CB+NN Ensemble to Improve Tracking Accuracy in Air Surveillance

Anoop Karnik Dasika, Praveen Paruchuri

IIIT Hyderabad, Gachchibowli,
Hyderabad, Telangana, India - 500032
anoop.dasika@research.iiit.ac.in, praveen.p@iiit.ac.in

Abstract

Finding or tracking the location of an object accurately is a crucial problem in defense applications, robotics and computer vision. Radars fall into the spectrum of high-end defense sensors or systems upon which the security and surveillance of the entire world depends. There has been a lot of focus on the topic of Multi Sensor Tracking in recent years, with radars as the sensors. The Indian Air Force uses a Multi Sensor Tracking (MST) system to detect flights pan India, developed and supported by BEL(Bharat Electronics Limited), a defense agency we are working with. In this paper, we describe our Machine Learning approach, which is built on top of the existing system, the Air force uses. For purposes of this work, we trained our models on about 13 million anonymized real Multi Sensor tracking data points provided by radars performing tracking activity across the Indian air space. The approach has shown an increase in the accuracy of tracking by 5 percent from 91 to 96. The model and the corresponding code were transitioned to BEL, which has been tested in their simulation environment with a plan to take forward for ground testing. Our approach comprises of 3 steps: (a) We train a Neural Network model and a CatBoost model and ensemble them using a Logistic Regression model to predict one type of error, namely Splitting error, which can help to improve the accuracy of tracking. (b) We again train a Neural Network model and a CatBoost model and ensemble them using a different Logistic Regression model to predict the second type of error, namely Merging error, which can further improve the accuracy of tracking. (c) We use cosine similarity to find the nearest neighbour and correct the data points, predicted to have Splitting/Merging errors, by predicting the original global track of these data points.

Introduction

Object tracking is one of the most important problems in computer vision (Ozuysal, Lepetit, and Fua 2009) and robotics (Andreopoulos et al. 2010). It has a number of applications in industries such as entertainment (Belka 2019), security (Lyon 2006) and automobile (Al-Khedher 2012) among others. In the entertainment industry, tracking is primarily seen in VR applications involving tracking of hands, legs or even full body. In the automobile industry, tracking is used by self-driving cars and for enhancing the safety features of vehicles where objects such as other vehicles and obstacles are tracked. Tracking is seen in a variety of security applications such as surveillance of large areas (land, sea, airspace or even space applications).

Tracking of objects during air surveillance is typically performed using *radars* across most countries of the world since they work best in long distances and all kinds of weather.A radar (Ruotsalainen and Jylhä 2017) is an electromagnetic system that is useful for large scale surveillance. It can be viewed as an active sensor that can detect targets by emitting large power electromagnetic signals and also determines the targets' angle, range or velocity.

Given the range, precision and fault tolerance limitations of a single radar, multiple radars are used for a (wide area or) countrywide air surveillance, such that at any single point of airspace, there are multiple radars surveying. Hence, for each object being tracked (flights in this work), there are multiple radars tracking it at any single point of time in the airspace. This flight data from the different radars is then fused together called *Multi Sensor Fusion* **(MSF)**, and the systems which do this are called *Multi Sensor Tracking* **(MST)** Systems.

MSF has received a lot of attention in recent years due to a significant increase in applications involving the usage of a wide variety of sensors. Through the integration of data obtained from the different sensors, the results can, in general, be optimized better in terms of having a complete picture rather than having individual snapshots of a scenario. Considering the fact that newer radars with different properties may replace some of the older ones over time, defense organizations would, in general, need to perform MSF for a heterogeneous multi radar system. Statistical techniques like gating, Chi-square testing, Kalman and Advance Kalman filters are the most frequent techniques used in air surveillance by most countries as they are known to provide dependable and fast results. However, due to the differences in sampling rate of sensors, the communication delay between sensors and the overlapping regions of observations for the various sensors, there can be asynchronicity in the observations leading to a significant reduction in the tracking accuracy of an MSF system. There is, therefore, a strong need in the defense and commercial industry to develop solutions that improve their tracking accuracy.

Our contribution in this paper involves the development of a Machine Learning based approach, which acts as a layer

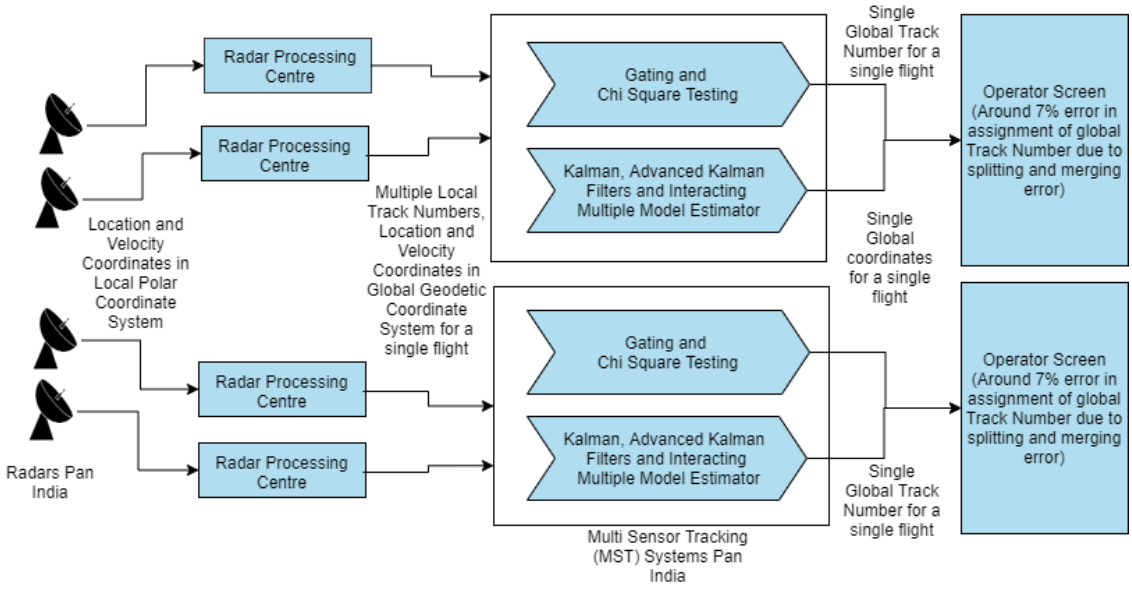

Figure 1: Working of MST System

and provide improvements in terms of tracking accuracy. The rest of this paper is structured as follows: We provide a brief introduction on radars and MSF based tracking in Section and define the problem being faced by the defense industry in Section . We propose our ML approach in Section , present the experimental results obtained using our approach in Section , present the path to deployment details in section and present the conclusions in Section .

Problem Description

The Indian Air Force uses a Multi Sensor Tracking (MST) system to detect flights pan India by generating an *Air Situation Picture* (ASP). This system is developed and supported by BEL(Bharat Electronics Limited), a defense agency we are working with. The sensors used are radars because of their ability to perform long-range detection and to perform optimally in all kinds of weather compared to other sensors. These sensors are located in different parts of the country, and each radar provides data in local polar coordinate system. This data obtained from individual radars is converted into global geodetic coordinate system in a processing centre while accounting for parameters like range, bias etc., of the radar. Each track sensed by each of the radars is assigned a unique local track number in the processing centre.

Let's say there are n radars covering m locations, i.e. parts of the country (many-to-one relation). Each radar r_i would have its own processing center p_i (one-to-one relation). The data processed by all the radars would correspond to one of the locations m_i, and each of the m locations has its own MST system MST_i (one-to-one relation between location and MST system).

Each MST_i uses data in its location m_i, to generate a single global track number gt_j for a flight f_j, which is to be used throughout the journey of the flight. In addition, MST_i also computes the location and velocity coordinates for each flight throughout its journey. In order to generate a global track number from multiple local track numbers, the MST system uses data association techniques like gating and chi-square testing, where each global track number represents a single flight. In addition, it uses parameter computation techniques like Kalman filter to obtain a single location coordinate for each flight from multiple location coordinates, advanced Kalman filter to compute a single velocity coordinate for each flight from multiple velocity coordinates and IMM (Interacting Multiple Model) estimator to improve the results obtained using the Kalman and advanced Kalman filters. Figure 1 captures the process flow involved.

An operational MST system has a lot of manual engineering involved apart from the above techniques and typically includes human-generated rules tailored to specific use cases. As the use cases expand, as heterogeneous radars get added, as the nature of targets change and as the requirements for accuracy increase, the system becomes more prone to errors due to the limitations in MST logic. Consider a scenario in which there are, say 4 air targets. Sensors gather data from these air targets and generate ASP displaying the 4 air targets with relevant attributes. However, there will be times when the actual number of air targets are shown as $4 + /4-$ for a certain period and then again shown as 4. If the MST system observes $4+$ air targets (when there are 4 targets in reality), we call the error as a **Splitting error** $\mathbf{E_s}$, i.e. sensing multiple targets when only a single one is present. When it observes $4-$ air targets (when there are 4 targets), we call the error as **Merging error** $\mathbf{E_m}$, i.e. sensing a single target when multiple targets are present. $\mathbf{E_s}$ can lead the system or operator looking at the ASP, to assume that there is an enemy flight in the air even though there is not any. $\mathbf{E_m}$ can lead the system or operator looking at the ASP, to assume that there is no enemy flight in the air even though there may be one (or more). Both these errors in air

surveillance can result in errors in threat evaluation, resulting in wrong action(s) getting taken, thus leading to severe threats and security issues.

Proposed Solution

Figure 2 presents a block diagram of our proposed ML solution, which helps to improve the tracking accuracy using the data obtained from the output of MSF. Our proposed solution takes inspiration from (Dasika and Paruchuri 2020), which deals with a similar but significantly smaller dataset and builds upon it. This data from MSF is used to train two separate models that can predict the data points with Splitting and Merging issues, respectively. Please note that the dataset includes flight code that helps obtain the target variables for Splitting and Merging models, while the real-time data does not contain flight code. The Splitting model $\mathbf{M_s}$ would provide Splitting confidence $\mathbf{C_s}$, i.e. probability of Splitting. When a data point has more than 0.5 $\mathbf{C_s}$, it is considered to have a Splitting error, else it is considered to be Real, i.e. only a single track is present. We identify the original Global Track number $\mathbf{G_t}$ for all the data points, which have Splitting Error. Similar to the Splitting model $\mathbf{M_s}$, the trained Merging model $\mathbf{M_m}$ provides the Merging confidence $\mathbf{C_m}$, i.e. probability of Merging.

Our approach comprises of the following 3 steps as shown in Figure 3: *In the first step,* we train a Neural Network model and a CatBoost model and ensemble them using a Logistic Regression model to classify all the data points into *Splitting or Real* (i.e., single track in reality although shown as multiple in operator screen), which can help to improve the accuracy of tracking. *In the second step,* we train a second Neural Network model and a CatBoost model and ensemble them using a Logistic Regression model to classify all the data points into *Merging or Real* (i.e., multiple tracks present in reality although shown as single flight in operator screen), which can improve the accuracy of tracking further. *The third step* uses cosine similarity to find the nearest neighbour to *correct the data points* classified as Splitting and Merging by finding the real global track from which the split track was obtained or finding the track which joined with the current track.

Step 1/2: Splitting/Merging Classification Model

Based on our understanding of the dataset, we identified the following features to use for training of Splitting/Merging model $\mathbf{M_s}/\mathbf{M_m}$ – latitude, longitude, altitude, speed and direction of an object detected by the MST system at each time step precise to a millisecond, number of radars, one-hot encoding of all the radars used and not used in the creation of this data point, time of day, 3 features formed using k-means clustering of latitude and longitude into $10, 100$ and 1000 cluster centres respectively and a unique id depicting the specific combination of radars used in the MST system for the creation of individual data points. Using the input data provided by individual radars, the MST system assigns a global track number $\mathbf{G_t}$ to each of the newly detected objects. If for a particular flight, a new $\mathbf{G_t}$ is assigned even though there is an existing one, all the data points detected with this new $\mathbf{G_t}$ are a result of Splitting from the original $\mathbf{G_t}$ of the flight and will be classified as Splitting(data points created as a result of Splitting error ($\mathbf{E_s}$). If for a particular global track $\mathbf{G_t}$, a new flight is associated even though there is an existing one, all the data points with this $\mathbf{G_t}$ associated with this new flight are a result of Merging of multiple flights and will be classified as Merging(data points created as a result of Merging error ($\mathbf{E_m}$).

As part of the *ensemble model* for the classification of Splitting/Merging, we first train a Neural Network with the train dataset. We then train a CatBoost algorithm with the train dataset. Next, we create a new train dataset using the predictions obtained from the above two models. We then train a logistic regression model on this new dataset, which determines the ratio to combine the outputs of the trained Neural Network and CatBoost so as to obtain the best (f1) score. The parameters obtained from training both the models are stored in a server and are used to check for Splitting or Merging issues whenever a new data point comes up. During real-time, this server listens to a socket in the MST system server. As soon as this server receives new data, it prepossesses the data by modifying and creating features required for the Splitting/Merging models. The preprocessed features are then passed to the model to obtain the Splitting/Merging prediction and their confidence values.

Step 3: Finding Original Global Track Number

In the previous steps, we identified the data points which have Splitting/Merging errors. Identifying the Splitting/Merging error alone is not sufficient to correct it, and we cannot just delete the data points which have Splitting/Merging errors, as we may lose important tracking information of targets. The third step, therefore, helps in correcting/retaining the data points, which were classified as Splitting/Merging. This step does not involve ML algorithms, hence it works directly in real-time without any training involved. Even in real-time, this step is only for those data points classified as Splitting by the Splitting Model and Merging by the Merging Model. In real-time, we keep storing the previous 1000 data points (p_{t-1000} to p_{t-1}) which are not classified as Splitting/Merging. For each new data point p_t classified as Splitting/Merging, we find the cosine similarity between this data point and the stored previous data points (p_{t-1000} to p_{t-1}) . We then pick the p_{t-k} previous data point that provides the highest cosine similarity with our data point p_t. We assign original global track number $\mathbf{G_t}$ of this point p_t to the global track number $\mathbf{G_{t-k}}$ of the picked previous data point p_{t-k}.

Experiments

Dataset

Our dataset contains 16 million data points obtained as part of 12 days of continuous data from the output of MST system. The raw data provided contains 121 columns with column names in bold and sample data in italicised font (for illustration purposes) -

1. **Time** (Discretized precise to a milisecond) - *66599204*
2. **Latitude** - *27.299*

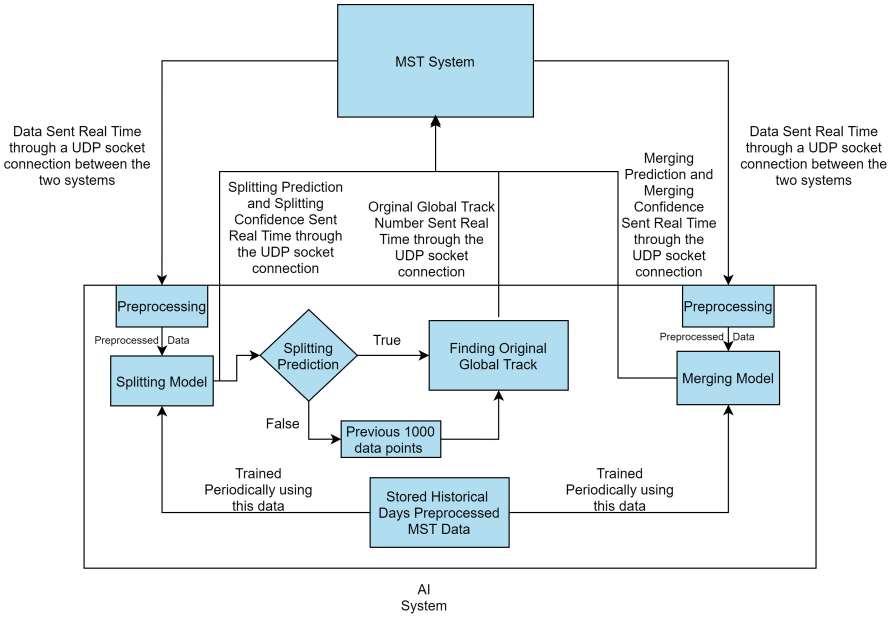

Figure 2: Proposed AI Methodology

3. **Longitude** - *80.1836*
4. **Speed** - *234.95*
5. **Course** - *311.049*
6. **IFF** - flight code (would not be present in real time) - *31747*
7. **GeoAltitude** - *10683*
8. **CTN** - global track number - *LE660*
9. **Call-Sign** - signal used to recognize type of flight - *SEJ7263*
10. **No of Radars** - No of radars whose sensor data was fused together to generate the current data point - *5*
11. **SAC** - first identifier of first radar - *4*
12. **SIC** - second identifier of first radar - *1*
13. **TrackNo** - local track number of the flight identifier by the first radar - *748*
14. **SAC1** - first identifier of second radar - *8*
15. **SIC1** - second identifier of second radar - *3*
16. **TrackNo1** - local track number of the flight identifier by the second radar - *4434*
17. 17-121) **SAC(2-n), SIC(2-n) and TrackNo(2-n)** first and second identifier and local track number of third to n radars if used in the generation of current data point.

It is post-processed by us as per the technique provided in section , since the data does not contain Splitting and Merging values. After preprocessing this raw data, we obtain the following preprocessed data which we use in our model training as shown in Figure 3 -

1. **IFF** - *31747*
2. **flight encoding** (encoding of IFF variable used for getting the target variables of splitting, merging and track cluster) - *4*
3. **CTN**- *LE660*
4. **track encoding** (encoding of CTN variable used for getting the target variables of splitting, merging and track cluster) - *10*
5. **Time** - *66599204*
6. **Time of day** - which hour of the day it is - *15*
7. **latitude** - *27.299*
8. **longitude** - *80.1836*
9. **altitude** - *10683*
10. **speed** - *234.95*
11. **direction** - *311.049*
12. **Call sign** - *SEJ7263*
13. **number of radars** - *5* The area under consideration is divided into states, cities and districts but that division is not helpful as a feature in air so we use k cluster with k as 10,100 and 1000 to make the below 3 features.
14. **location cluster1** - *4*
15. **location cluster2** - *24*
16. **location cluster3** - *501*
17. **4-1** (if this radar with SAC 4 and SIC 1 is used in the creation of a particular data point) - *1*
18. **8-3** (if this radar with SAC 8 and SIC 3 is used in the creation of a particular data point) - *1*
19. **6-3** - *0*
20. **3-7** - *0*
21. **Splitting** (Target variable which tells if a particular data point is created because of splitting error) - *1*
22. **Merging** (target variable which tells if a particular data point is created because of merging error) - *0*

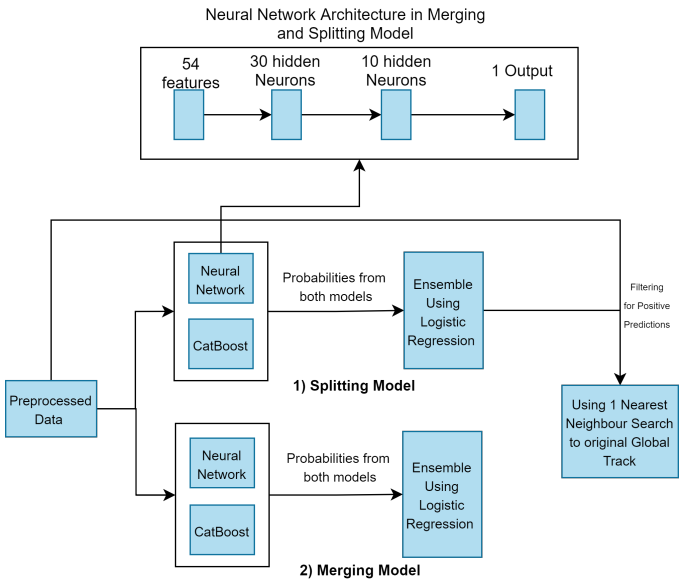

Figure 3: Proposed Machine Learning Models

23. **Track Cluster** (target variable which tells the correct CTN of the points with splitting and merging error) - *SEL8764*
24. 24-72) **SACn - SICn**

About 1.1 million data points in this preprocessed data are classified as *Splitting* (i.e., formed as a result of Splitting error) and about $400k$ data points classified as *Merging* (i.e., formed as a result of Merging error). This translates to an error rate of 9 percent (i.e., tracking accuracy of 91 percent) where 6.5 percent is due to Splitting error and 2.5 percent is due to Merging error. We use 80 percent of stratified random sampled data for *training* and 20 percent for *testing*.

Training

For the prediction of Splitting and Merging target variables, we experimented with several Machine Learning algorithms and created an ensemble using the ones which provide the best and unique results. We used a server with 1 RTX 2080 Ti GPU and 40 2GB RAM CPUs' for hyperparameter tuning and training. We used 3-fold cross-validation in the training set for identifying hyperparameters for each of the algorithms.

For algorithms with few parameter choices such as Logistic Regression (Kleinbaum et al. 2002), Gaussian Naive Bayes (Rish et al. 2001), Support Vector Machines (Scholkopf and Smola 2018) and K Nearest Neighbours (Liao and Vemuri 2002), we use grid search (Liashchynskyi and Liashchynskyi 2019) for hyperparameter tuning. For Algorithms, which have medium level of hyperparameter choices like CatBoost (Prokhorenkova et al. 2018) and XGBoost (Chen and Guestrin 2016), we use random search (Liashchynskyi and Liashchynskyi 2019). For Neural Networks, which have a high level of parameter choices, including the number of layers and their sizes, we use Bayesian optimization (Snoek, Larochelle, and Adams 2012) for hyperparameter tuning.

We present in table 1 and table 2, the Splitting and Merging results we obtained for the different ML algorithms. The best results were obtained for XGBoost, CatBoost and Neural Networks. While an ensemble of these models is likely to provide the best results, XGBoost and CatBoost are both tree-based non-parametric algorithms whose results overlap significantly, with CatBoost providing better results.

Hence, we use an ensemble of CatBoost and Neural Network, which have lesser overlap (with the former being a non-parametric algorithm while the latter is parametric). Using grid search, we identify the best hyperparameters for logistic regression - penalty type:l2 regression and C:1. Using Random Search, we find best hyperparameters for catboost - learning rate:0.1, max depth:8 and n estimators:200. Using Bayesian optimization, we find best hyperparameters for neural network - number of hidden layers:2, hidden layer sizes:(30,10), activation function : relu, optimization function : sgd, batch size:auto, learning rate initialization:0.1, learning rate:adaptive, max iterations:2000, early stopping True and alpha:0.0001. Our experiments show that we indeed obtain better results using the ensemble with the identified hyperparameters than using each of the techniques individually as shown in table 3 and 4.

Path to Deployment

The model and the corresponding code for both training and getting the model to work in real-time were transitioned to BEL, which has been tested in their simulation environment with a plan to take forward for ground testing. Our models were trained on 7 days of data and were tested on the dataset corresponding to the next 5 days. Per initial testing, the trained model seems to lose effectiveness after a period of time. Ground testing is planned to be performed

Sr No	Algorithm	Tracking Accuracy	Error Identification Accuracy
1)	Logistic regression	0.943	0.13
2)	Gaussian Naive Bayes	0.944	0.14
3)	Support Vector Machines	0.943	0.12
4)	K Nearest Neighbours	0.941	0.1
5)	Random Forest Classifier	0.946	0.17
6)	XGBoost	0.955	0.3
7)	Stacking Logistic Regression with XGBoost	0.956	0.33
8)	Neural Networks	0.949	0.22
9)	CatBoost	0.961	0.41
10)	Ensemble - CatBoost with Neural Networks	0.964	0.45

Table 1: Different Algorithm Results for Splitting Error

Sr No	Algorithm	Tracking Accuracy	Error Identification Accuracy
1)	Logistic regression	0.979	0.15
2)	Gaussian Naive Bayes	0.978	0.1
3)	Support Vector Machines	0.975	0
4)	K Nearest Neighbours	0.981	0.23
5)	Random Forest Classifier	0.982	0.28
6)	XGBoost	0.984	0.35
7)	Stacking Logistic Regression with XGBoost	0.984	0.36
8)	Neural Networks	0.98	0.18
9)	CatBoost	0.986	0.44
10)	Ensemble - CatBoost with Neural Networks	0.988	0.5

Table 2: Different Algorithm Results for Merging Error

	Predicted : 0	Predicted : 1
Actual : 0	2.79M	10k
Actual : 1	107k	88k

Table 3: Confusion Matrix of Splitting

	Predicted : 0	Predicted : 1
Actual : 0	2.92M	5K
Actual : 1	37k	38k

Table 4: Confusion Matrix of Merging

to identify the optimal time after which we may need to retrain the model, which may be periodical or when changes like bias, range, etc are made to any radar. Ground testing would also be used to determine, if each zone should have its own model(s) or if all the MSF data from pan India should be assembled together to train a single Splitting and Merging model. The best course of action to take and steps for production deployment will be identified after this phase of ground testing, which is likely to take several months.

Our AI system is currently deployed in a separate server, called AIMSF, different from the existing server which has the MST system in BEL. Following are the steps which occur in real-time:

- A UDP socket connection is set between the existing server and the AIMSF server.
- Whenever a new datapoint enters the existing server, it gets sent to the AIMSF server.
- The AIMSF server processes this data and identifies the Splitting confidence, Merging confidence, Splitting prediction, Merging prediction and the original global track using the trained Splitting and Merging models. It then sends back these results to the existing server.

Simulation of scenario and results would be visible on the operator screen, who can determine a suitable course of action to take.

Conclusion

In this paper, we focus on improving the tracking accuracy of a real-world Multi Sensor (Radar) Tracking System used in the context of tracking flights across the Indian airspace. Several errors happen during the fusion process, primarily categorized into Splitting and Merging errors. We introduced a three-step ML approach to improve the confidence in tracking and reduce the errors observed. In particular, we trained models to predict the Splitting or Merging of tracks and used cosine similarity to identify the nearest neighbour and correct the Splitting and Merging errors. Using this three-step process, we are able to improve the accuracy of tracking from 91 percent to about 96 percent on anonymized real-world data.

Acknowledgments

We would like to thank BEL (Bharat Electronics Limited), Ghaziabad, for their generous support of this work.

References

Al-Khedher, M. A. 2012. Hybrid GPS-GSM localization of automobile tracking system. *arXiv preprint arXiv:1201.2630*.

Andreopoulos, A.; Hasler, S.; Wersing, H.; Janssen, H.; Tsotsos, J. K.; and Korner, E. 2010. Active 3D object localization using a humanoid robot. *IEEE Transactions on Robotics*, 27(1): 47–64.

Belka, R. 2019. An indoor tracking system and pattern recognition algorithms as key components of IoT-based entertainment industry. In *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2019*, volume 11176, 111765P. International Society for Optics and Photonics.

Chen, T.; and Guestrin, C. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 785–794.

Dasika, A. K.; and Paruchuri, P. 2020. An Ensemble Learning Approach to Improve Tracking Accuracy of Multi Sensor Fusion. In *International Conference on Neural Information Processing*, 704–712. Springer.

Kleinbaum, D. G.; Dietz, K.; Gail, M.; Klein, M.; and Klein, M. 2002. *Logistic regression*. Springer.

Liao, Y.; and Vemuri, V. R. 2002. Use of k-nearest neighbor classifier for intrusion detection. *Computers & security*, 21(5): 439–448.

Liashchynskyi, P.; and Liashchynskyi, P. 2019. Grid search, random search, genetic algorithm: a big comparison for NAS. *arXiv preprint arXiv:1912.06059*.

Lyon, D. 2006. *Theorizing surveillance*. Routledge.

Ozuysal, M.; Lepetit, V.; and Fua, P. 2009. Pose estimation for category specific multiview object localization. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 778–785. IEEE.

Prokhorenkova, L.; Gusev, G.; Vorobev, A.; Dorogush, A. V.; and Gulin, A. 2018. CatBoost: unbiased boosting with categorical features. In *Advances in neural information processing systems*, 6638–6648.

Rish, I.; et al. 2001. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, 41–46.

Ruotsalainen, M.; and Jylhä, J. 2017. Learning of a tracker model from multi-radar data for performance prediction of air surveillance system. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, 2128–2136. IEEE.

Scholkopf, B.; and Smola, A. J. 2018. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Adaptive Computation and Machine Learning series.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25.