find the best parameters.

* This process took many iterations since it was done manually.

3. Training - **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 evaluate them on the following metrics.

Evaluation

The output string has n labels which classify the nodes (0 or 1) that the message was forwarded to.

To validate the multi-label classification performance, four well known multi-label prediction metrics are used.

1. **Jaccard similarity score** - or multi-label accuracy 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.
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.

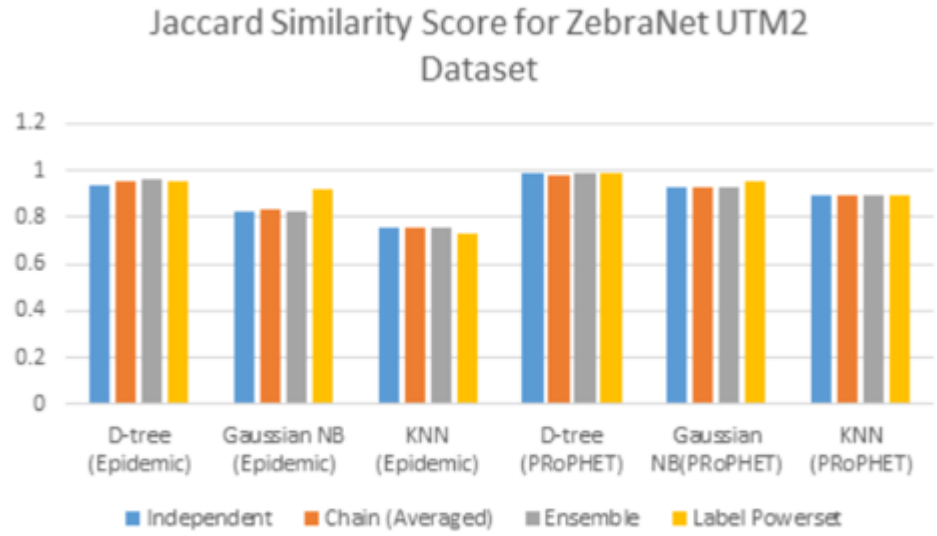
   In the above two metrics a higher score implies more accurate classifications

3. **Hamming Loss** - Calculates the fraction of labels incorrectly classified.
4. **Zero-one loss** - similar to Hamming Loss but considers the entire prediction incorrect if any label in the prediction is incorrect.

   In both of the above metrics, a higher value represents more misclassifications.

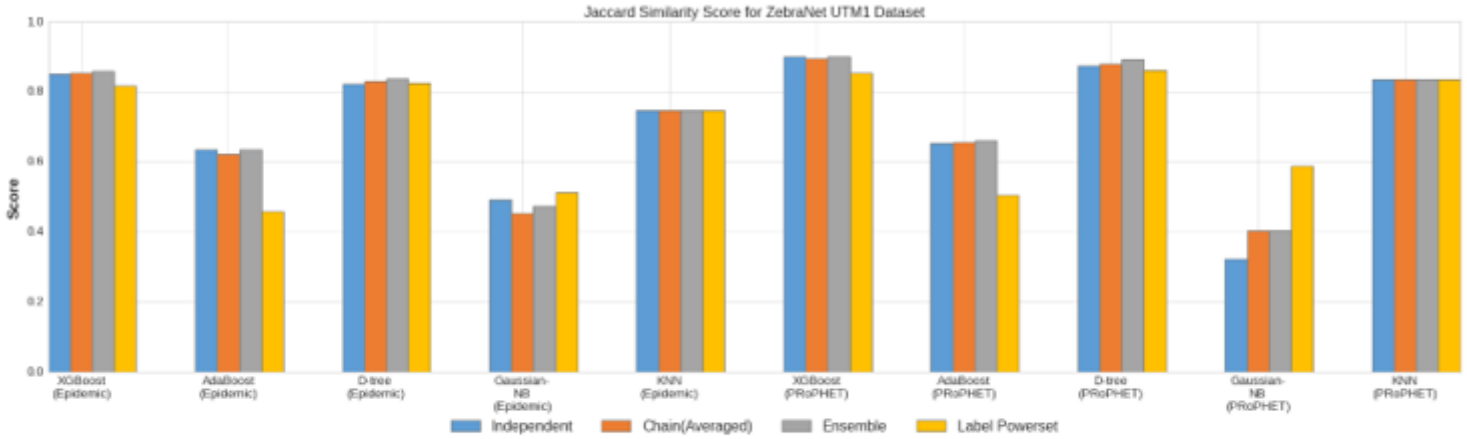
Evaluation – Metrics

1. **Jaccard similarity score**
    1. Higher score→ More accurate classification



Original Implementation

1. XGBoost is the best performing classifier.
2. PRoPHET router performs better than Epidemic router.
3. Gaussian NB is less performant than expected.

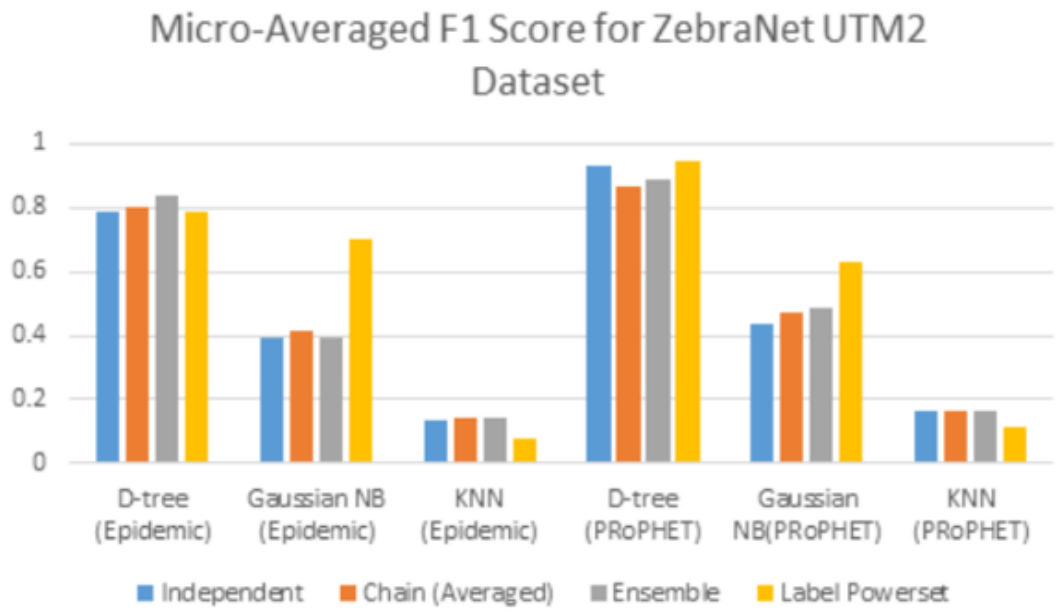


Our Implementation

Evaluation – Metrics

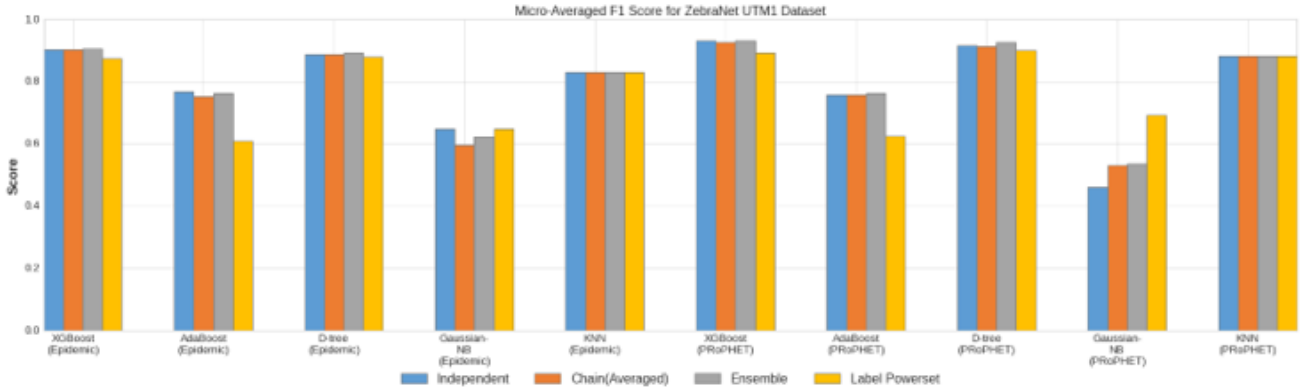
2. f1 score

1. Higher score→ More accurate classification



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

Original Implementation

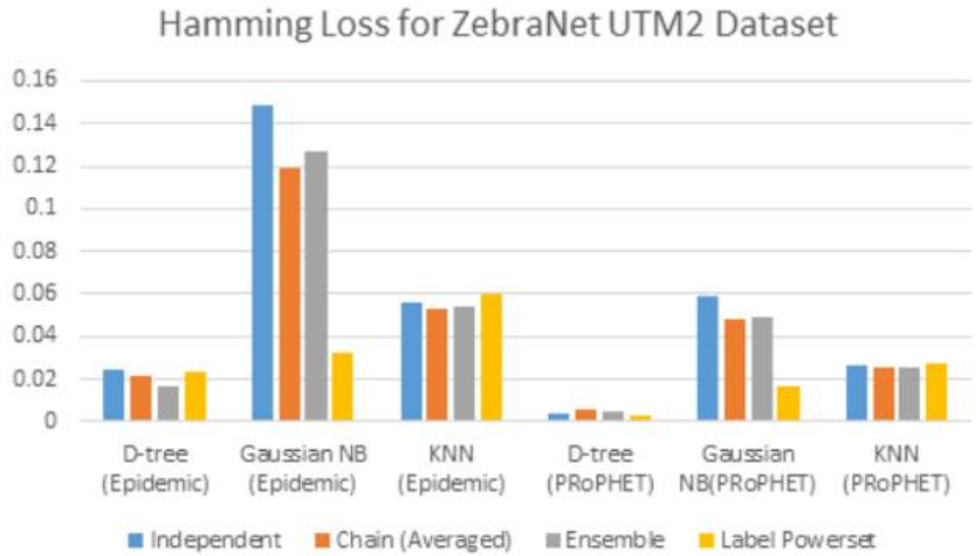


Our Implementation

Evaluation – Metrics

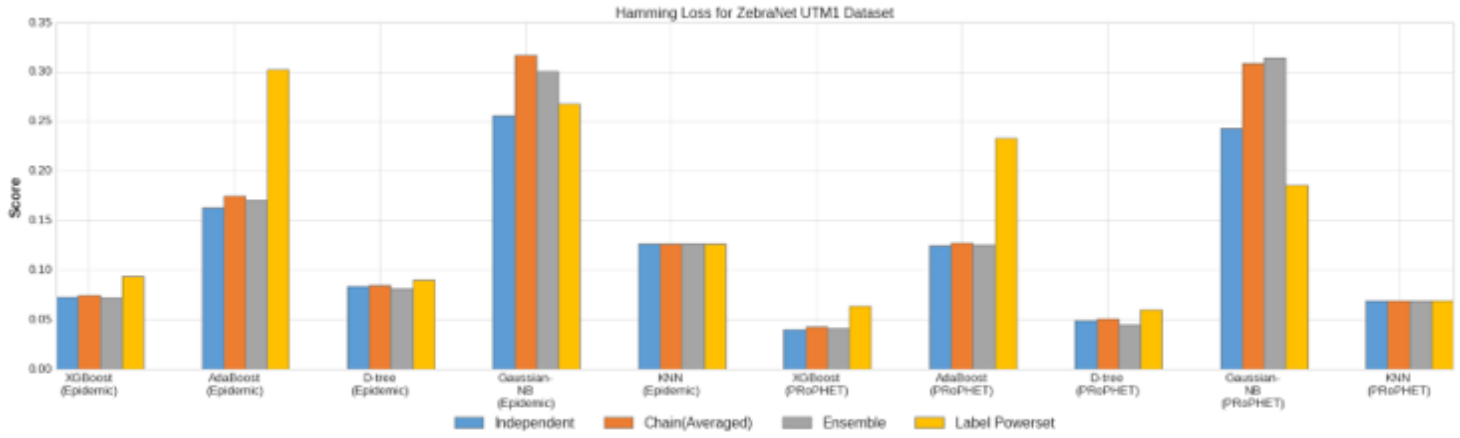
3. Hamming Loss

1. Lower value → Lesser misclassifications, better results



Original Implementation

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

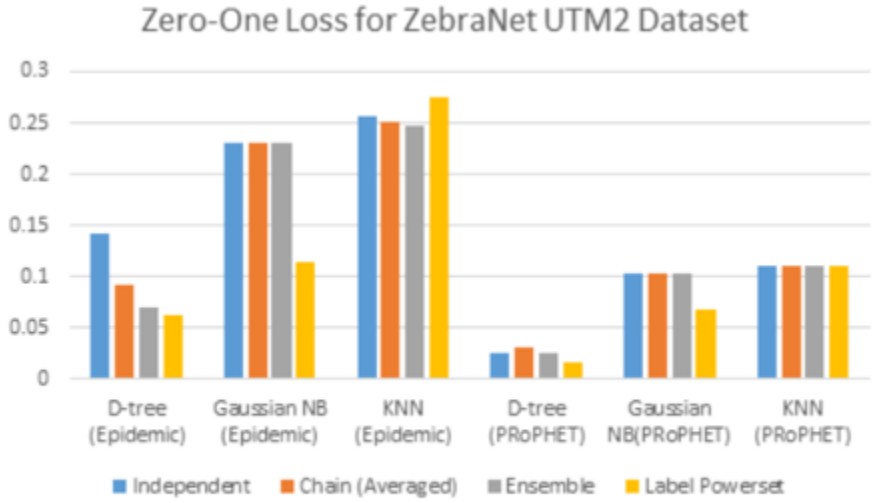


Our Implementation

Evaluation – Metrics

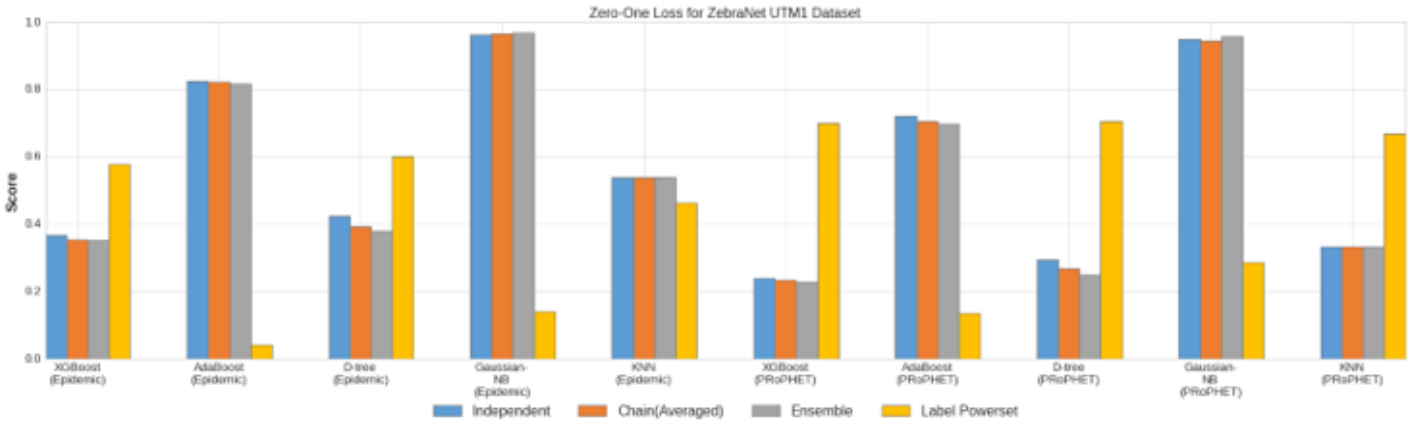
4. Zero-One loss

1. Lower value → Lesser misclassifications, better results



Original Implementation

1. XGBoost is the best performing classifier.
2. ProPHET router performs better than Epidemic router.
3. Label Powerset is an outlier in all cases.
4. KNN is more performant and Gaussian NB is significantly less performant than expected.
5. All scores are much higher than expected.



Our Implementation

Results

1. The base classifier selected (Naive Bayes, Decision Tree, and K-Nearest Neighbor) had a much greater impact on performance than either the under- lying routing method used for data collection (PRoPHET, epidemic and flooding) or the method of multi-label classification.

2. 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.

3. There were some performance gains when an appropriate number of clusters were selected when the node region was used as an attribute to our classifier, paired with the existing ones.

Results

1. XGBoost is the best performing classifier on all 4 metrics with an improved performance compared to the previous best performer Decision Tree Classifier.

2. 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.

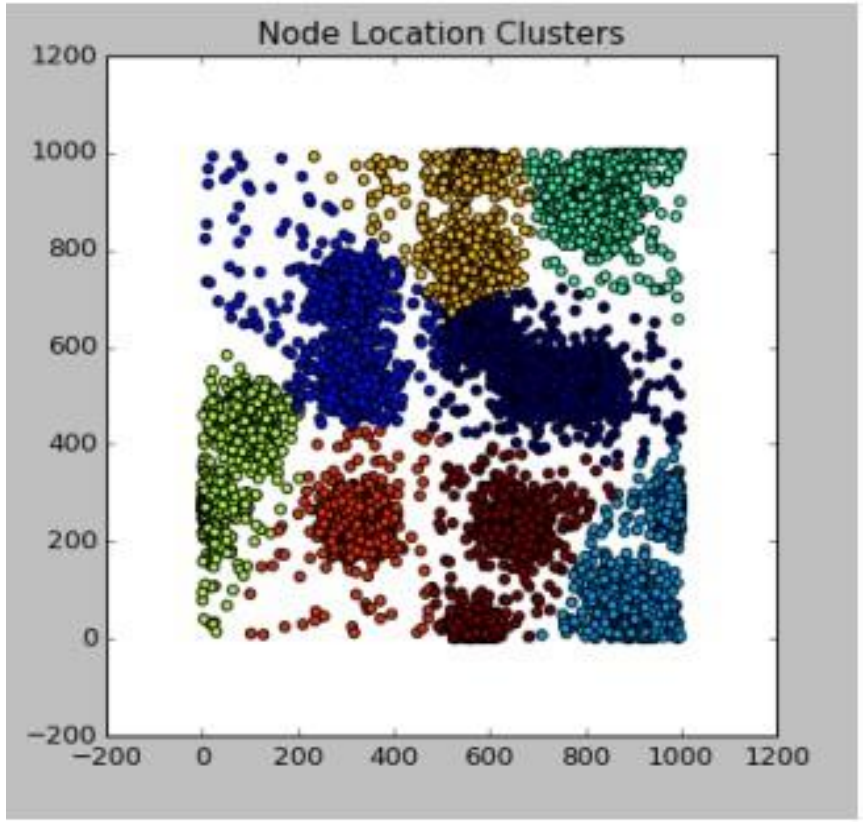
Results

3. The best performing base classifier (DT) followed the original result, but there were some deviations with Gaussian NB and KNN.

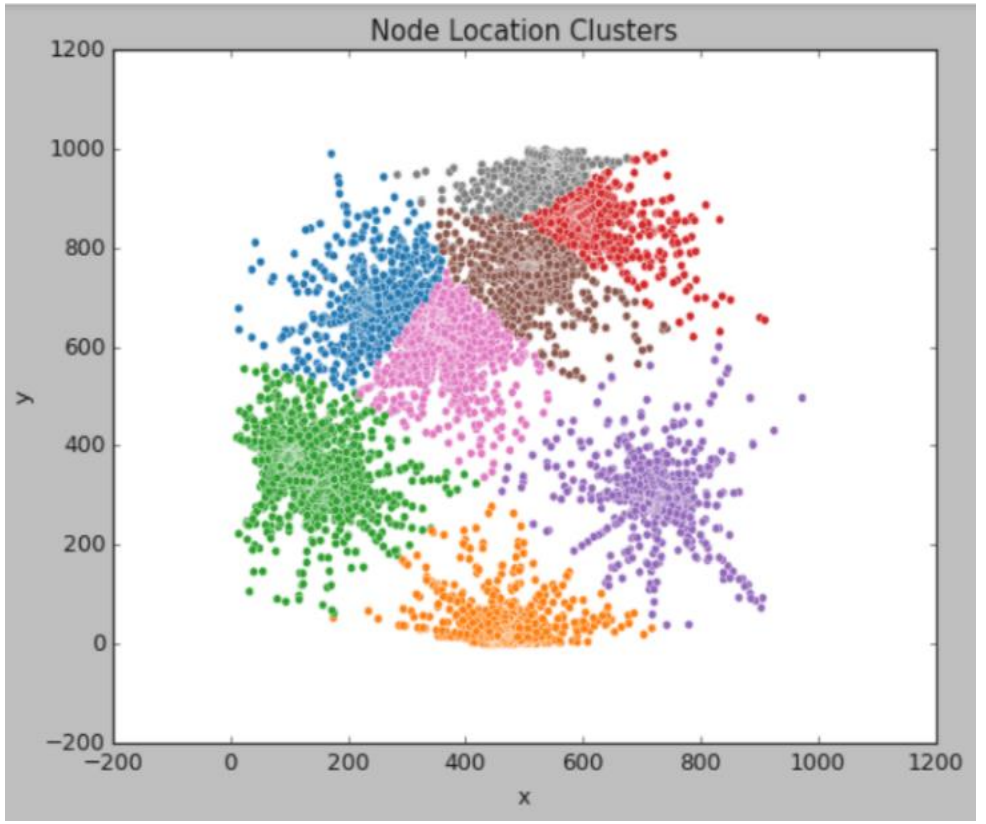
4. The deviations could be due to the following –
   a. Different Network Emulators using different Bundle Protocol implementations – Original uses IBR-DTN in CORE (Common Open Research Emulator) while ours uses DTN-2 in ONE (The Opportunistic Network Environment).
   b. Differences in network parameters 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.

Results

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



Original implementation



Our implementation

Conclusion

- XGBoost paired with PRoPHET Router gave an improved performance over the previous best performer Decision Tree Classifier.

- Our results suggest that machine learning classification is a viable method to predict network traffic and determine the most likely nodes to be encountered in a given path which can be used to make more informed routing decisions, reducing overhead in epidemic-based routing approaches.

- This method was explored as opposed to learning in real-time, since many classification algorithms involve a training phase, followed by model validation before they can be used to make predictions on new instances of data.

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.

Deep Learning and Reinforcement Learning based methods can be explored to improve routing and congestion control in DTNs.[8]

3. 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 other

Scope for future work -

4. 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.

5. 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.

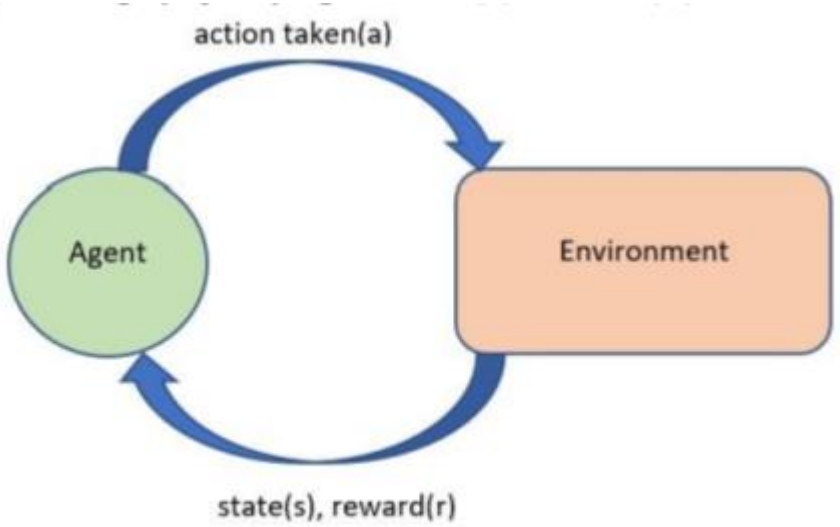


Fig. 4.Q-Learning Process

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

References

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. http://www.iosrjournals.org/iosr-jece/papers/Conf.15010/Volume%201/S.ECE-154.pdf
11. Mailing List archives - http://theonekb.pythonanywhere.com/
12. Configuration files - http://delay-tolerant-networks.blogspot.com/p/one-tutorial.html
13. Parametrized simulation - http://delay-tolerant-networks.blogspot.com/p/parametrized-simulations.html
14. http://one-simuator-for-beginners.blogspot.com/2013/08/one-simulator-introduction.html
15. http://delay-tolerant-networks.blogspot.com/

Appendix: ONE Configuration File

General Steps-

Below are the steps that we follow to perform our simulation. In fact, these are some steps that every simulation would involve.

- Specify the general settings for the scenario to be simulated
- Specify the network interface(s) of the nodes
- Create a group of nodes
- Specify the motion patterns
- Specify the traffic pattern
- Specify a set of reports to be generated

Appendix: General Settings

- Scenario.name: The value set for this is the name we give for current simulation. For each simulation this value should be changed, otherwise the reports generated for each simulation will be replaced with the new contents. So if you want separate report files for each simulation change names here.
- Scenario.simulateConnection: this should be set to true for simulations. In some cases we may use external traces (like those downloaded from CRAWDAD); in such cases the value is set to false.
- Scenario.updateInterval: This defines the interval in which the update() function in the MessageRouter.java file has to be called. As default it is set to 0.1s.
- Scenario.endTime: The time for which the scenario is to be simulated. You can keep it to any number of seconds. By default it is kept to 43200s which is 12hrs.

Appendix: Configuration File

Router-wise settings

1. PRoPHET router

```
Scenario.name = PRoPHET-%%ProphetRouter.secondsInTimeUnit%%siu
Group.router = ProphetRouter

ProphetRouter.secondsInTimeUnit = 30

# Define POI data files
PointsOfInterest.poiFile1 = data/ParkPOIs.wkt
PointsOfInterest.poiFile2 = data/CentralPOIs.wkt
PointsOfInterest.poiFile3 = data/WestPOIs.wkt
PointsOfInterest.poiFile4 = data/shops.wkt

# Define probabilities for different groups selecting POIs from different POI files
Group1.pois = 1,0.3, 2,0.1, 3,0.1, 4, 0.1
```

2. Epidemic Router

```
Scenario.name = Epidemic
Group.router = EpidemicRouter
```

Configuration File

To include the trace file

Scenario.simulateConnection: this should be set to true for simulations. In some cases we may use external traces (like those downloaded from CRAWDAD); in such cases the value is set to false.

```
Scenario.simulateConnections = false

## Trace information
Events1.class = ExternalEventsQueue
Events1.filePath = <your_trace_file>
```

The ExternalMovementReader class already documents the format of movement trace files.

First line of the file should be the offset header. Syntax of the header should be:

minTime maxTime minX maxX minY maxY minZ maxZ

Last two values (Z-axis) are ignored at the moment but can be present in the file.

Following lines' syntax should be:

time id xPos yPos

Report Generation

- We can specify the number of reports and the type according to our needs for each simulation. To do this we specify the required report classes.

```
# Reports - all report names have to be valid report classes
# length of the warm up period (simulated seconds)
Report.warmup = 0
# default directory of reports (can be overridden per Report with output s
etting)
Report.reportDir = reports/
# how many reports to load
Report.nrofReports = 2
# Report classes to load
Report.report1 = MessageStatsReport
Report.report2 = MessageDeliveryReport
```