Predicting the spatial coordinates of particles by employing ML techniques

Adel Aboutalebi Pirnaeimi
Politecnico di Torino
Student id: s324930
adel.aboutalebipirnaeimi@studenti.polito.it

Mohammad Saghali
Politecnico di Torino
Student id: s328134
s328134@studenti.polito.it

***Abstract*—The report focuses on predicting the spatial coordinates of particle interactions in particle physics. Through meticulous preprocessing and strategic feature selection, we fed refined data into both random forest regressors and multi-layer perceptron (MLP) based models. Further, the integration of ensemble techniques and strategic rounding in the models' predictions demonstrates a notable improvement in predictive accuracy. The results showed that the proposed models provide a robust solution to the problem of predicting the (x,y) coordinates of particles.**

I. Problem overview

A challenging aspect of particle physics is detecting the trajectory of particles. RSDs (Resistive Silicon Detectors) improve particle position detection. When particles hit RSDs, they create signals which contain a lot of valuable information, including amplitude, area, width, and slew rate. The hit positions can be reconstructed using this information. The information obtained from the detected signals is used in this project to estimate the (x,y) coordinates of the hit position [1].

Our study involves two datasets: the development dataset with 385,500 events, and the evaluation dataset with 128,500 events. Each event in these datasets captures a unique interaction of a particle with sensor pads, recording various signal properties. These interactions provide trajectories with specific (x, y) coordinates. From our analysis of the development dataset, as shown in Fig. 1, we found that the x and y coordinates range from 200 to 600 and typically increase in increments of five. The data collected from the sensor pads include several types of signal measurements: the highest positive signal peak (pmax), the highest negative signal peak (negpmax), the time delay for the peak signal (tmax), the area under the signal curve (area), and the root mean square value of the signal (rms).

The dataset posed a unique challenge due to hardware limits and noise: it had 18*5 feature readings per event, but we expected only 12*5 features, as there are just 12 sensor pads. For feature analysis, we grouped them – for example, pmax[0] to pmax[17], area[0] to area[17], and so on. By creating box plots for each group, we understood their behavior better. We noticed that some feature groups had unusual patterns, especially pads 0, 7, 12, 15, 16, and 17. These pads behaved differently in the pmax, tmax, and area groups compared to others. Fig. 2 shows this clearly with the pmax values of these pads differing from the others. This way of visualizing helped us identify pads with unusual behavior, giving us insights into potential noisy features in our dataset.

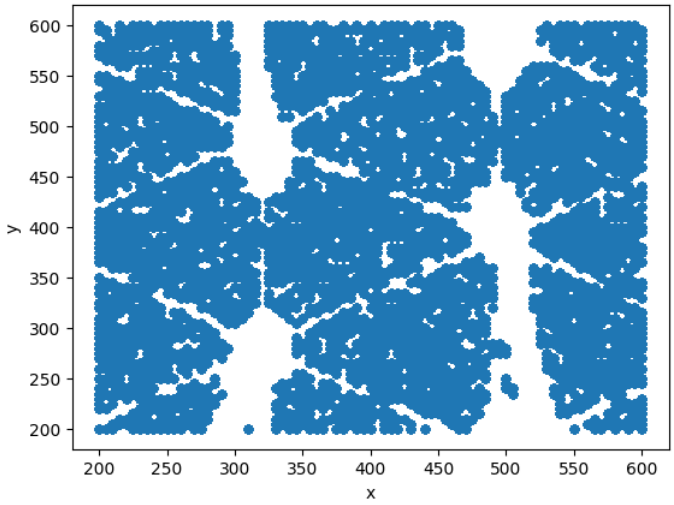

Fig. 1: The x and y coordinates shown in the image range from 200 to 600.

II. Proposed approach

A. Preprocessing

The preprocessing phase in the regression pipeline is crucial to making accurate predictions of (X, Y) coordinates. In addition, it facilitates the creation of features that capture relevant spatial relationships while ensuring the quality and consistency of the data. A regression model's robustness and accuracy are enhanced by scaling, normalizing, handling and reducing noise. A well-preprocessed dataset lays the foundation for a successful predictive model, making it essential for achieving accurate predictions. Several of these techniques will be utilized in this study, which will be briefly discussed.

1) Feature selection: At this stage of the regression process, we chose to treat the prediction of x and y coordinates as two different tasks. This decision leads to two simpler tasks, making it easier to focus on feature engineering and model building for each coordinate separately. Therefore, we developed two separate models: one for the x and the other

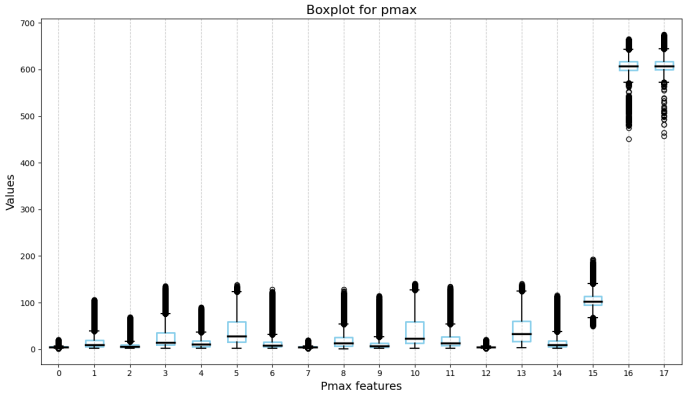

Fig. 2: The figure presents a boxplot for the pmax values across different sensor pads, highlighting that pads 0, 7, 12, 15, 16, and 17 have readings that stand out from the others. These differences suggest that the data from these pads might be noisy or less consistent compared to the rest.

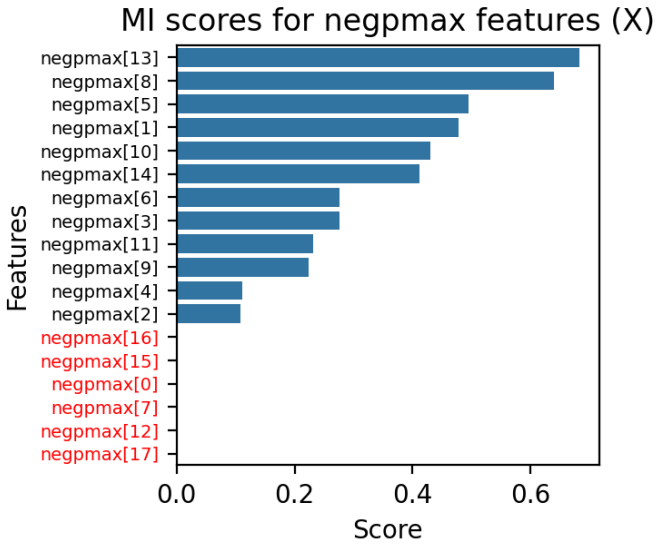

Fig. 3: The figure illustrates the Mutual Information (MI) scores for negpmax features, which quantify their predictive relevance for the x coordinates. A threshold of 0.02 for MI scores has been applied to select features. Those falling below this threshold, indicating lower predictive power for the x coordinate, red colors have been omitted.

for the y, since we consider these predictions to be separate challenges.

From our previous analysis, where we plotted different attributes, it was evident that there are potential noisy features within the dataset. Choosing the right attributes for our model is key to improving predictions. However, simply removing these features is challenging, as they vary significantly across different groups. To effectively select them that will enhance our model's performance, we employed algorithms like mutual information and random forest regressor feature importance.

a) Mutual Information, Filtering Features Relevant to the Target: Mutual information, a feature selection metric, quantifies the information shared between each feature and the target variable (coordinates in this case). Features with high mutual information are considered more relevant to the prediction task. This method not only aids in selecting informative features but indirectly assists in reducing the influence of outliers by prioritizing features with stronger predictive power.

We use mutual information to evaluate both x and y separately, assigning a rank to each feature based on its informativeness for x and y. For example, Fig. 3 illustrates the mutual information score for each negpmax feature in relation to the x coordinates. our experimentation revealed that by selecting features that have at least 0.02 information related to x and y we could reach suitable result for our dataset. After defining 0.02 threshold for mutual information we have 51 features for x and 49 features for y.

b) Random Forest Regressor Feature Importance, Robustness Against Outliers: After choosing features for x and y using mutual information, we then pass these features to the random forest regressor to determine their importance. We believe that using both mutual information and random forest feature importance as ways to select features has two main advantages. First, it helps us find features that are good at predicting, so we don't have to depend too much on features that are affected by outliers. Second, it makes use of the random forest's ability to deal with outliers well, which leads to a stronger and more reliable model.

Random Forest Regressor Feature Importance offers a unique advantage in outlier mitigation. By evaluating the contribution of each feature to the overall predictive performance, Random Forest inherently diminishes the impact of outliers. The ensemble nature of Random Forest, comprised of multiple decision trees, ensures that the influence of extreme values is mitigated during the aggregation of predictions [2].

We use the random forest regressor to find out how important each feature is for x and y, and then we rank them based on this importance. Figure 4 shows the importance score for each feature that we still had after picking features with mutual information for the y coordinates. From our testing, we found that choosing features with an importance score of 0.001 or more for both x and y gives us good results for our dataset. Once we set this level of importance as our cut-off, we had 39 features for x and 38 features for y.

2) Scaling and Normalization: A crucial step involves standardizing the features to ensure a robust and effective training process for our machine learning model. To achieve this, we employ scikit-learn's StandardScaler [3], except for the coordinates (x and y) representing the trajectory points of the particles. Standardization is vital to establish a standard normal distribution with a mean of 0 and a standard deviation of 1 for each feature. This mitigates the impact of varying scales and units across our diverse set of remaining features, promoting convergence during training, enhancing generaliza-

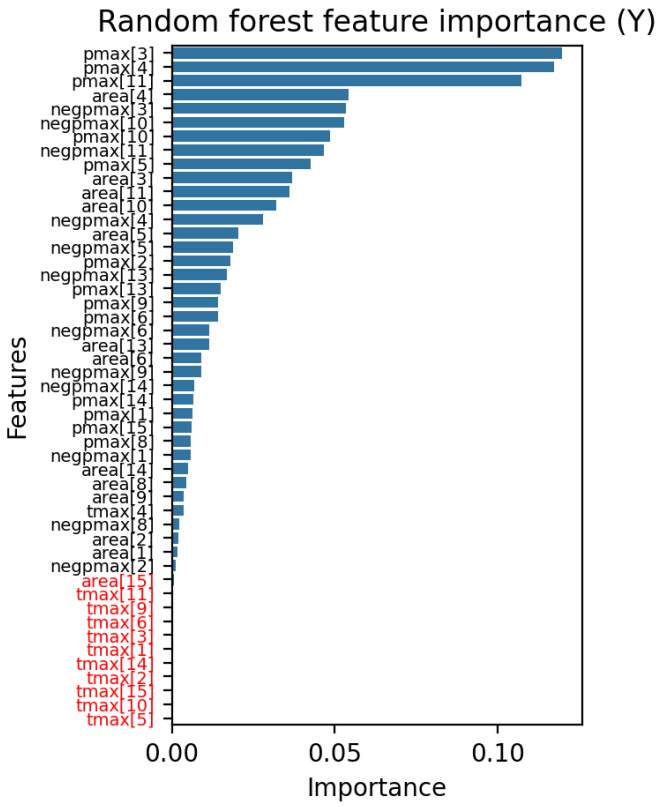

Fig. 4: The chart shows the feature importance of remaining features, focusing on the features that influence the prediction of the y coordinate. We chose a threshold of 0.001 to keep the features. The features that scored above this threshold in terms of their importance for predicting y are displayed as black color.

tion capabilities, and enabling effective pattern recognition within the complex data. While we also explored the option of min-max normalization, Our experiments showed that using the z-score normalization method led to better results when training our dataset which we will explore it in more detail in following sections.

B. Model selection

In our model selection process, we adopt a tailored approach by selecting features separately for each x and y coordinate, subsequently training two distinct models—one for x and another for y. To address the unique challenges posed by our regression task, we independently leverage two different algorithms: the Random Forest Regressor and the Multi-Layer Perceptron (MLP). The subsequent sections of this report delve into the structural details of each model.

1) Random Forest Regressor: In our regression task to predict x and y coordinates, we applied the Random Forest Regressor, an ensemble learning method renowned for its effectiveness in regression scenarios [4]. This algorithm operates by constructing a plethora of decision trees during the training phase and then generating predictions by averaging the outputs of these individual trees. Such a technique not only provides stability to the model by diminishing the risk of overfitting but also improves the accuracy of its predictions.

We made two Random Forest Regressor models, each dedicated to predicting one of the two coordinates. We chose to use 100 trees for each model because this gives a good trade-off between the speed of computation and the quality of the predictions—the more trees used, generally, the better the model can predict. We set `max_features` to `sqrt`, so each tree looks at a subset of features—specifically, the square root of the total number of features—when it decides where to split. This makes each tree in the model a bit different from the others, which makes the overall model more varied in how it makes decisions. We also made sure each tree learned from a different part of the data by using bootstrapping. This variety is one of the key strengths of the random forest method. With `min_samples_split` at 2, our model can make a decision with even a small amount of data, which lets it pick up on more subtle patterns in the data. And by not setting a `max_depth`, the trees in our model can grow as big as they need to, as long as they keep getting purer in the predictions or until they have less than `min_samples_split` data points to make a decision on.

2) MLP: In this part, we utilized a neural network approach by implementing the MLP regressor, a type of architecture known for its capacity to learn and model complex, non-linear relationships within datasets [5]. The MLP consists of multiple layers of nodes, or neurons, which are intricately connected to form a network capable of detecting subtle patterns and adapting to the various dependencies among features.

In the construction of our models—separate entities for each coordinate axis—we defined a structure with three layers of neurons, consisting of 128, 64, and 32 nodes respectively. This architecture was chosen to create a balance between the model's ability to learn detailed patterns and its computational demand. The `tanh` activation function was employed for its efficacy in allowing neurons to model complex functions. The `adam` solver, a popular optimization algorithm, was utilized for adjusting neuron weights efficiently, thus reducing the error between the predicted outcomes and the actual values. Furthermore, the models were designed to automatically determine the optimal batch size for processing and employed an 'adaptive' learning rate, which adjusts as training progresses to improve convergence. The iterative process was set to run for a maximum of 70 iterations.

During training, these MLP models were fitted with a subset of features specifically selected for their relevance to each target variable, x and y. This methodical feature selection, paired with the strategic configuration of the MLP models, equips the system to effectively harness the power of neural networks in capturing the complexities inherent in predicting x and y coordinates, thereby catering to the nuanced demands of the regression task at hand.

3) Ensemble MLP: In this section, the x and y coordinates were predicted using an ensemble of MLP models. This approach capitalizes on the strengths of MLP in regression,

while further enhancing its effectiveness by combining the insights from multiple models.

For each coordinate, x and y, we created a set of MLP models. These models vary slightly in their configuration to capture a broader range of patterns in the data. Some models in the ensemble have two hidden layers with 100 and 50 neurons, respectively, while others are more complex with three hidden layers comprising 128, 64, and 32 neurons. This variety in architecture allows each model to learn different aspects of the data.

4) Ensemble MLP with rounding strategy: Following the application of an ensemble of MLP models and the calculation of their average predictions, we introduced a rounding strategy for the final predicted values. This decision was informed by insights gained during our data analysis phase.

Our analysis revealed that both x and y coordinates typically range between 200 to 600 and increment in steps of five. To align our predictions with this observed pattern, we applied a rounding function to the averaged predictions from the MLP ensemble. This function systematically rounds each predicted value to the nearest number that is divisible by five.

C. Hyperparameters tuning

In this section, we discuss the approach we took to determine the best settings for our MLP and ensemble MLP models. This involved a trial and error method that was crucial in achieving the best outcomes for our dataset. For the hidden layer sizes, we initially tried various configurations, including a complex model with four hidden layers (256, 128, 64, 32). However, we found that simpler structures with either (128, 64, 32) or (100, 50) neurons were more effective for our specific needs. As for the activation function, our initial use of the ReLU function was reconsidered. Since we have negative values in our data, using ReLU will cause us to lose part of the data. So we realized that the `tanh` function was a better choice due to its ability to handle these negative inputs.

Regarding the maximum number of iterations, we initially experimented with higher values like 100 and 200. However, reducing this to 70 and 65 proved beneficial in preventing overfitting, a critical aspect of maintaining accuracy in our model. It was important to find a sweet spot here, as too few iterations could lead to underfitting. For the learning rate, we opted for an `adaptive` setting, which dynamically adjusts during the model's training process. This choice yielded better results compared to the `constant` and `invscaling` options. The alpha and initial learning rate were kept at their default values (0.0001 and 0.001, respectively), as these provided a good balance and enhanced the performance of our models. This process of hyperparameter tuning was pivotal in refining our model to suit the unique characteristics of our dataset, highlighting the significance of such adjustments in the realm of machine learning.

III. Results

As depicted in Table I, we observed the performance of various algorithms on our dataset, measured by the Euclidean distance. The Random Forest Regressor achieved a distance of 4.824, indicating its effectiveness in the task. The MLP model further improved this metric, bringing it down to 4.251. An even more notable enhancement was observed with the ensemble MLP, which reduced the Euclidean distance to 3.98, demonstrating the strength of combining multiple models. The most significant improvement, however, was achieved by implementing the ensemble MLP with a rounding strategy, which further lowered the distance to 3.851. This progression of results clearly illustrates the benefits of each successive approach.

Model	Euclidean Distance
Random Forest Regressor	4.824
MLP	4.251
Ensemble MLP	3.98
Ensemble MLP with rounding strategy	3.851

TABLE I: Performance of models on the test set.

IV. Discussion

Given the constraints of time and resources, and the satisfactory outcomes achieved, the research met our expectations. However, future endeavors could explore the direct removal of outlier data points for potentially improved predictions, beyond the indirect reduction through our current feature selection methods. Additionally, due to the lack of support for training with two separate feature lists in the multioutput regressor library, we had to train models for x and y independently, which was less efficient. With access to more robust resources, expanding the number of MLP models in our ensemble could be explored to better address overfitting and enhance results.

References

[1] F. Siviero, F. Giobergia, L. Menzio, F. Miserocchi, M. Tornago, R. Arcidiacono, N. Cartiglia, M. Costa, M. Ferrero, G. Gioachin, *et al.*, "First experimental results of the spatial resolution of rsd pad arrays read out with a 16-ch board," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 1041, p. 167313, 2022.

[2] G. Louppe, "Understanding random forests: From theory to practice," 2015.

[3] "Scikit-learn standardscaler documentation." Accessed: [Accessed 26-01-2024].

[4] "sklearn.ensemble.RandomForestRegressor — scikit-learn.org." [Accessed 26-01-2024].

[5] "sklearn.neural_network mlpregressor — scikit-learn 0.24.1 documentation." Accessed: [Accessed 26-01-2024].