Improving Precision GNSS Positioning and Navigation Accuracy on Smartphones using Machine Learning

Improving Precision GNSS Positioning and Navigation Accuracy on Smartphones using Machine Learning

Akpojoto Siemuri, Kannan Selvan, Heidi Kuusniemi, Petri Välisuo, Mohammed S. Elmusrati
School of Technology and Innovations, University of Vaasa, Finland

BIOGRAPHY

Akpojoto Siemuri received his B.Sc.(tech) degree in electrical and computer engineering from Federal University of Technology Minna, Nigeria in 2010, the M.Sc.(tech) degree in wireless industrial automation, and a minor study in industrial management from the University of Vaasa, Finland in 2019. He is currently pursuing a Ph.D. degree in automation technology at the University of Vaasa. From 2018 to 2019, he was a Research Assistant in the Smart Energy Systems Research Platform (SESP) Project at the University of Vaasa, Finland. He is currently a Project Researcher in Digital Economy Research Platform, University of Vaasa. His research interest includes machine learning, GNSS technologies, smart devices, embedded systems, communication systems, and game theory.

Kannan Selvan received his B.Sc.(tech) degree in Electronics and Communication Engineering from Anna University, India in 2012, the M.Sc.(tech) degree in Communications and Systems Engineering from the University of Vaasa, Finland in 2020. From 2018 to 2020, he was a Research Assistant in the Digital Economy Research Platform at the University of Vaasa, Finland. He is currently a Project Researcher in Digital Economy Research Platform, University of Vaasa. His research interest includes GNSS technologies, satellite-data analysis, machine learning, satellite communication, smart devices and embedded Systems.

Dr. Heidi Kuusniemi is a professor in computer science and director of Digital Economy at the University of Vaasa in Finland. She is also a part-time research professor in satellite navigation at the Finnish Geospatial Research Institute. She has a M.Sc. (Tech.) degree (with distinction) from 2002 and a D.Sc. (Tech.) degree from 2005 in information technology, respectively, from Tampere University of Technology, Finland. She serves as a member of the council of natural sciences and technology at the Academy of Finland and was a member of the scientific advisory committee for GNSS (GSAC) at ESA. Her technical expertise and interests include GNSS reliability and resilience, estimation and data fusion, mobile precision positioning, indoor localization and PNT in new space.

Petri Välisuo is currently working as an Associate Professor (tenure track), sustainable automation, in the School of Technology and Innovation Mangement of University of Vaasa, Finland. He received M.Sc.(tech) degree in computer science from the Tampere University of Technology, Finland, and D.Sc.(tech) degree in automation technology from University of Vaasa, in years 1996 and 2011 respectively. He has authored and co-authored 27 peer reviewed and more than 10 other scientific publications. His research interests cover machine learning, IoT, positioning methods and other technologies relevant to industrial automation. He has been working 10 years in telecommunication industry before research career in the University of Vaasa.

Prof. Mohammed S. Elmusrati received his B.Sc. (with honors) and M.Sc. (with high honors) degrees in electrical and electronic engineering, University of Benghazi, Libya, in 1991 and 1995, respectively, and the Licentiate of Science in technology (with distinction) and the Doctor of Science in Technology (D.Sc.) degrees in automation and control engineering from Aalto University Finland, in 2002 and 2004, respectively. Currently, he is Full Professor and Head of the Digitalization Unit at the School of Technology and Innovations – University of Vaasa, Finland. His research interest includes wireless communications, artificial intelligence, machine learning, biotechnology, big data analysis, stochastic systems, and game theory. Elmusrati has published more than 130 papers, books, and book chapters. Prof. Elmusrati is an active member in different scientific societies such as Senior Member at IEEE, Member at Society of Industrial and Applied Mathematics (SIAM), and Member at Finnish Automation Society.

ABSTRACT

In this work, we developed a precision positioning algorithm for multi-constellation dual-frequency global navigation satellite systems (GNSS) receivers that predicts the latitude and longitude from smartphone GNSS data. Estimation for all epochs that have at least four valid GNSS observations is generated. Receivers (especially low-cost receivers) often have limited channels and computational resources, therefore, the complexity of the algorithm used in them needs to be kept low. The datasets and results in this paper are based on the data provided by Google under the session "High Precision GNSS Positioning on Smartphones Challenge" in the Institute of Navigation (ION GNSS+ 2021) conference.

We began by exploring and analysing the raw GNSS data which includes the training dataset and its ground truth and the test dataset without the ground truth. This analysis gave insight into the nature and correlation of the dataset and helped shape the algorithm that was proposed for the accuracy improvement problem. The design of the algorithm was done using data science techniques to compute the average of the predictions of several devices data in the same collection (training dataset baseline coordinates and their ground truth) and then the data was used to train a few selected machine learning algorithms namely, Linear Regression (LR), Bayesian Ridge (BR) and Neural Network (NN) to predict the offset of the test data baseline coordinates from the expected ground-truth (which was not provided).

A simple weighted average (SWA) which combines all the previous three ML technique was also implemented. The results showed improvement in the position accuracy with the simple weighted average (SWA) method having the best accuracy followed by Bayesian Ridge (BR), Linear Regression (LR), and then Neural Network (NN) respectively.

Keywords - Machine Learning; Linear Regression; Bayesian Ridge; Neural Networks; Simple Weighted Average; Global Navigation Satellite Systems; Smartphones; Mean Prediction

I. INTRODUCTION

1. Global Navigation Satellite Systems (GNSS)

Global Navigation Satellite Systems (GNSS) is a term used to collectively describe satellite-based positioning and timing systems. In the past decades, we have seen significant improvements in the performance of the Global Navigation Satellite System (GNSS) technology. The time it took to get a first accurate position has been improved from minutes to under thirty seconds. This improvement has been seen in the early 2000s. The latter half of the decade was the time when receiver sensitivity improvement was significant, that is, from -130 dBm to -167 dBm. The number of functional positioning satellite constellations has seen an increase in 2015. In 2000, there was only one global constellation (the USA's GPS) and as of 2015, there are four (GPS, GLONASS, BeiDou, and Galileo). This has led to the development of multi-constellation GNSS receivers. Multi-band GNSS became more affordable as of 2018 due to the modernization of the satellite signals. These advances are the foundations for the next big steps to be taken in GNSS such as achieving millimeter- or centimeter-level accuracy.

The benefit of multi-constellation GNSS is the availability of more visible satellites that can be used to improve user positioning performance [1]. This is very useful in challenging environments where GNSS signals could be partially or totally blocked and can suffer from multipath reflections, for example, in urban areas or places with dense foliage. However, most receivers have limited tracking receiver channels therefore, it may be not possible, or desirable, to use all satellites in view for positioning. Other reasons for this include power consumption, multipath or non-line-of-Sight (NLOS) signal measurements, and other issues [2]. This means a subset of satellites from the available satellites in view needs to be selected.

Randomly selecting satellites can result in a poor satellite geometry, which can increase the geometry dilution of precision (GDOP), and consequently induce accuracy degradation. It has been common to use a GDOP or weighted GDOP (WGDOP) as a satellite selection criterion for positioning [1], [3], [4], and [5] etc. This considers only the geometric effects, however, there are other factors that cause positioning error, such as ionospheric delay, tropospheric delay, and multipath propagation causing ranging measurement errors.

The Global Navigation Satellite System (GNSS) provides raw signals, which the GNSS chipset in smartphones uses to compute a position. The Google challenge for which this paper is written provided the data collected from some Android smartphones. These data were released to be used in computing location down to decimeter or even centimeter resolution, if possible. Google also provided access to precise ground truth (for training dataset only), raw GNSS measurements, and assistance data from nearby GNSS stations, in order to train and test machine learning algorithms for providing improved positioning performance.

2. Machine Learning Algorithms

Soft computing techniques are well suited for real-life problems that use methods to extract useful information from complex and intractable problems in less time. They are tolerant to data that is imprecise, partially incorrect,or uncertain. One of the important components of soft computing techniques includes machine learning (ML) techniques. The machine learning (ML) techniques have been used in the literature in order to predict models for estimating the GNSS performance. For example, [6] have employed Gaussian Process Regression (GPR) model while comparing it to the Autoregressive moving average model, and the Artificial neural network model, in order to predict ionospheric time delays using Global Navigation satellite system observations. There are several advantages of making use of these ML techniques for estimation, prediction, and enhancement of GNSS performance.

ML is a very powerful tool in processing time-series data, as it can be applied in learning time-dependent patterns across multiple models. It has been found useful in discovery hidden and unknown patterns and information in GNSS data. The GNSS provides nearly limitless and affordable quantity of data, also, data storage and the growth of less expensive and more powerful processing capabilities has propelled the growth of ML. When compared to statistical methods, ML techniques enable us to identify tricky dependencies in data for which exploratory analysis has not enabled the proper determination of the shape of the underlying model [7]. The aim of using ML is not to generate an explicit formula for the distribution of the data; however, it is used to train an algorithm to detect the relationships between the features of a data set, directly from the data. This learning methods makes it possible to avoid making assumptions as seen in many statistical methodologies.

II. MATERIALS, METHODS, AND MODELS

In our work, we used the average of the predictions of several phones in the same data collection using baseline location data provided by Google. These baseline location data were computed using a simple least square solution. Next, we apply three machine learning (ML) algorithms Linear Regression (LR), Bayesian Ridge (BR) and Neural Network (NN) respectively to get the final prediction. These ML algorithms were implemented separately for comparison of their accuracy.

The steps or approach taken are as follows:

1. Data Analysis and Preparation
2. Reject Outlier and Apply Kalman Filter
3. Perform Phones Mean Prediction
4. Apply Linear Regression (LR), Bayesian Ridge (BR) and Neural Network (NN) respectively to get the final prediction for Train Dataset
5. Evaluate Train Score
6. Apply Linear Regression (LR), Bayesian Ridge (BR) and Neural Network (NN) respectively to test dataset to get the final prediction in latitude and longitude.
7. Perform a simple weighted average (SWA) on the results of the BR, LR, and NN models.

1. Data Analysis and Preparation

Google releases a total of 121 traces in the challenge, whose collection process is described in a paper published in the proceedings of ION GNSS+ 2020 [8]. The data were released twice, first the training data and next the test data. Most of the data were collected in US Bay area highways with open sky. A small part of some traces show sparse buildings, overpasses, or trees.

a). Training datasets

The training datasets consisted of 73 traces. The data included GnssLogger files, RINEX observation files, ground truth files, and derived files (GNSS intermediate values derived from raw GNSS measurements, provided for convenience).

The derived values contains values like:

1. The ionospheric delay in meters, estimated with the Klobuchar model.
2. The tropospheric delay in meters, estimated with the EGNOS model by Nigel Penna, Alan Dodson and W. Chen (2001).
3. Raw pseudorange uncertainty in meters.
4. GNSS constellation type. An integer number, representing the available GNSS constellations.
5. The satellite ID.
6. The GNSS signal type which is a combination of the constellation name (forexample, GPS) and the frequency band (for example, L1).
7. The signal transmission time received by the chipset, which is the numbers of nanoseconds since the GPS epoch.

8. The satellite position in meters ([x/y/z]SatPosM) in an ECEF coordinate frame at best estimate of "true signal transmission time" defined as ttx = receivedSvTimeInGpsNanos - satClkBiasNanos (defined below). They are computed with the satellite broadcast ephemeris, and have 1-meter error with respect to the true satellite position.
9. The satellite velocity (meters per second - [x/y/z]SatVelMps) in an ECEF coordinate frame at the signal transmission time (receivedSvTimeInGpsNanos). They are computed with the satellite broadcast ephemeris, with this algorithm.
10. The satellite time correction combined with hardware delay in meters at the signal transmission time (receivedSvTime-InGpsNanos) called satellite clock bias (satClkBiasM). Its time equivalent is termed as satClkBiasNanos.
11. The satellite clock drift in meters per second (satClkDriftMps) at the signal transmission time (receivedSvTimeInGp-sNanos). It equals the difference of the satellite clock biases at t+0.5s and t-0.5s.
12. Raw pseudorange in meters (rawPrM). It is the product between the speed of light and the time difference from the signal transmission time (receivedSvTimeInGpsNanos) to the signal arrival time (Raw::TimeNanos - Raw::FullBiasNanos - Raw::BiasNanos).

b). Test datasets

In the released test datasets, there are 48 more traces, which include the same types of data and follow the same convention as the training dataset, except that the ground truth files are not provided. The results of test datasets is used as the prediction of the expected ground truth. The results are evaluated by Google using the unreleased ground truth via kaggle (www.kaggle.com). In both the training and test datasets, the derived values can be used to compute a corrected pseudorange which is a closer approximation to the geometric range from the phone to the satellite.

This can be done using the formula in equation 1:

$$\rho_c = \rho + c\Delta t_{sat} - B_{isr} - c\Delta t_{in} - c\Delta t_{tr} \tag{1}$$

The baseline locations are then computed using corrected pseudorange and the satellite positions, using a standard Weighted Least Squares (WLS) solver, with the phone's position (x, y, z), clock bias (t), and Bisr (The Inter-Signal Range Bias (ISRB) in meters from a non-GPS-L1 signal to GPS-L1 signals) for each unique signal type as states for each epoch. The GnssLogger files in both the training and test datasets include: receivedSvTime (equivalent: pseudorange), pseudorange-rate (Doppler), Accumulated Delta Range (carrier phase), for all visible GNSS satellites: GPS, GLO, GAL, BDS, QZS at 1 Hz. Data also includes uncalibrated accelerometer, gyroscope, and magnetic field readings. The baseline locations (estimated latitude and longitude coordinate for both the train/test datasets) were also released. This were generated using a simple approach, weighted least square (WLS). These baseline locations were used as the primary data for the algorithm implemented in this paper along side the training dataset (also released as latitude and longitude coordinate).

The next section described the next steps taken in the data pre-processing before the ML algorithms were implemented.

2. Remove Outliers and Apply Kalman Filter

An outlier in an observation set is a point that is unlike other points in an observation set. It is distinct, rare and does not fit to the other observations set in some way. It can be generally defined as samples that are exceptionally far from the mainstream of the data.

The causes of outliers could include:

1. Measurement or input error
2. Data corruption
3. A true outlier observation

In general, outliers cannot be precisely defined and identified because of the specifics of each dataset. Rather, raw observations must be interpreted and a decision made whether a value is an outlier or not. Therefore, great care should be taken not to hastily remove or change values, especially for small sample size data.

In data pre-processing and cleanup, identifying outliers and bad data is probably one of the most difficult task and it takes time to get it right. Even with a very deep understanding of statistics and how outliers might affect your data, it is always an area to address with caution. This is because it does not mean that the values identified will be outliers and should be removed. A way to approach this may be to plot the identified outlier values against the context of non-outlier values to visualize the possible systematic relationship or pattern to the outliers. If there are, this may mean that they are not outliers and can therefore, be explained. Furthermore, this can help the outliers themselves to be more systematically identified.

The removal of outliers was performed using a function adapted from [9]. This function computes the difference in distance between the previous coordinate and the next coordinate point. It used this to generate a new column that holds the distance between the previous and next coordinate points and a threshold value is set above which values are regarded as outliers. The threshold values used in our case was set at 50, based on the analyses of the results from the observation of the distance between the coordinates.

a). Kalman Filter Implementation

Kalman filter is applied to the results to smooth the output in order to produce more accurate values. The kalman filter function used is adapted from [10]. The applied Kalman Filter is aimed at improving the baseline slightly.

3. Mean Prediction of the Phone Data

The average of the predictions of several phones in the same data collection from the data provided by Google for the "*High Precision GNSS Positioning on Smartphones Challenge (sponsored by Google)*" was computed.

This was done by generating an interpolated latitude, longitude values for different phone times in the same collection. Linear interpolation was used to achieve this. The generated records to be interpolated were derived using a combination of the time of collection of the data and the phone used. An open source function from https://www.kaggle.com/t88take/gsdc-phones-mean-prediction was used to achieve this. Afterwards, the latitude and longitude predictions were made based on the average of the baseline location predictions of phones in the same collection provided by Google. The open source function used to achieve this is also from https://www.kaggle.com/t88take/gsdc-phones-mean-prediction.

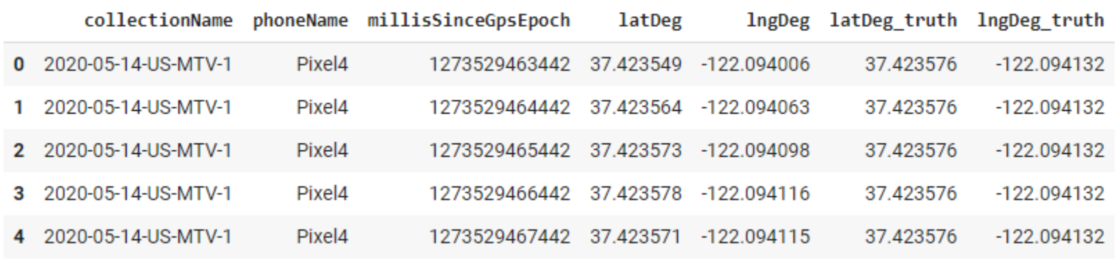
The next step taken was to apply the results from the phone mean method to some selected ML algorithm as described in the next section.

III. BASELINE APPROACH - MACHINE LEARNING IMPLEMENTATION

The goal of this study is to use machine learning techniques to estimate the position produced by the use of a certain smartphone taking into account that currently, mobile phones only offer 3-5 meters of positioning accuracy. The approach used in this study is the baseline approach. We used the baseline location values for the train and test datasets provided by Google. These baseline location values were computed using a simple least square method. The ground truth data were also provided by Google only for the training datasets. These ground truth data were the true latitude and longitude coordinates of the data collected using high end GNSS receivers.

1. Aligning the Ground Truths with Training Data Inputs

The ground truth data were extracted and aligned with training data inputs. This was done in order to have the latitude and longitude coordinates of the training data and the ground truth data in one dataset to be used in training the ML algorithm. Fig. 1 shows a sample of the aligned data.



	collectionName	phoneName	millisSinceGpsEpoch	latDeg	lngDeg	latDeg_truth	lngDeg_truth
0	2020-05-14-US-MTV-1	Pixel4	1273529463442	37.423549	-122.094006	37.423576	-122.094132
1	2020-05-14-US-MTV-1	Pixel4	1273529464442	37.423564	-122.094063	37.423576	-122.094132
2	2020-05-14-US-MTV-1	Pixel4	1273529465442	37.423573	-122.094098	37.423576	-122.094132
3	2020-05-14-US-MTV-1	Pixel4	1273529466442	37.423578	-122.094116	37.423576	-122.094132
4	2020-05-14-US-MTV-1	Pixel4	1273529467442	37.423571	-122.094115	37.423576	-122.094132

Figure 1: Extracting ground truths and aligning with train inputs.

Plotting the training data coordinates against the ground truth can be seen in Fig. 2.

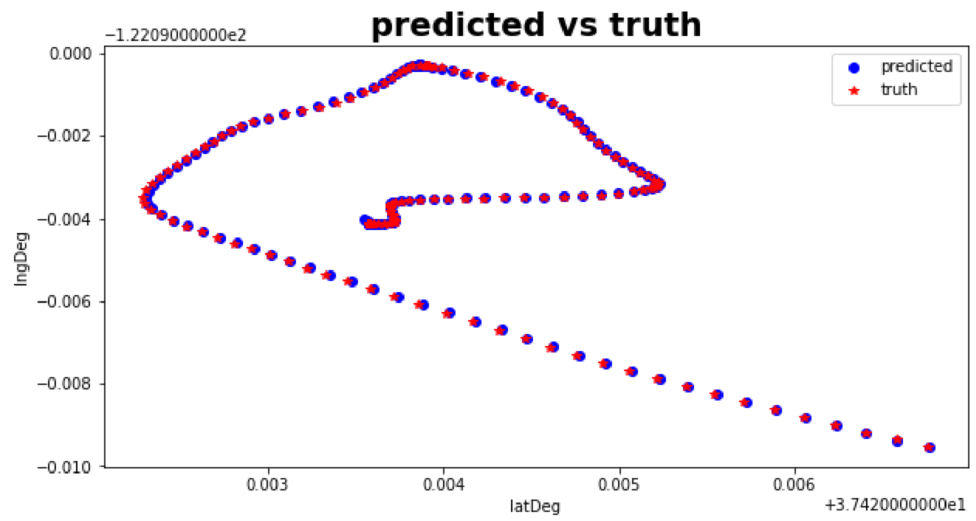


Figure 2: Plotting predicted vs truth coordinates.

2. One-hot Encoding

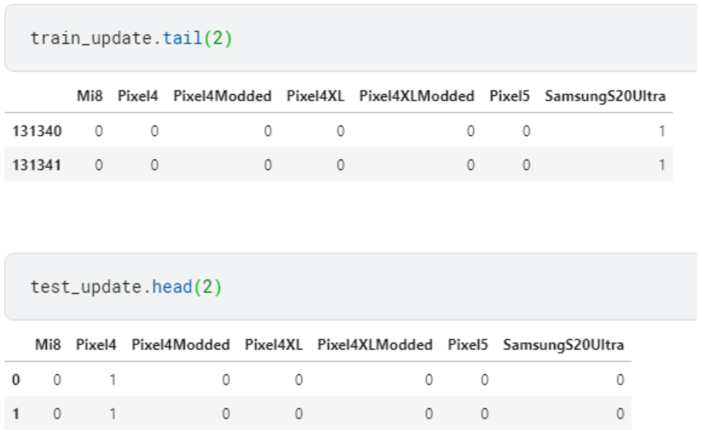
Usually, in machine learning (ML) it is recommend or required that you prepare your data in specific ways before fitting a machine learning model. Categorical data are variables containing label values instead of numeric values and the number of possible values is often limited to a fixed set. There are often referred to as nominal. A nominal category or a nominal group is a group of objects or data collectively grouped together on the basis of a particular characteristic or a qualitative property [11]. Many ML algorithms cannot directly make use of label data. They require all input variables and output variables to be numeric. This means that categorical data must be converted to a numerical form. For a categorical variable being an output variable, you may want to convert predictions by the model back into a categorical form. This is done in order to use them in some application.

A good way to achieve this is to use a one-hot encoding on categorical data. One-hot encoding refers to the process of splitting the column which contains numerical categorical data to many columns depending on the number of categories present in that column. This will result to each column containing “0” or “1” corresponding to which column it has been placed. However, One hot encoder only takes numerical categorical values, therefore, any value of string type must be label encoded (also called integer encoded) before one-hot encoded. For example, label encoding a gender column could be done as “Female” is 1, “Male” is 2.

The one-hot encoder then gives binary variables often called “dummy variables” in the form:

Female,	Male
1	0
0	1

The one-hot encoding was applied to the "phoneName" variable (name of the phone used to collect the data) of the train and test data. The train and test columns were then aligned and in Fig 3, we can see the output of the one-hot encoding. The train and test data after one-hot encoding is illustrated in Fig 4. The next step after one-hot encoding the data is the implementation of the ML algorithms.



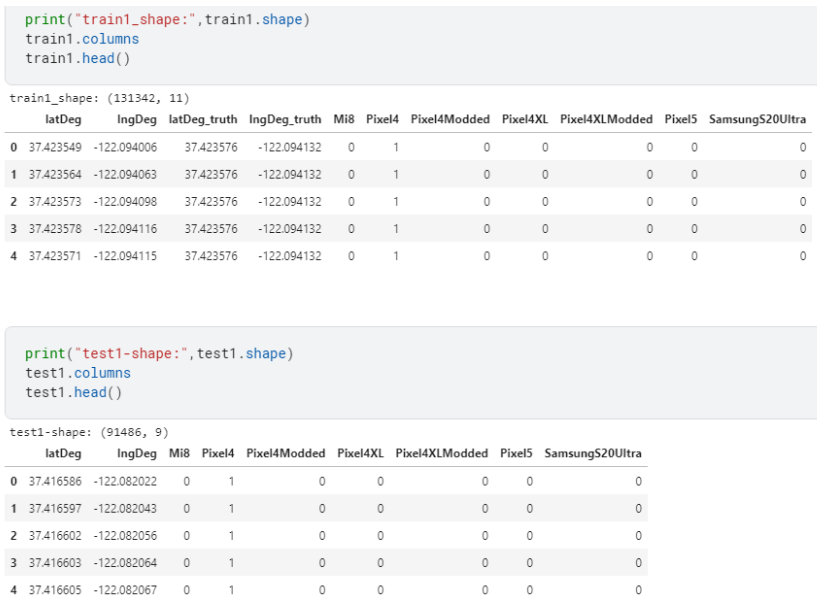
```
train_update.tail(2)
```

	Mi8	Pixel4	Pixel4Modded	Pixel4XL	Pixel4XLModded	Pixel5	SamsungS20Ultra
131340	0	0	0	0	0	0	1
131341	0	0	0	0	0	0	1

```
test_update.head(2)
```

	Mi8	Pixel4	Pixel4Modded	Pixel4XL	Pixel4XLModded	Pixel5	SamsungS20Ultra
0	0	1	0	0	0	0	0
1	0	1	0	0	0	0	0

Figure 3: Performing one-hot encoding.



```
print("train1_shape:",train1.shape)
train1.columns
train1.head()
```

```
train1_shape: (131342, 11)
```

	latDeg	lngDeg	latDeg_truth	lngDeg_truth	Mi8	Pixel4	Pixel4Modded	Pixel4XL	Pixel4XLModded	Pixel5	SamsungS20Ultra
0	37.423549	-122.094006	37.423576	-122.094132	0	1	0	0	0	0	0
1	37.423564	-122.094063	37.423576	-122.094132	0	1	0	0	0	0	0
2	37.423573	-122.094098	37.423576	-122.094132	0	1	0	0	0	0	0
3	37.423578	-122.094116	37.423576	-122.094132	0	1	0	0	0	0	0
4	37.423571	-122.094115	37.423576	-122.094132	0	1	0	0	0	0	0

```
print("test1-shape:",test1.shape)
test1.columns
test1.head()
```

```
test1-shape: (91486, 9)
```

	latDeg	lngDeg	Mi8	Pixel4	Pixel4Modded	Pixel4XL	Pixel4XLModded	Pixel5	SamsungS20Ultra
0	37.416586	-122.082022	0	1	0	0	0	0	0
1	37.416597	-122.082043	0	1	0	0	0	0	0
2	37.416602	-122.082056	0	1	0	0	0	0	0
3	37.416603	-122.082064	0	1	0	0	0	0	0
4	37.416605	-122.082067	0	1	0	0	0	0	0

Figure 4: Train and test data after one-hot encoding.

3. Implementing ML Algorithms

Several models were used to attempt to capture the relationship between datasets and overall positioning performance. These algorithms includes Bayesian Ridge (BR), logistical regression (LR), and a neural networks (NN). For the BR and LR approach, the one-hot encoding was used to prepare the data. However, for the NN approach, the one-hot was not applied. Each of the models were used and evaluated separately.

Preliminary tests, implementing all regressor algorithms for a quick performance analysis, showed that these were the most promising methods.

a). BR and LR Models

Each model was tuned using the one-hot encoded data, The input and output for the model was created from the latitude and longitude columns of the data. The data is then split into training and validation datasets using the library from sklearn model selection.

The BR model is now fit using the data. Similarly, the **the LR model** is fit using the data. The one-hot encoding is then reversed for each phone data (one-hot decoding). The validation data is used to evaluate the models as seen in Fig. 5 and Fig. 6 in the results section which shows the evaluation score for each phone (50th, 95th and average percentile) for the BR and LR models respectively. The model is then applied to the complete test data set to predict the latitude and longitude coordinates. The evaluation of these results is done in kaggle.com as part of the Google competition requirements. These models are adapted from [12].

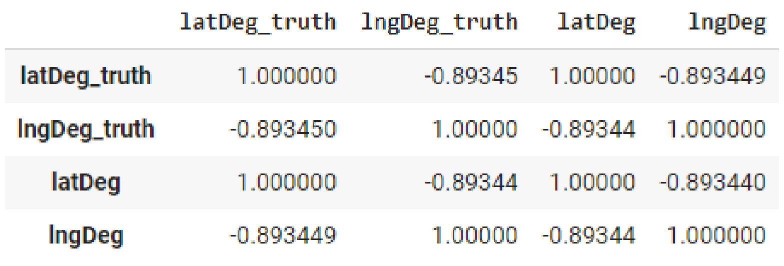
b). The NN model

The **NN model** used here is from the of sklearn python library. The model is adapted from [13].

Feature creation

The features generated are the aggregated features used for training the model. These features includes previous latitude and longitude, the corrected pseudoranges (correctedPrM) for each satellite from derived files shared by Google, and the latitude and longitude difference of the training data from the ground truth. These features are generated for both the train and test datasets.

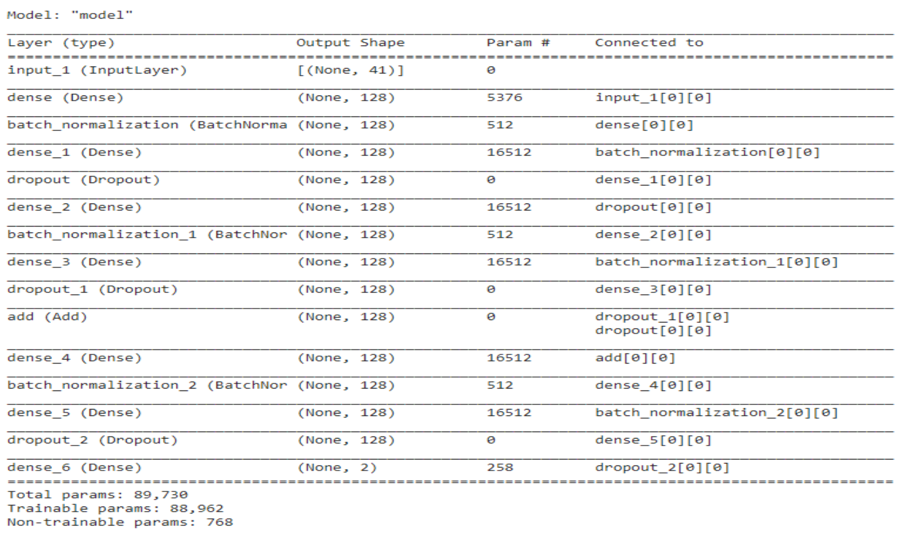
The correlation between train data baseline coordinates and ground truth coordinates are seen in Fig. 5 below.



	latDeg_truth	lngDeg_truth	latDeg	lngDeg
latDeg_truth	1.000000	-0.89345	1.00000	-0.893449
lngDeg_truth	-0.893450	1.00000	-0.89344	1.000000
latDeg	1.000000	-0.89344	1.00000	-0.893440
lngDeg	-0.893449	1.00000	-0.89344	1.000000

Figure 5: Checking the correlation between current and truth coordinates.

Training the NN Model The NN model training is achieved using a skip connection in Keras on Tensor Processing Unit (TPU). TPU is highly-optimised for large batches and CNNs and has the highest training throughput. It is a custom build ASIC used to accelerate TensorFlow projects. The model summary is shown in Fig. 6 below.



```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 41)]         0
__________________________________________________________________________________________________
dense (Dense)                   (None, 128)          5376        input_1[0][0]
__________________________________________________________________________________________________
batch_normalization (BatchNorma (None, 128)          512         dense[0][0]
__________________________________________________________________________________________________
dense_1 (Dense)                 (None, 128)          16512       batch_normalization[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 128)          0           dense_1[0][0]
__________________________________________________________________________________________________
dense_2 (Dense)                 (None, 128)          16512       dropout[0][0]
__________________________________________________________________________________________________
batch_normalization_1 (BatchNor (None, 128)          512         dense_2[0][0]
__________________________________________________________________________________________________
dense_3 (Dense)                 (None, 128)          16512       batch_normalization_1[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 128)          0           dense_3[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 128)          0           dropout_1[0][0]
                                                                 dropout[0][0]
__________________________________________________________________________________________________
dense_4 (Dense)                 (None, 128)          16512       add[0][0]
__________________________________________________________________________________________________
batch_normalization_2 (BatchNor (None, 128)          512         dense_4[0][0]
__________________________________________________________________________________________________
dense_5 (Dense)                 (None, 128)          16512       batch_normalization_2[0][0]
__________________________________________________________________________________________________
dropout_2 (Dropout)             (None, 128)          0           dense_5[0][0]
__________________________________________________________________________________________________
dense_6 (Dense)                 (None, 2)            258         dropout_2[0][0]
==================================================================================================
Total params: 89,730
Trainable params: 88,962
Non-trainable params: 768
```

Figure 6: NN model summary.

4. Implementing Weighted Average of BR, LR and NN Models

The results from the three ML algorithms (BR, LR and NN) were integrated using a simple weighted average (SWA) to check on improving the results. A simple weight was used. First, equal weights were assigned to the model (i.e., 0.3333), the results showed slight improvement compared to the individual ML models. Secondly, it was noticed that when different weights were used, an improvement was seen compared to when the same weights were used.

The different weights were assigned based on the accuracy of the technique. The weight of a certain ML method is the accuracy of that technique divided by the total accuracy (sum of all the models accuracy). Hence, when the average accuracy of a certain ML technique increases, its summation weight would increase. However, the total weight sum is 1.

The results are presented and discussed in table 1 in the next section.

IV. DISCUSSION OF RESULTS

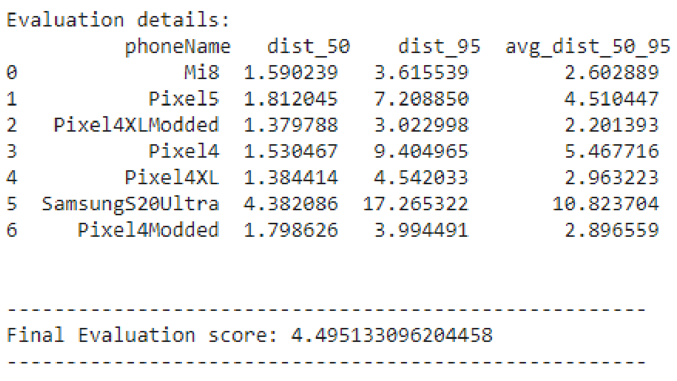
This section presents the results and the evaluations of the results.

1. Evaluating the Models

The evaluation of the model is done for all the implemented ML algorithms.

For the BR and LR models:
The evaluation of the model is done on the validation datasets. The training data was split into train data and validation data. The Fig. 7 and Fig. 8 shows the evaluation of the validation data for the BR and LR models respectively.

For the BR model:

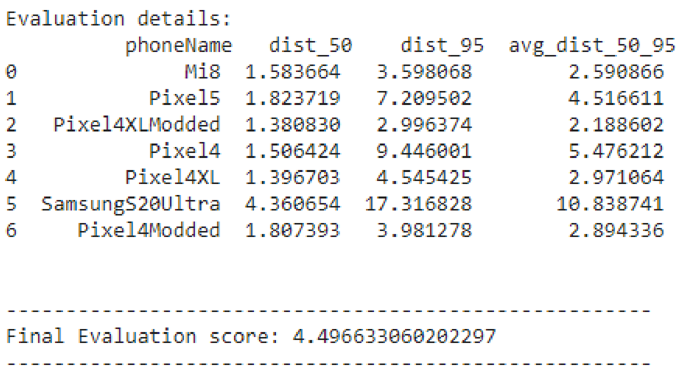


```
Evaluation details:
         phoneName   dist_50    dist_95  avg_dist_50_95
0              Mi8  1.590239   3.615539        2.602889
1           Pixel5  1.812045   7.208850        4.510447
2   Pixel4XLModded  1.379788   3.022998        2.201393
3           Pixel4  1.530467   9.404965        5.467716
4         Pixel4XL  1.384414   4.542033        2.963223
5  SamsungS20Ultra  4.382086  17.265322       10.823704
6     Pixel4Modded  1.798626   3.994491        2.896559


------------------------------------------------------
Final Evaluation score: 4.495133096204458
------------------------------------------------------
```

Figure 7: Phone level evaluation score for Bayesian Ridge.

For the LR model:



```
Evaluation details:
         phoneName   dist_50    dist_95  avg_dist_50_95
0              Mi8  1.583664   3.598068        2.590866
1           Pixel5  1.823719   7.209502        4.516611
2   Pixel4XLModded  1.380830   2.996374        2.188602
3           Pixel4  1.506424   9.446001        5.476212
4         Pixel4XL  1.396703   4.545425        2.971064
5  SamsungS20Ultra  4.360654  17.316828       10.838741
6     Pixel4Modded  1.807393   3.981278        2.894336


------------------------------------------------------
Final Evaluation score: 4.496633060202297
------------------------------------------------------
```

Figure 8: Phone level evaluation score for Linear Regression.

The Fig. 7 and Fig. 8 show the evaluation score for each phone in the 50th, 95th and average percentile. These are the computed haversine distance estimation for BR and LR model respectively. The haversine distance estimation is the calculated great circle distance between two points on the earth.

A function was adopted from [14] to get the prediction and ground truth distance estimation (in meters). The inputs are array-like and specified in decimal degrees. This is therefore, the distance between the training/validation data baseline coordinates and the ground truth baseline coordinates.

We can see that the BR model has a better performance than LR model both at the phone level (that is for each phone in most cases) and at the final evaluation score which is the mean of the average distance for the 50th and 95th percentile (avg-dist-50-95).

For the test data, since we do not yet have the ground truth, the pre-evaluation was done using the predicted baseline coordinates and the actual baseline coordinates in the original data.

In the LR model on test data:
The prediction for test data, haversine distance is (latitude/longitude)-
haversine distance (smoothed): [0.90276   4.05308457]

The 2D error is then sqrt(0.9028*pow(2) + 4.0531pow(2)) = 3.6592.

In the BR model on test data:
The prediction for test data, haversine distance is (latitude/longitude)-
haversine distance (smoothed): [0.91345946   4.05016769]

The 2D error is then sqrt(0.9135*pow(2) + 4.0501pow(2)) = 3.6998.

For the NN model: The evaluation of the model is done by calculating the great circle distance between baseline coordinates and the ground truth in the case of the training datasets. For the test data, since the ground truth was not released, the pre-evaluation was done using the predicted baseline coordinates and the actual baseline coordinates in the original data.

The prediction for training data, haversine distance is (latitude/longitude)-
haversine distance (smoothed): [2.14165623 9.83846538]

The prediction for test data, haversine distance is (latitude/longitude)-
haversine distance (smoothed): [0.91779853 5.35367849]

2. Visualize NN Model Training History

We created plots from the collected history data for visualization of NN Model Training History. The plots are from training the model:

1. A plot of model accuracy (Fig. 9).
2. A plot of model loss (Fig. 10).
3. A plot of model learning rate (Fig. 11).

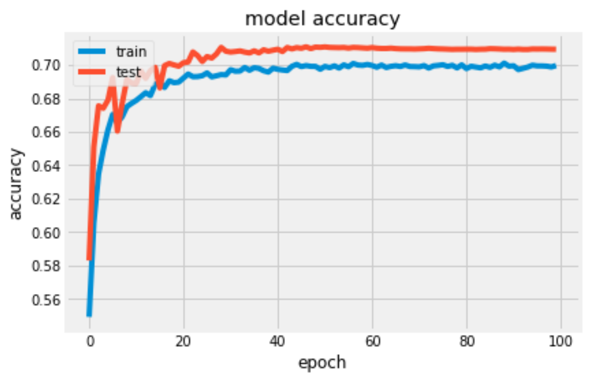


Figure 9: Plot of Model Accuracy on Train and Validation Datasets.

From Fig. 9, we can see that training the model a little more may not be effective as the trend for accuracy on both datasets has stopped rising but rather it moves horizontally for the last few epochs. We can also see that the training dataset has not yet been over-learned by the model, as both datasets show comparable skill.

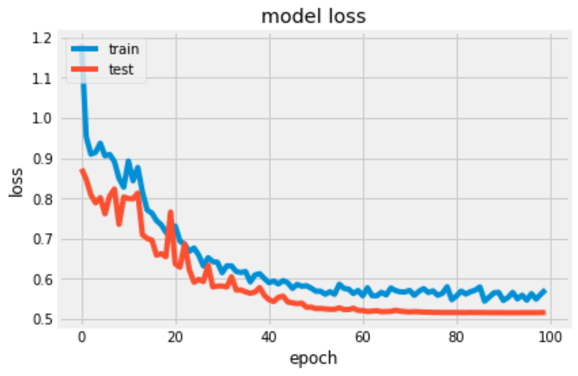


Figure 10: Plot of Model Loss on Training and Validation Datasets.

From Fig. 10, the model loss, we can see that the model performs comparably on both train and validation datasets (labeled as test). If these parallel plots start to show a consistent departure from each other, this could be a good indication to stop training at an earlier epoch.

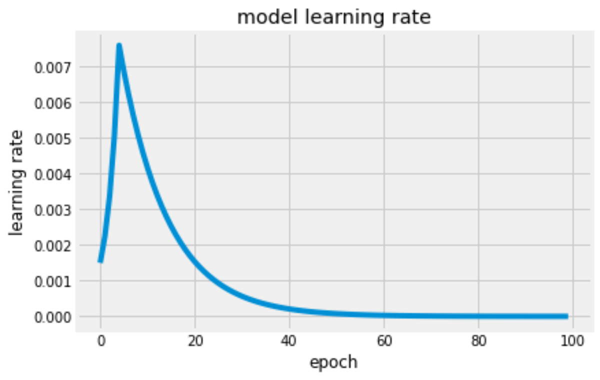


Figure 11: Plot of Model Learning Rate on the Datasets.

From Fig. 11, the model learning rate shows that most of the learning takes place at the first few epochs after which training the model may not lead to any improvement in the learning.

The history for the validation dataset is usually labeled as test by convention as it can be regarded as a test dataset for the model.

In all ML models, the exact evaluation of the models for the test data with the ground truth is done on kaggle.com as part of the google competition.

In Table 1, the results of the preliminary evaluation done on kaggle.com for the test data baseline prediction is presented for each ML model implemented (BR, LR, NN , SWA method).

Table 1: Results from evaluation of test data prediction on kaggle.com

Model List						
					SWA	
	WLS	Phone mean + BR	Phone mean + LR	Phone mean + NN	same weights	different weights
Accuracy in meters (m)	20.49	5.631	5.656	5.743	5.626	5.620

From table 1, we can see that the SWA performs better, followed by the BR, LR and NN model respectively. In the case of the SWA, when the average accuracy of a certain ML technique increases, its summation weight would increase, thereby, increasing the accuracy. This is seen in the accuracy improvement when different weights are used compared to using the same weights.

V. CONCLUSION AND FUTURE WORK

The data used in this paper was provided by Google under a competition hosted on kaggle.com for the ION GNSS+ 2021 conference. The Global Navigation Satellite System (GNSS) provides raw signals, which the GNSS chipset in smartphones uses to compute a position. The accuracy are usually not good enough and can create a “jumpy” experience for many applications relying on localization.

In this paper, we presented an algorithm that can improve the precision GNSS positioning and navigation accuracy on smartphones using a machine learning approach. First, we analyzed the data and pre-processed it for the model by identifying and removing outliers. Then we applied a "phone mean method" to the data in combination with some pre-selected ML models namely, bayesian ridge, linear regression, NN model and SWA method respectively to predict the latitude and longitude coordinates. The results show an improvement in the GNSS positioning and navigation accuracy when evaluated against the ground truth of the data collected using high end GNSS receivers for the same locations. The positioning accuracy obtained is in the range of 5.620 - 5.743 m (meters). This is an improvement to the WLS of 20.49 meters positioning accuracy.

In future publications (on-going research), we will develop a precision positioning algorithm for multi-constellation dual-frequency global navigation satellite systems (GNSS) receivers that would select a subset of satellites out of the tracked ones to achieve improved location accuracy with the help of machine learning. Receivers (especially low-cost receivers and smartphones) often have very limited channels and computational resources, therefore, the complexity of the algorithm needs to be kept low. We will use both the GDOP and measurement errors as criteria for satellite selection. From initial analysis, the trained models will effectively predict pseudorange and delta range residuals and also select the satellites making the most contribution to the desired GDOP value. The optimization of GDOP and the ability to filter out erroneous pseudorange and delta range measurements can lead to improved location accuracy. We will also work on integrating "outlier removal" to Kalman filters (KF) because if they are not removed, they would disturb linear KF heavily. Furthermore, we will first make use of the available raw GNSS data (including pseudorange rates, accumulated delta range - aka carrier phase, and base station data), and IMU data to generate a much better position (than WLS used in this study) and then the results will be further improved with ML.

ACKNOWLEDGEMENTS

The authors would like to acknowledge Google for providing the datasets used in this study and give credits to some kaggle.com notebooks adapted in this study as reference in this paper.

REFERENCES

[1] P. Huang, C. Rizos, and C. Roberts, “Machine learning algorithm to forecast ionospheric time delays using global navigation satellite system observations,” *GPS Solutions*, vol. 22, pp. 221–231, 2018. [Online]. Available: https://doi.org/10.1007/s10291-018-0776-0

[2] N. I. Ziedan, “Multipath and nlos signals identification and satellite selection algorithms for multi-constellation receivers,” *Proceedings of the 29th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2016)*, pp. 521–533, 2018. [Online]. Available: https://doi.org/10.33012/2016.14844

[3] A. M. El-naggar, “An alternative methodology for the mathematical treatment of gps positioning,” *Alexandria Engineering Journal*, vol. 50, no. 4, pp. 359–366, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1110016811000470

[4] E. Wang, C. Jia, S. Feng, G. Tong, H. He, P. Qu, Y. Bie, C. Wang, and Y. Jiang, “A new satellite selection algorithm for a multi-constellation gnss receiver,” *in Proceedings of the 31st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2018), Miami, Florida*, pp. 3802–3811, 2018. [Online]. Available: https://doi.org/10.33012/2018.15993

[5] P. F. Swaszek, R. J. Hartnett, K. C. Seals, and R. M. Swaszek, “Tighter gdop bounds and their use in satellite subset selection,” *in Proceedings of the 32nd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2019)*, pp. 637–649, 2019. [Online]. Available: https://doi.org/10.33012/2019.16869

[6] L. Mallika I, D. V. Ratnam, S. Raman, and G. Sivavaraprasad, “Machine learning algorithm to forecast ionospheric time delays using global navigation satellite system observations,” *Acta Astronautica*, vol. 173, pp. 221–231, 2020. [Online].

Available: https://www.sciencedirect.com/science/article/pii/S0094576520302630

[7] A. Siemuri, H. Kuusniemi, M. S. Elmusrati, P. Välisuo, and A. Shamsuzzoha, "Machine learning utilization in gnss—use cases, challenges and future applications," in *2021 International Conference on Localization and GNSS (ICL-GNSS)*, 2021, pp. 1–6.

[8] G. M. Fu, M. Khider, and F. van Diggelen, "Android raw gnss measurement datasets for precise positioning," in *Proceedings of the 33rd International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+2020)*, September 2020, pp. 1925–1937.

[9] @t88take. (2021) Gsdc phones mean prediction. [Online]. Available: https://www.kaggle.com/t88take/gsdc-phones-mean-prediction

[10] M. Bodych. (2021) Demonstration of the kalman filter. [Online]. Available: https://www.kaggle.com/emaerthin/demonstration-of-the-kalman-filter

[11] G. Rugg and M. Petre, *A Gentle Guide To Research Methods*. Open University Press, 2006, no. ISBN 9780335219278. [Online]. Available: https://books.google.com/books?id=AFDIsoyH6MIC&pg=PA182

[12] Ramswaroop. (2021) Baseline appraoch - linear regression. [Online]. Available: https://www.kaggle.com/ramswaroopbhakar14/baseline-appraoch-linear-regression

[13] J.-Y. Lee. (2021) Google smartphone decimeter eda + keras (tpu). [Online]. Available: https://www.kaggle.com/jeongyoonlee/google-smartphone-decimeter-eda-keras-tpu#Model-Training

[14] stackoverflow.com. (2021) How can i quickly estimate the distance between two (latitude, longitude) points? [Online]. Available: https://stackoverflow.com/questions/15736995/how-can-i-quickly-estimate-the-distance-between-two-latitude-longitude-points#