Data-Driven Particle Position Estimation: A Comprehensive Machine Learning Approach for Two-Dimensional Space

Paolo Riotino, Christian D'Alleva
Politecnico di Torino
Student id: s332530, s331883
s332530@studenti.polito.it, s331883@studenti.polito.it

***Abstract*—In the field of particle and trajectory studies, the challenge associated with accurately determining positions finds a potential solution in a data-driven approach. Within this context, a machine learning pipeline has been developed with the aim of estimating particle positions in a two-dimensional space. The process, beginning with dataset exploration and concluding with the application of the regression model, involved comparing various methodologies. This comprehensive approach resulted in both satisfactory predictions and a deeper understanding of the domain of interest**

I. Problem overview

Exploring the trajectories of particles, particularly electrons, is a crucial aspect of particle physics. The Resistive Silicon Detector (RSD) serves as a valuable sensor for this purpose, featuring a 2-dimensional surface capable of detecting particle passages. When a particle traverses the sensor, 12 metallic pads come into play to gather information. Each occurrence of a particle passing through the sensor is termed an "event." During each event, the metallic pads measure signals characterized by specific features, which are in turn correlated with the precise position of the particle. The ultimate objective is to deduce the positions of particles by leveraging the information extracted from these signals.

A. Dataset

The supplied dataset pertains to a series of events, each defined by the data gathered from 18 pads. Every pad records five distinct measurements: `pmax`, `negpmax`, `area`, `tmax` and `rms`.

- **Pmax**: the magnitude of the positive peak of the signal, in mV.
- **Negpmax**: the magnitude of the negative peak of the signal, in mV.
- **Tmax**: the delay from a reference time when the positive peak of the signal occurs, in ns.
- **Area**: the area under the signal.
- **Rms**: the root mean square (RMS) value of the signal.

In each event, there are 18 readings for each feature, even though the sensor only has 12 pads. This discrepancy arises due to hardware limitations during data acquisition, resulting in a subset of the 18 features containing noise rather than actual readings.

B. Exploratory Data Analysis

1) **Attribute Types and Data Domain**: The dataset consists only of numerical data with various unit measures, as specified earlier. The range of feature values typically falls within the order of hundreds, except for `negmax` and `rms`, where the values are in the tens of thousands and units, respectively.
2) **Missing Values and Duplicate Records**: No missing values or duplicate records have been identified in the dataset.
3) **Data Distribution**: A variance analysis revealed that each feature exhibits a distinct order of magnitude. Further examination of data distribution highlighted that the `negpmax` feature contains extreme variance, primarily attributed to the presence of extreme outliers and a notable negative skewness. For the `tmax` feature, certain pads (0, 7, 12, 15, 16, 17) exhibit a distribution distinct from the others. A boxplot analysis indicates a substantial prevalence of outliers. To address this situation, seeking the opinion of a domain expert is recommended.
4) **Correlation**: Pearson correlation analysis indicates that certain pads have features that are not correlated with both others features and targets. Notably, `area` and `pmax` measured on the same pad exhibit a positive correlation.

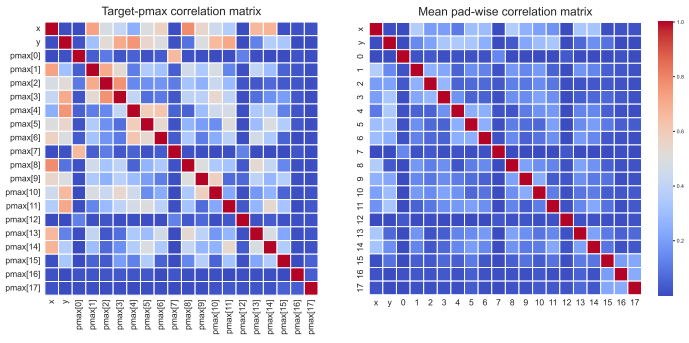

Fig. 1. Correlation matrices

II. Proposed approach

A. Preprocessing

Following the insights gained from the Exploratory Data Analysis, no action is required to eliminate records based on the absence of duplicates or missing values. The primary focus of preprocessing has centered on data transformation. Specifically, efforts were concentrated on feature reduction and scaling to ensure computational efficiency for the regression model.

1) **Removing unhelpful pads**: We decided to eliminate certain pads (0, 7, 12, 16, 17) for two reasons:
   a) As shown in Figure 1 where the `pmax` correlation matrix and mean correlation matrix are plotted, it is evident how those pads are completely uncorrelated, even with the targets.
   b) The model's performance would be negatively impacted by their presence.

   An exception is pad 15, which aligns with point a but not point b. Consequently, we opted to retain it. Nevertheless, we were aware, based on the problem description, that six pads were merely introducing noise into our data.
2) **Choosing between `pmax` and `area`**: We noticed that `pmax` and `area` were pair-wise correlated. Consequently, we opted to retain `pmax` and discard `area` since `pmax` demonstrates a stronger correlation with the target variables. This decision is further substantiated by a slight enhancement in the model's performance.
3) **Comparing Different Methods**: The feature selection process followed two paths.
   a) The first considered correlation and the feature importances index from the Random Forest.
   b) The second path utilized the `mutual information` statistic, implemented in sklearn [1]. This statistic provides a measure of interdependence between two features, and it was employed, using `mutual_info_regression`, to extract only the features with the strongest interdependence with the target variables.

   The results are explained below in Table I, $1^{st} Method$ refers to point a), $2^{nd} Method$ to point b).
4) **Scaling considerations**: We also saw that big differences in the magnitude of features could make our regression model not work well. So, we tried different ways to make sure the values are normalized. We compared methods like min-max scaling, Z-score normalization, and robust scaling from sklearn to find the best one for our model. In the end, we opted for standard normalization.

B. Model selection

Initially, a Random Forest Regressor was employed for feature selection and pruning based on feature importance. Subsequently, two closely related models, namely the RandomForestRegressor and the ExtraTreeRegressor, both rooted in Decision Trees, were investigated. These models exhibit high accuracy, adeptness in handling large datasets, and resilience to missing data. Notably, the dataset under consideration contains a considerable number of outliers, particularly within the selected features. To address this challenge and mitigate overfitting, robustness to outliers was identified as a critical requirement.

The RandomForestRegressor, chosen for its ability to reveal feature importance post-fitting, facilitates the identification of each feature's contribution to the model solution. This insight enables the pruning of features with minimal impact, thereby reducing computational complexity associated with extraneous features. The RandomForest models were configured with a Multi-Output Regressor to enhance accuracy and efficiency.

The utilization of two closely related models, despite their functional similarities, was motivated by the acknowledgment that their performances may differ in certain scenarios. This dual-model approach allows for a comprehensive evaluation, ensuring that the model with superior performance is identified in varying contexts.

C. Hyperparameters tuning

In the context of developing a predictive model, the development dataset was partitioned into training and validation sets, denoted as X (features) and y (targets), respectively. The objective was to assess the dissimilarity between the predicted y-values and the actual y-values in the validation dataset. Key attention was directed towards optimizing hyperparameters within the RandomForestRegressor. In Figure 2, a comparison of distances obtained from RandomForest and ExtraTreesRegressor during the hyperparameter tuning process is depicted.

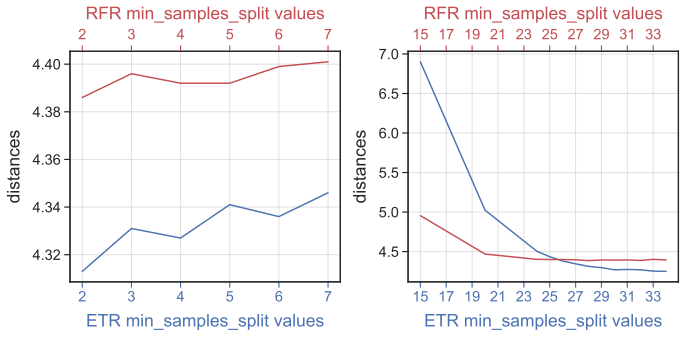

Fig. 2. RandomForest and ExtraTreesRegressors comparison

The hyperparameters under consideration included:

- **n_estimators**: Initially set to 100 to manage computational costs, subsequent experimentation involved increments up to 700 to enhance accuracy scores.
- **max_depth**: Commencing at 10, this hyperparameter was iteratively increased to 34 but still, being mindful not to overfit the model.
- **min_samples_split**: Initially adhering to the default value of 2, this hyperparameter was subsequently iteratively increased to 7.

- **max_features**: We tried different settings like `auto`, `sqrt` and `log2`.
- **criterion**: Various split evaluation functions, including `squared_error`, `friedman_mse`, and `poisson`, were evaluated.
- **bootstrap**: Experiments toggled the value between `True` and `False`.
- **random_state**: Fixed at 42 to ensure the replicability of the model, facilitating effective hyperparameter tuning.

For each combination of hyperparameter values, a model was trained, and predictions for y were generated. The ensuing evaluation involved calculating the euclidean distance on the development dataset. Through this iterative process, optimal hyperparameter values were determined based on minimizing the observed distance.

III. Results

Finally, after a comprehensive tuning process on the development dataset, we achieved a satisfactory alignment between the training and validation subsets. The best hyperparameters identified for the ExtraTreesRegressor for this problem are shown in Table I.

Hyperparam	1^{st} method		2^{nd} method	
	RandForest	ExtraTrees	RandForest	ExtraTrees
n_estimators	700	700	700	700
max_depth	29	29	29	29
max_features	auto	auto	auto	auto
min_samples_split	2	2	2	2
criterion	default	default	default	default
random_state	42	42	42	42
distance	4.830	4.746	4.750	4.734

TABLE I
BEST HYPERPARAMETERS

It is crucial to emphasize that iteratively increasing the number of estimators to 700 consistently yields incremental improvements in accuracy. However, due to constraints in computational capacity and time considerations, the experimentation was concluded at this specified value.

We conducted a comparison between our solution, one that randomly generated target values within the possible range, and a naive solution that utilized all available features along with a simple random forest with default hyperparameters, 100 estimators, and a maximum depth of 10. The obtained results are as follows:

- Random solution: 209.847
- Naive solution: 8.897

Furthermore, our model was trained on the entire development dataset without any splitting, aiming to enhance generalization on a larger dataset. Clearly, our solution outperforms the naive approach, benefiting from a thoughtful reasoning and a systematic process of improvement.

IV. Discussion

A. What we have accomplished

The pipeline we applied led us to interesting results both in the preprocessing phase and in the actual regression phase. Our comparative approach allowed us to justify the removal of certain features that proved to be unhelpful in achieving satisfactory predictions of the target variables. The experimental approach we adopted prompted us to reassess some choices regarding the selection of the most relevant features, while comparing two preprocessing methods highlighted how different combinations of features were suitable for the task.

During the forecasting phase, the adoption of two similar models yielded similar results, although not identical. The benefits of this comparative choice are numerous, with improved performance and greater robustness standing out, foremost among them.

B. Limitations encountered

The size of the problem and the computational complexity of the applied models are compounded by the constrained computational capacity at hand. Additionally, there is a restricted understanding of the domain related to the proposed problem.

C. Future developments

- Enhanced domain knowledge to improve feature selection.
- Enhance hyperparameter tuning leveraging improved computational capabilities.
- Explore different approaches such as GradientBoostingRegressor and Neural Network, to uncover underlying relationships and enhance both robustness and accuracy.

References

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.