Regression Problem : *Particle Position Prediction*

[*]Note: LLMs have not been used for generating any kind of content in this report.

Ali AL HOUSSEINI
Politecnico di Torino
Student ID : s287634
s287634@studenti.polito.it

Sana AILLA
Politecnico di Torino
Student ID : s321630
s321630@studenti.polito.it

***Abstract*—In this report we present a multi-output regresion pipeline to predict the positions of particles (e.g, electrons) as they pass through a Resistive Silicon Detector (RSD). In particular, the proposed approach consists on exploiting the characteristics of signals measured by RSD pads (total of 12 pad) as inputs to build robust multi output regression models to predict the particle position (x, y) as output. The proposed approach has shown stable and satisfactory results superior than provided results of the baseline.**

I. Problem overview

The task we are focusing on is a multi-output regression problem involving a dataset composed by (x, y) coordinates of particle events and the signal characteristic from the detector's pads. The objective of this task is to accurately predict the position of the particle at each occurent event. The dataset is divided into two parts :

- *Development* set: containing $385,500$ events and particle positions (x, y).
- *Evaluation* set containing $128,500$ events.

Each event in the dataset is characterized by 90 features, which represent the signal characteristics measured by each pad of the RSD. The key features per pad include pmax, negpmax, area, tmax, (See Figure 1) and the root mean square (rms).

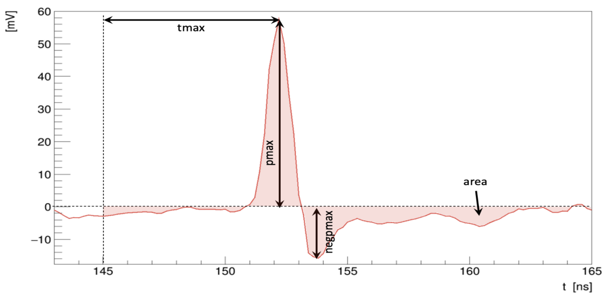

Fig. 1. An example of a signal measured by one of the pads of a sensor, for a given event.

First, we need to make some considerations based on the development set:

- **Inconsistent Features**: The dataset includes 90 features for each event, instead of the expected 60 features (5 features for each of the 12 pads). This is due to a hardware constraints during data acquisition. Consequently, for each pad, there are 6 additional measures, representing noise that needs to be removed. This issue is also present in the evaluation set and is addressed as the first step in our pipeline.
- **Sensor Coverage Gaps**: Visual analysis x and y values, we can notice that some areas of the sensor are not covered by any event. That occurs because of the presence of pads. See Figure 2
- Values of extracted features of signals have different ranges within the same feature. Some of these values are extremely divergent, suggesting the presence of outliers. Upon a manual inspection using TABLEAU tool, we confirmed our assumption that the dataset represents a considerable number of outliers that should be removed. Example Figure 3.

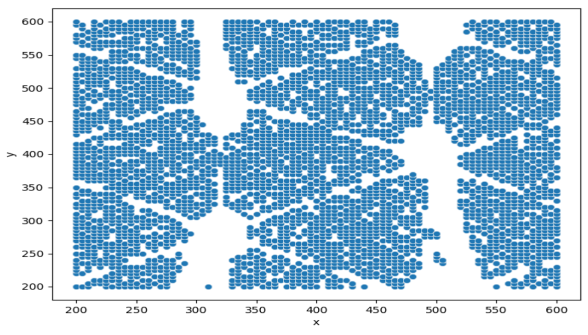

Fig. 2. Position of events from the developement set

II. Proposed approach

A. Data Preprocessing

As mentioned before, our dataset comprises extracted features rather than raw signals. The preprocessing steps play a crucial role in preparing the data for subsequent analysis and building the model. The preprocessing phase involved two primary steps regarding noise removal:

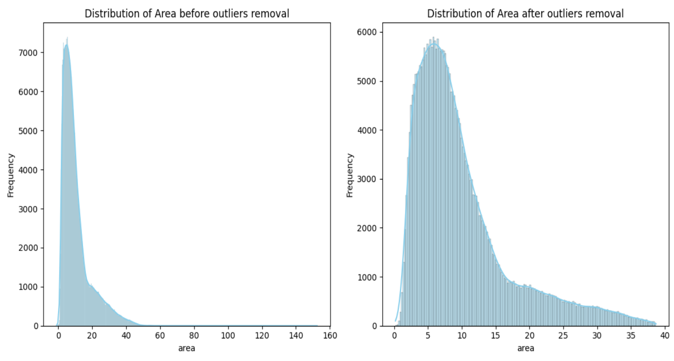

Fig. 3. Distribution of the area under the signal measured by pad number 2 before and after the removal of outliers.

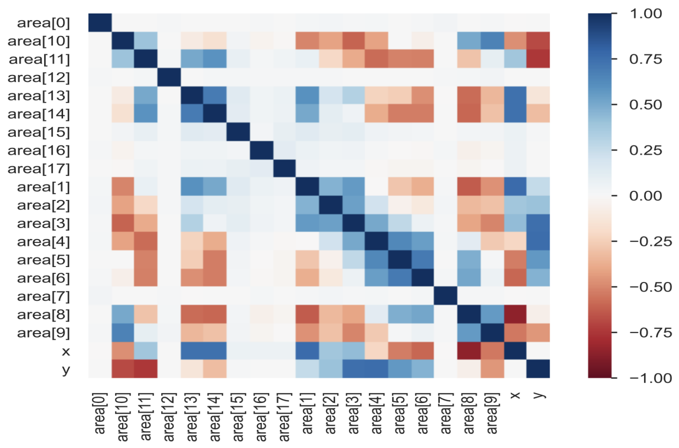

Fig. 4. Correlation matrix of area measures and targets (x, y)

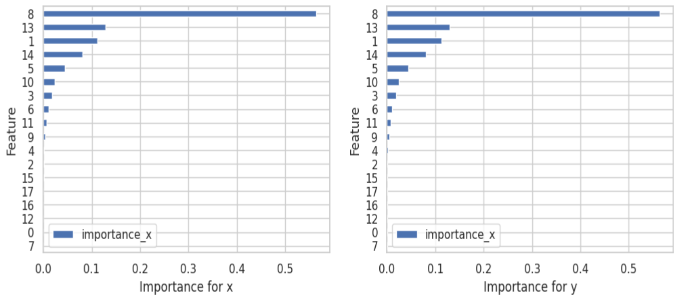

Fig. 5. Feature importance of a tree-based model (Random Forest) for area characteristic for 18 features

1) Noise removal: Given that the dataset contains additional features due to hardware constraints during data collection, it was imperative to identify and eliminate noise to enhance the quality and accuracy of the data, thus, we employed some methods:

- **Correlation Analysis Approach**: We analyzed the correlation of the 18 given measures with the target (x, y) and selected the top 12 relevant features for each characteristic. For instance, the area measures at indices (0, 7, 12, 15, 16, 17) exhibited low correlations with both x and y targets, as well as with other features. This led to the identification of these indices as noise columns for the area measure. A similar analysis identified noise columns (0, 7, 12, 15, 16, 17) for the other characteristics (negpmax, pmax, tmax, and rms) [See Figure 4]. Further analysis based on feature variances and mean values was also conducted.
- **Wrapper Approach:** We built a tree-based model (Random Forest in our case) for each measure in a multi-output regression task with the target (x, y) to assess feature importance, [Figure 5]. This allowed us to rank features by relevance and identify noise columns.
  The above-mentioned approaches led to the same results, confirming that the noise columns to be removed were consistent across different characteristics and indexing (0, 7, 12, 15, 16, and 17). Additional investigations using the Tableau tool were conducted specifically for the rms characteristic.
- **Noisy features:** The measured rms were too noisy, and therefore, the use of some interactive-professional data visualization tools was required. After further investigations, we concluded that RMS values are noisy features and therefore we removed them.

Starting from now, we are going to deal with a new development set without noise features. Here, we recall that we have a total of 50 features: The coordinates, and 4 measures for each pad out of 12 pads.

2) Outliers removal: To address the issue of outliers in the given development set, we explored two machine learning approaches, specifically the *iForest* algorithm and *One-Class SVM*. These models requires a predefined parameter representing the percentage of outliers present in the dataset. However, implementing these models with a fixed percentage of outliers poses challenges because the selection of an appropriate percentage often requires more supervision and careful consideration.

Thus we investigated a statistical approach based on Z-score combined with Principal Component Analysis (PCA) [1] to identify and remove outliers in the new development dataset.

- **Z-score** is a statistical measure that quantifies how many standard deviations a data point is from the mean of a dataset. For a given data point x, its Z-score is calculated as:

$$Z = \frac{x - \mu}{\sigma}$$

where μ is the mean and σ is the standard deviation of the dataset. A data point is considered an outlier if its Z-score is outside a predefined range generally $[-M\sigma, +M\sigma]$.
- **PCA** is a dimensionality reduction technique used to transform high-dimensional data into a lower-dimensional representation while preserving the most important information.

Given the correlation among features, the removal of outliers in one column necessitates a re-evaluation of the entire dataset. This is because the removal of outliers in one feature might alter the statistical properties (like mean and standard deviation) of other correlated features, potentially changing their outlier status.

The threshold M ($2.5 \leq M \leq 5$) in the Z-score method, which defines the range for outlier detection, is not a one-size-fits-all parameter. Instead, it requires adjustment based on the specific characteristics of each feature in the dataset. This customization is crucial to accurately identify outliers without unnecessarily excluding valid data points.

The combined use of PCA and Z-Score in our proposed technique is designed to effectively identify and remove outliers in a dataset with correlated features. Here's how the technique work:

First, we start by introducing a set S where S is the set of selected features after removing outliers, noise, and RMS columns.

Let $S_i \in S$ a subset where S_i is the set of features related to pad i. We recall here that our selected pads are $i \in [1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 13, 14]$. Then, we define a PCA algorithm to apply on Normalized data where each feature has been normalized using both Standard Scaler and MinMax Scaler. The goal of these two normalizations is to benchmark (sort) the principal components according to their absolute value.

Thus, we define P_{ij} where $0 \leq j < 5$, the set of Principal Components retrieved by applying PCA to S_i. The elements of P_{ij} are then sorted decreasingly as mentioned earlier.

Finally, and according to the order in P_{ij} we start removing outliers for each feature in S_i. Thus, now we have a cleaned dataset that is ready to be an input for a model, and this what we are going to discuss in the next section.

B. Model Selection

First, since our task is a multi-output regression [4], it was necessary to experiment models according to two different benchmarks. The first, is a set of models that are flexible for multi-output regression, while the second is a set of models that are not possible to be tested for a multi-output regression task. Thus, we started first by exploiting general models for multi-output task.

In the regression pipeline we built, we used a mutioutput regression considering the nature of the task (regression on x and y), to do so we experimented with several estimators as input of the multioutput regressor. Yet, we anticipate that two independent models have been outperforming any multi-output regressor model.

During the initial phase of our experimentation, we assessed a variety of models, including K-Nearest Neighbors (KNN), Support Vector Machine (SVM), Linear Regression, Ridge Regression, Lasso Regression, and Gradient Boosting. These models, however, were quickly eliminated from further consideration due to their suboptimal performance in preliminary evaluations.

- Random Forest Regressor [2] : An ensemble learning technique that combines multiple decision trees to provide accurate predictions for regression tasks. This approach helps reduce overfitting that can occur in individual decision trees, leading to improved predictive performance and robustness. The performance of this model depends on many parameters, including the number of estimators used, and it can be tuned using similar parameters as normal decision trees (e.g., maximum depth). Random Forest works on features independently, which is why a normalization step is not necessary. We chose to work with Random Forest because, in addition to their providance by feature importance that helps us removing noise , they have demonstrated the ability to capture non-linear and complex relationships between features and target variables better than other models, such as linear models.

- A Multi-Layer Perceptron (MLP) is a class of feedforward artificial neural network (ANN) that consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Each node, except for the input nodes, is a neuron with a nonlinear activation function. MLPs utilize a supervised learning technique called backpropagation for training. They are well-suited for complex pattern recognition, which makes them a strong candidate for model selection in scenarios where non-linear relationships and interactions within data are suspected.

- Extra Trees Regressor [2] : Also an ensemble learning technique, similar to Random Forest Regressor, but with a key difference in how it constructs individual trees. In RF, a random subset of features is selected for each tree, and the best feature is chosen for each node split. In contrast, ET uses a random subset of features for each tree and selects a random threshold value for each feature to split the node. The reason behind this choice is that Extra Trees has a faster training speed while still delivering performance similar to Random Forest. This decision makes sense practically, as it enables us to build a strong regression model efficiently given the relatively big amount of data we have and limited resources.

In our study, all models were initially tested within a multi-output regression framework. Interestingly, we observed that the performance of models configured as independent surpassed multi-output. This outcome aligns with expectations, as independent models often have the advantage of being finely tuned to specific outputs.

C. Hyperparameters tuning

For the three described models, the best-working configuration of hyperparameters has been identified through a grid search accompanied of diferent combinations of parameters mentionned in I. Other parameters exist and can be added to the grid search.

During the hyperparameter tuning, we started by training a model for each algorithm mentioned before with the default parameters and assessed their performance using 30% of the original dataset. In fact, we performed hyperparameter tuning with a 70% of data to train the model and use the rest to asses the performance of the best resulting model. We defined the scorer as the Euclidian distance between predictions defined as :

$$d = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2}$$

where n is the number of events, and (x_i, y_i) represent the ground truth, while $(\hat{x}_i, \hat{y}_i)$ represent the predicted position.

Model	Parameter	Values
Random Forest	*estimator__n_estimators*	$\{200, 300, 900\}$
	estimator__max_depth	$\{None, 15, 30\}$
Extra Trees	*estimator__n_estimators*	$\{200, 300, 900\}$
	estimator__max_depth	$\{None, 15, 30\}$
MLP	*estimator__activation*	$\{'relu', 'tanh'\}$
	estimator__alpha	$\{0.0001, 0.001, 0.01\}$
	estimator__max_iter	$\{200, 300, 500\}$

TABLE I
HYPERPARAMETERS CONSIDERED

As a total result, we trained around 30 models, including 3 with default parameters and 9 for each algorithm as a result of hyperparameter tuning (see Table I).

Given the nature of single-variate models, the Euclidean distance metric was unsuitable for our analysis in this case, we selected the Root Mean Sqaure Error

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

This choice was made according to the similarity of $RMSE$ to the Euclidean distance in terms of evaluating the magnitude of error between predicted and observed values. $RMSE$ effectively captures the average magnitude of errors across predictions, making it a robust measure for single-variate models. Additionally, for a more comprehensive evaluation, we computed the Euclidean distance post-training by using two distinct models, one for the x-variable and another for the y-variable.

III. Results

As a result of hyperparameter tuning, the tree-based models (ET and RF) outperform the MLP algorithm in terms of distance measure ($d_{\text{MLP}} > 7$), an unacceptable result compared to the baseline model. However, the RF model leads to a smaller and thus considerable performance improvement (since we are working on distance measures) compared to the baseline; in fact, $d_{\text{Tree Models}} \in (4, 5)$. Thus, to strike a compromise between performance and time, we chose the Extra Trees regressors, leading to a distance on the test set of $d_{\text{ET}} = 4.7$ with $best\_params = n\_estimators = 900, max\_depth = None$.

The public score obtained is $d_{\text{ET}} = 4.770$ for Extra Trees and $d_{\text{RF}} = 4.807$ for Random Forest.

Since these are the first scores obtained using the evaluation data, there should be no overfitting, and we can reasonably assume that similar results can be obtained on the private score.

IV. Discussion

The proposed approach obtains results that far outperform the given baseline ($d_{\text{base}} = 6.629$). We have empirically shown that the selected models perform similarly for this specific task, achieving satisfactory results in terms of distance d. The following are some aspects that might be worth considering to further improve:

- The current hyperparameter tuning process has explored a limited set of parameters for the tree-based models. To enhance model performance we can expand the range of hyperparameters considered, and splitting criteria (mse and absolute error). Thus hyperparameter tunning can lead to further improvements in the model. Unfortunately, the constraints were the lack of high-memory and the absence of GPU.

- Considering the relatively huge dataset we were working on (with hundreds of thousands of data points), the potential benefits of employing more complex models using deep learning become even more pronounced. Deep learning architectures, such as artificial neural networks (ANNs) are designed to handle large-scale datasets. These models can leverage the data to capture hidden patterns and dependencies that might not be as effectively captured by traditional approaches. For this part, a hyperparameter tuning for a large ANN could lead to better results. Still, this is not guaranteed [3].

- A clustering-based approach could be beneficial here, and may lead to better results, this is done by clustering the (x, y) plane, and then any new data point will be evaluated first according to its possible cluster. Then, the evaluation can be made based on the characteristic of that cluster. This could reduce the feature space and provide more concise predictions.

References

[1] Jonathon Shlens, 'A Tutorial on Principal Component Analysis' arXiv:1404.1100 [cs.LG]
[2] D. Bertsimas, V. Digalakis Jr, 'Improving Stability in Decision Tree Models' arXiv:2305.17299 [stat.ML]
[3] L. Grinsztajn, E. Oyallon , G. Varoquaux, 'Why do tree-based models still outperform deep learning on tabular data?' arXiv:2207.08815 [cs.LG]
[4] Donna Xu et al. 'A Survey on Multi-output Learning' arXiv:1901.00248 [cs.LG]