Proximity Sensing for Contact Tracing

Sheshank Shankar*, Ayush Chopra*, Rishank Kanaparti*, Myungsun Kang†, Abhishek Singh*, Ramesh Raskar*
PathCheck Foundation
Cambridge, MA, USA
*{firstname.lastname}@pathcheck.org,
†sunny.myungsun.kang@gmail.com

***Abstract*—The TC4TL challenge is aimed towards designing an effective proximity sensing algorithm that can accurately provide exposure notifications. In this work, we describe our approach to model sensor and other device level data to estimate distance between the two phones. We also present our research and data analysis on the TC4TL challenge dataset and discuss various limitations associated with the task as well as dataset.**

I. Introduction

As economies open up, digital contact tracing is emerging as an important tool to help contain the spread of COVID-19 by providing exposure notification to susceptible individuals who came in close to infected individuals. There have been several proposals varying across different modalities, but Bluetooth is the most widely emerging technology for digital contact tracing due to the technology's aptness for the task. However, there are well known limitations with the effectiveness of Bluetooth , described in [7], [8]. We focus on the task of proximity sensing, which is the key to facilitating efficient contact tracing. Proximity sensing is concerned with predicting if two individuals have been in "close contact" for "too long" that may open the *possibility* of COVID-19 transmission. Specifically, we focus on the TC4TL challenge recently introduced by NIST.

Current approaches to automated exposure notification rely on using Bluetooth Low Energy (BLE) signals (or chirps) from smartphones to detect if a person has been too close for too long (TC4TL) to an infected individual. However, the received signal strength indicator (RSSI) value of Bluetooth chirps sent between phones is a very noisy estimator of the actual distance between the phones and can be dramatically affected in real-world conditions [5].

In this challenge, we're trying to predict the distance using the phone sensor data, particularly the received signal strength indicator (RSSI) from the Bluetooth Low Energy (BLE) signals (or chirps) along with other factors which have been seen to have an impact on these RSSI values. We train our networks using the datasets provided by NIST and MITRE, and the evaluation is derived from data sets being collected by organizations around the world studying this problem.

We trained and tested many networks with varying internal architectures, all with the singular goal of trying to obtain a model which understands the subtle nuances of the phone's sensor data (and other factors), to try and predict an accurate distance reading. We trained a few networks based on Deep Learning (*LSTM*, *ConvGRU*, etc.) and also networks with architecture based on Support Vector Machines and Decision Trees. Of all the networks we tested on the NIST Test Data (Subset of the full NIST dataset), the *Temporal Conv1D* network gave us the most favourable results.

We also ran a few tests and projections (*Ablation studies, Data analysis* and *Training and Dev set distribution discrepancy*) to try and get an idea of the practicality of this problem along with its future implications. We've discussed our observations and hypotheses in section V.

II. Challenge Description

The basic task in the NIST TC4TL Challenge is estimating the distance and time between two phones given a series of RSSI values along with other phone sensor data. Current approaches to automated exposure notification rely on using Bluetooth Low Energy (BLE) signals (or chirps) from smartphones to detect if a person has been too close for too long (TC4TL) to an infected individual. Some of the identified factors affecting distance estimation from RSSI values are (1) the number and time spread of observed chirps, (2) the carriage position of the phones (i.e., hand, front pocket, back pocket, etc), (3) bodies and barriers between phones, and (4) multi-path signals from surfaces (e.g., indoor vs outdoor). To better characterize the effectiveness of range and time estimation using the BLE signal, the dataset collects Bluetooth chirp data as well as other phone sensor data (e.g., accelerometer and gyroscope) between various types of phones with simulated real-world variability. The dataset is divided into chunks of 4-sec device interactions (sender/receiver) with corresponding readings for each sensor.

For ease of analysis, the current version of the challenge restricts focus to estimation of the range (distance) and not the time duration.

The initial experiments were conducted on a subset of the PACT dataset for training the model. However we were soon asked to train the model from the MITRE dataset to prevent overfitting or underfitting.

III. Methodology

Bluetooth and other mobile sensor data tend to be extremely noisy, therefore our main strategy was to try and exploit the temporal characteristics of the dataset.

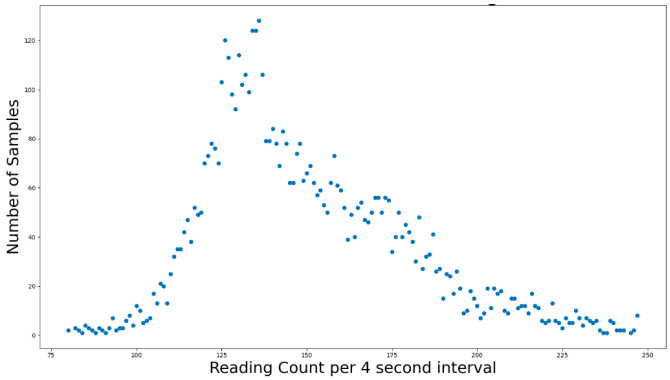

Fig. 1: Distribution of # of Sensor Reading Counts

A. Data Processing

We model the task as a time series task and break-down the dataset into *150* timesteps for each 4-sec chunk. The number of timesteps per interval is chosen to minimize the need for oversampling or under-sampling datapoints for each file to mitigate noise.

We chose 150 timesteps as the ideal number of timesteps per 4 second interval by calculating the mean of the number of timesteps throughout the entire dataset.

Every time-step is represented by a normalized fixed-length feature vector representing the most recent values for each sensor. In addition, the metadata of the experiment (TXDevice, RXDevice, TXPower, etc.) is one-hot encoded and concatenated to each time-step's vector. For the models that do not use a time series input, all readings are concatenated into a single feature vector for each 4 second interval.

We have also tried using mix-up data augmentation to increase the size of the training data, however, it did not provide a significant performance increase. We use the entire MITRE train set training, however, we did try creating an optimal subset by selecting only the k nearest train set neighbors of each point in the NIST development set.

B. Model Architecture

1) Deep Learning based Models:

- *LSTM:* Using the time-series input format, we've implemented an LSTM network and experimented with multiple layers, varying hidden sizes.
- *ConvGRU:* The ConvGRU is a GRU with Conv1d reset, update, and output gates. Using the time-series input format, we've implemented an ConvGRU network and experimented with multiple layers, varying hidden sizes and kernel sizes. We have also experimented with adding fully connected layers after the ConvGRU.
- *Conv1d:* We drew similarity of our learning task to that of what was tackled by Google's Wavenet. Wavenet leveraged 1D convolutional neural net for predicting the sequential audio signal. Inspired by this approach, we've implemented three distinct variations of 1D convolutional neural net, differing in ways to regularize the neural net: 1D CNN + Dropout, 1D CNN + Dropout + Maxpool, 1D CNN + Dropout + Dilation. In addition, we've experimented with hyper parameters, such as number of epochs, batch size, weight decay and learning rates.
- *Feed Forward:* Using the concatenated time-step input format, we've implemented a feed forward neural network. We experimented with multiple layers, varying percentages of dropout, and different activation functions.

2) Support Vector Machines: Using the concatenated time-step input format, we've implemented variations of the support vector machine [3], specifically, Nu-Support Vector Classification and C-Support Vector Classification.

3) Decision Tree based Models: Using the concatenated time-step input format, we've implemented XGBoost [2] and Random Forest Classification [1].

IV. Results

All experiments were run on a Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz server with 528 GB RAM, 48 cores, and on a single GPU. As can be seen in the nDCF results table [I], the temporal Conv1D network has the lowest nDCF out of all models trained on the MITRE train set.

The XGBoost and SVM based models would take too much time/memory to train on the full train set, so we present results on partially trained networks. These partially trained networks tend to completely overfit (over 99% accuracy) on the training data. The other model architectures that are trained on the complete MITRE dataset, complete within 3 hours.

All experiments are optimized using the Adam optimizer [6]. The temporal networks were built using PyTorch, whereas the decision tree and Support Vector Machine [3] based models were implemented with scikit-learn [9]. In the results presented, we do not use mix-up data augmentation [10] or the k nearest neighbors [4] method.

V. Analysis

In this work we analyze the feasibility of the problem by investigating the data distribution and different modalities present in the data and how they affect the overall results and outcomes.

a) Ablation studies: For ablation studies, we train the model in variety of input data streams with each data stream referring to only feeding a subset of input to a given model. This way, we try to estimate the role of different types of data and which sensors could be useful for the TC4TL task described previously. At the time of writing this report, our ablation study is not completely exhaustive, hence we share only a limited number of findings. Using our existing typical training scheme as described in the section III, we exclude some of the sensors from the input pipeline and train on the rest of the data. This requires minimal adjustment in the first layer of the neural network to accommodate varying sized input feature vector.

We perform the first set of experiments by excluding device level information such as TXDevice, RXDevice, TXPower, RxPower, Device carriage, and activity. We do not observe

Network Description	Train Set	Train %	Epochs	Batch Size	1.2m FINE	1.8m FINE	3m FINE	1.8m COARSE
GRU	NIST dev	90.0	200	100	0.65	0.13	0.28	0.08
ConvGRU	**NIST dev**	**100.0**	**200**	**100**	**0.37**	**0.04**	**0.23**	**0.02**
ConvGRU	MITRE	100.0	500	4000	1.07	1.0	0.98	1.05
LSTM	MITRE	100.0	40	100	1.0	1.08	0.93	0.97
GRU	MITRE	100.0	40	100	1.02	0.99	0.93	0.97
Feed Forward	MITRE	100	100	500	0.71	0.79	0.85	0.75
Temporal Conv1D	**MITRE**	**100.0**	**100**	**50**	**0.62**	**0.61**	**0.59**	**0.53**
C-SVC	MITRE	1.0	-	100	1.01	0.97	0.97	1.01
Nu-SVC	MITRE	1.0	-	100	0.82	0.8	0.78	0.69
XGBoost	MITRE	2.0	-	100	1.0	1.04	1.03	1.04
Random Forest	MITRE	100.0	-	100	1.0	1.05	1.02	1.1

TABLE I: nDCF RESULTS ON THE DEV DATASET FOR THE DIFFERENT MODELS.

Network Description	Train Set	Prediction Time
GRU	NIST dev	0.02
ConvGRU	NIST dev	0.02
ConvGRU	MITRE	0.02
LSTM	MITRE	0.02
GRU	MITRE	0.02
Feed Forward	MITRE	0.01
Temporal Conv1D	MITRE	TBD
C-SVC	MITRE	0.4
Nu-SVC	MITRE	0.4
XGBoost	MITRE	0.01
Random Forest	MITRE	0.01

TABLE II: WALL CLOCK TIME RESULTS FOR ALL OF OUR EXPERIMENTS. EXPERIMENTS RAN ON INTEL(R) XEON(R) CPU E5-2650 V4 @ 2.20GHZ SERVER WITH 528 GB RAM, 48 CORES, AND ON A SINGLE GPU.

any significant performance improvement with including or excluding the device level data with performance being around 35% on the dev set. However, we found the training to be more unstable when we included this device level information and more susceptible to overfitting on two classes instead of uniformly training across all of the four classes.

In the next set of experiments, we try training our model on different combinations of sensors and evaluate its performance on the dev set. As there could be a large number of combinations to try out, we only try the combinations which make sense from the physics based modeling view of the dataset. In the first experiment, we train the model just with the bluetooth readings given in the dataset, excluding all other sensor data. The performance does not reduce significantly even with the bluetooth data but it does seem to be increasing the divergence in the final training and testing accuracy, indicating higher susceptibility to overfitting. In the next experiment, we add up more sensors which make intuitive sense for the forward modeling of the data recorded by bluetooth, like gyroscope (captures orientation of bluetooth antenna), accelerometer (captures relative linear motion) and magnetometer (captures the variance in the magnetic aberration in the environment over the time domain). Across all of the ablation studies, we get best performance on test as well as training set with this particular combination. We do try including and excluding other sensory input also like altitude, attitude, heading and etc.

b) Data analysis: We perform few low dimensional projections of the dataset to understand if there are any underlying clusters in the feature vectors or their components which capture the dataset decision boundary with respect to the respective target class. Figure 2 and Figure 3 show the principal component analysis for the two distributions, training and dev respectively. It can be observed that training data clusters are packed heavily with no clear decision boundary for any two labels. However, there can be a higher dimensional embedding which can have separating hyperplanes across the classes.

c) Training and Dev set distribution discrepancy: While the initial training and dev distribution was provided by the NIST team, the finalized training dataset was provided by the MITRE group which differed in several ways from the dev and the testing set provided by the NIST group. Another visible and significant difference in the two distribution is around the variety of devices used by the training dataset compared to the dev set. The dev dataset uses roughly 5 extra iPhone devices for the measurement compared to the training set which prevents any data driven model to capture invariances in the signal arising from the missing devices.

Another significant challenge we encounter across all of the model training and experiments, is around the lack of generalization of the models trained on training dataset. The training of models on dev set and evaluation on test set resulted in near perfect accuracies indicating certain amount of overfitting. At the same time training on the training dataset and evaluation on dev set indicated a level of result slightly better than the random guess over the classes. This clearly indicates that the training distribution does not lead to sufficient generalization. Therefore to assess the efficacy of the dev set, we also try training on dev set and evaluation on the training dataset, which also does not yield any significant improvement in generalization.

Informed by these results, we try to estimate the gap in the distribution and also see how much of the two distributions are skewed from each other. One of the clear inconsistency between the two distributions is around the distances as measured by ℓ_2 norm between any given two feature vectors. We measure the nearest neighbour for all the points in the dev dataset and with respect to the points in the training dataset. We find a significant number of all the closest points in training and dev set to be having a different class label. Then we perform the nearest neighbour based on same label and train

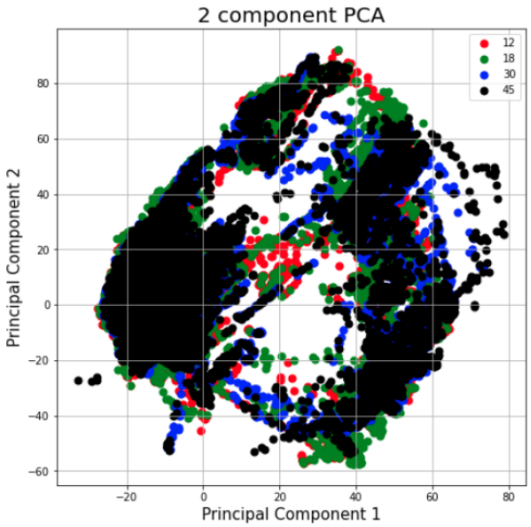

Fig. 2: PCA Visualization of MITRE dataset

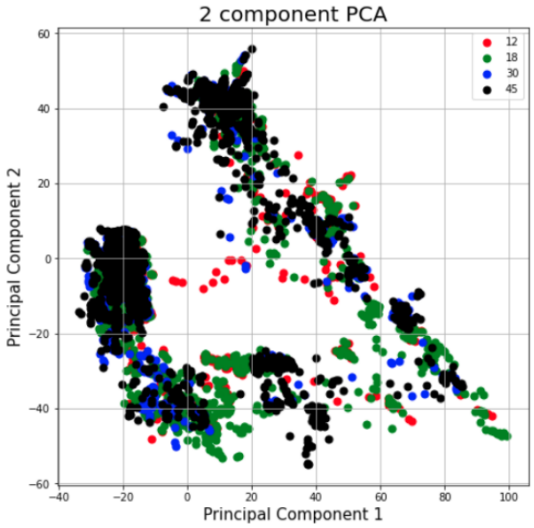

Fig. 3: PCA Visualization of NIST Development dataset

on the resulting training dataset with only 2 closest training points for each and every point in the dev set. The performance on this new training subset is not significantly different than the full training dataset. Furthermore, we measure the inter-class distances between the two distributions and find that nearest-neighbor between two datasets have an average of ℓ_2 distance of 24 while the neighboring points' classes are not same, however, if we find the closest ℓ_2 distance between two points with same class between training and dev set, then the average ℓ_2 distance is somewhere between 200. This supports our previously made argument about the data discrepency and distributions being not similar enough to capture any relevant and generalizable information. However, a more thorough analysis would be needed to confirm this argument in a statistical manner due to other possibilities like existence of highly non-linear manifolds which could fit both the distributions sufficiently.

VI. Conclusion and Future Work

In this work we present our approach for solving the TC4TL challenge and corresponding results. We also report our findings and analysis of the task as well as dataset. The task was marked by several challenges due to the noise in the data distribution and poor transferability of training data over the validation data. Therefore, we believe a proper physics based model which could capture appropriate invariances will be a good step towards solving the task. We also consider interpretable modeling and extensive breakdown of different sensor based data as part of the future work.

References

[1] Leo Breiman. Random forests, 2001.
[2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system, 2016.
[3] Corinna Cortes and Vladimir Vapnik. Support-vector networks, 1995.
[4] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theor.*, 13(1):21–27, September 2006.
[5] Gary F. Hatke1, Monica Montanari, Swaroop Appadwedula, Michael Wentz, John Meklenburg, Louise Ivers, Jennifer Watson, and Paul Fiore. Using bluetooth low energy (ble) signal strength estimation to facilitate contacttracing forcovid-19, 2020.
[6] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.
[7] Douglas J. Leith and Stephen Farrell. Coronavirus contact tracing: Evaluating the potential of using bluetooth received signal strength for proximity detection, 2020.
[8] Douglas J. Leith and Stephen Farrell. Measurement-based evaluation of google/apple exposure notification api for proximity detection in a commuter bus, 2020.
[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.
[10] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2017.