Trajectory Analysis for Particle Position Detection by means of RSD Sensor Signals

Matteo Bracco, Umberto Piccardi
Politecnico di Torino
Student id: s319845, Student id: s331183
s319845@studenti.polito.it, s331183@studenti.polito.it

***Abstract*—This research aims to develop a predictive model for estimating the position (*x*,*y*) where the particle of interest passed. For this purpose an RSD sensor with 12 pads is given. Detection is performed by observing the particle's passage through the green area of the sensor and then by measuring signals on the pads.**

I. Problem overview

The field of particle physics is a thriving domain. Particle passage detection stands out as one of its fundamental tasks since the introduction of the first silicon microstrip detector in 1980 [1]. The device utilized in this project belongs to the same category.
The RSD sensor is a 2-dimensional surface with a green area at its core. Within this region, 12 pads are positioned, each capable of measuring signals emitted by the particles.
The analysis of these signals allows for the extraction of 5 features for each pad. Nonetheless, noise has to be taken into account, thus the number of initial features is far greater than 72 and needs to be trimmed down through preprocessing.
Following the preprocessing step, models with the best estimated results will be fitted with the data from these features to determine which model exhibits the best performance.
To build the model 2 datasets are provided: *Development.csv* and *Evaluation.csv*. The former contains 514,000 records divided into 92 columns: 2 for the coordinates *x*,*y* and 5 for each of the 18 pads. Its purpose is to fit the model. The latter has 385,500 records divided into 91 columns: 1 for the Id and 5 for each of the 18 pads. It constitutes the set of features for which the model has to predict the coordinates. The accuracy of the model is calculated by measuring the average Euclidean distance between the predicted coordinates and the actual ones.

II. Proposed approach

A. Preprocessing

This study exploits 2 comma-separated CSV files containing the 5 features of each 18 pads. The first step involved loading both files into 2 separate pandas dataframes [2], using the Python programming language [3]. The dataframes were found to have no null values, only numerical values and be already scaled without the need for adjustments or transformations.
Among the 18 pads, 6 comprise noise outliers. Detection of the 6 worst pads and pruning is achieved through the analysis of 2 plots. A *pmax* distribution plot and a correlation matrix. In the distribution plot, as shown in Fig. 1 [4] [5], the chosen

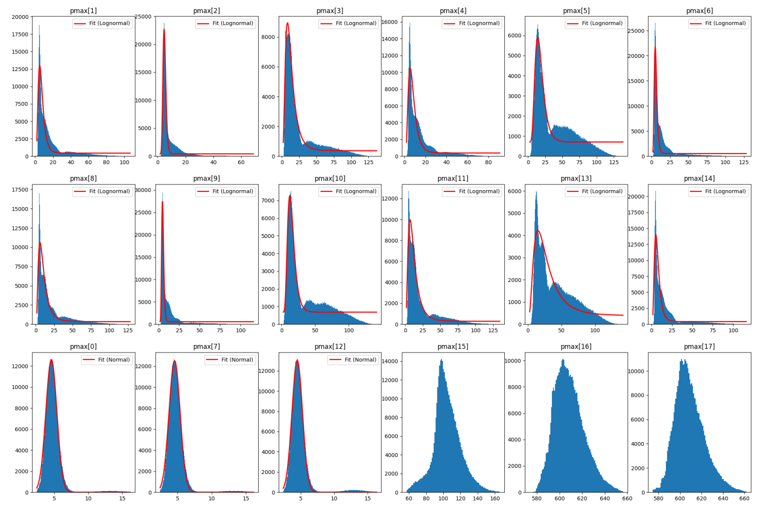

Fig. 1. *Pmax* Distribution

feature for the analysis is *pmax*, because it represents an interesting distribution. For each subplot, the most smooth function that, through interpolation, fitted the *pmax* distribution was chosen and displayed with a red line. The 12 best performing pads fitted a lognormal distribution function, while pads 0,7,12 only fitted a normal distribution and pads 15,16,17 showed a distribution so varied to not fit neither of them. Therefore, the 12 estimated non-outlier pads seem to be 1,2,3,4,5,6,8,9,10,11,13,14.
For further pruning and better understanding of the nature and association between the features, a correlation matrix was analyzed. To avoid 90 features on both the axes, a limit on the minimum absolute value for the correlation was imposed: 0.5. In this way, each feature without at least one correlation greater than the threshold, with a feature different from itself, was not displayed on the matrix. This is shown in Fig. 2 [4]. The correlation matrix confirmed the 12 selected best pads and established the need to prune them to obtain a better performance and a model with less noise.

B. Model selection

Since this is a multiple regression problem with a high number of features, we have initially investigated regression models that avoid imposing strict assumptions on their distributions, such as Decision Trees and Random Forest. In the

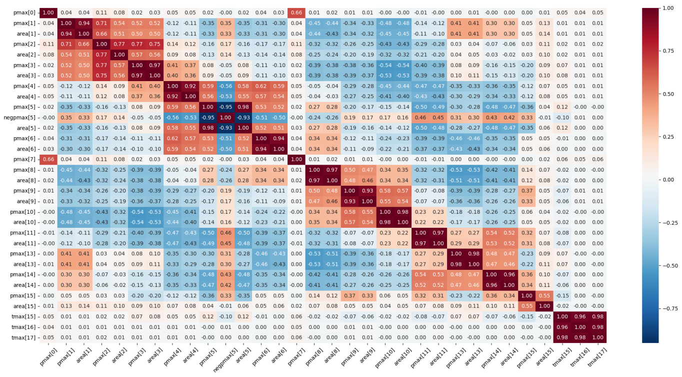

Fig. 2. Correlation Matrix

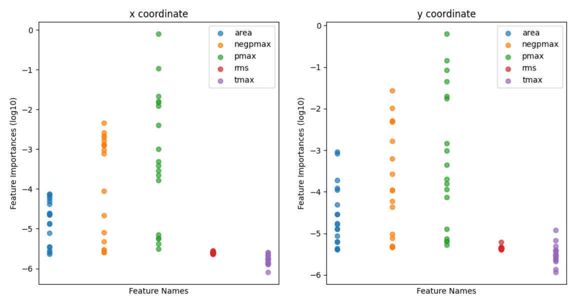

Fig. 3. Features Importance

initial phase, we trained various regression models using the *SciKit-learn* [6] library without pruning features. Our analysis is based on 3 different scores: $Adjusted\ R^2$, MAE, MSE.

Table I [4] shows that Extra Trees Regression and Random Forest Regression are the best performing algorithms. They are ensemble learning techniques that belong to the broader category of tree-based models. They leverage the strength of multiple decision trees to improve predictive performance, reduce overfitting, and enhance model robustness. The key difference lies in the degree of randomness introduced during the training process. Extra Trees's model introduces more randomness, making it potentially faster but also requiring more trees to achieve similar performances to Random Forests. Once these regression models were selected, we trained the Random Forest to extract the importance of the features. We started without pruning any feature and thus considering also the noisy pads. This first approach further confirmed our choice of pads. As we can notice in Fig. 3 [4], *pmax* and *negpmax* are the most significant features, while *tmax* and *rms* have much lower relevance. The values of *area* are in an intermediate level, but due to the high correlation between each respective *pmax* and *area*, we chose to prune them.

Then we fed the Random Forest and Extra Trees with all the *pmax* and *negpmax* and some combinations of the other components to find the ideal solution. Adding features of the 12 chosen pads didn't improve the solution. So, we turned back to the noisy pads and we found that adding the *pmax[15]* feature was linked with an improvement in the performance. The progress is visible in the table II [4].

C. Hyperparameters tuning

The two models were optimized using an exhaustive search method using the *ParameterGrid* from the *SciKit-learn* library [6]. To avoid overfitting, the model was trained using an hold out strategy: dividing the development set in train and validation set. The split was accomplished through *train_test_split* with $test\_size = 0.20$.

The initial tuning focuses on *max_depth*, *n_estimators*, *max_leaf_nodes*, *random_state*, *max_features*, *criterion*. For *max_leaf_nodes* the improvement obtained wasn't enough to justify its employment, whereas the models were performing much better with: $random\_state = 42, max\_features = "sqrt"$ and $criterion = "friedman\_mse"$.

The *n_estimators* hyperparameter refers to the number of individual decision trees that are built during the training process. It is useful to make the model more robust, but, according to Fig. 4 [4], it doesn't affect the accuracy in a meaningful way either for Extra Trees or for Random Forest.

On the other hand, the *max_depth* hyperparameter, which controls the maximum depth of each individual decision tree in the ensemble, displays a different behaviour between the two models. As portrayed in Fig. 5 [4], Random Forest already starts to decrease with lower values and stop to significantly increase its performance after a *max_depth* of 25, whereas Extra Trees need much higher values and its decrease continues also for higher depths.

TABLE I
MODELS' PREDICTIONS ON THE TRAINING DATA

Model	Adjusted R^2	MAE	MSE
ExtraTreesRegressor	0.999106	2.6282	12.1559
RandomForestRegressor	0.998765	2.939	16.8097
BaggingRegressor	0.998541	3.210	19.8735
DecisionTreeRegressor	0.996778	4.3552	43.9408
LinearRegression	0.984207	10.8743	215.8616

TABLE II
MODELS' PREDICTIONS ON THE TRAINING DATA AFTER FEATURE SELECTION

Model	Adjusted R^2	MAE	MSE
ExtraTreesRegressor	0.999121	2.6113	11.9560
RandomForestRegressor	0.998824	2.8808	16.0159
BaggingRegressor	0.998644	3.1062	18.4615
DecisionTreeRegressor	0.997043	4.1099	40.3471
LinearRegression	0.982062	11.3780	244.2752

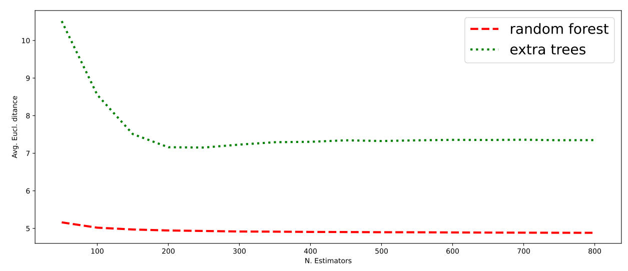

Fig. 4. *n_estimators* tuning

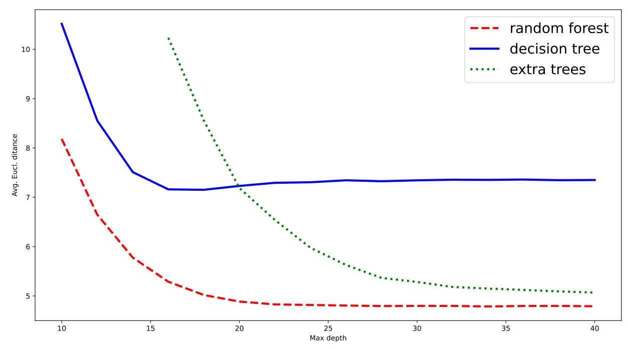

Fig. 5. *max_depth* tuning

III. Results

In the pursuit of our investigation, we have successfully discerned the trajectories of particles by leveraging the parameters extracted from the Resistive Silicon Detectors (RSD).

The role of *pmax* and *negpmax* proved crucial and far more relevant than any other feature. Even though the removal of the outlier pads improved the performance, we noticed that including *pmax[15]* had a positive effect on the average Euclidean distance of the model on average of 0.09. Therefore, we kept it.

The result obtained by means of Random Forest was 4.770 at most, with hyperparameters: *max_depth*: 30, *max_depth*: 'sqrt', *n_estimators*: 800, *random_state*: 42.

The adoption of the ExtraTrees regression model proved optimal as seen by the average Euclidean distance obtained: 4.589. This was rendered feasible by the last configuration of hyperparameters chosen: *max_depth*: 48, *max_depth*: 'sqrt', *n_estimators*: 750, *random_state*: 42, *criterion*: 'friedman_mse'.

IV. Discussion

As shown in Fig. 6, the actual coordinates we used to train the model (blue points) and the model predictions (red points) follow a similar pattern, proving the partial success of the model. An interesting property was discovered through this graph: the majority of points falls inside a region that coincides with the core of the RSD sensor green area and is surrounded

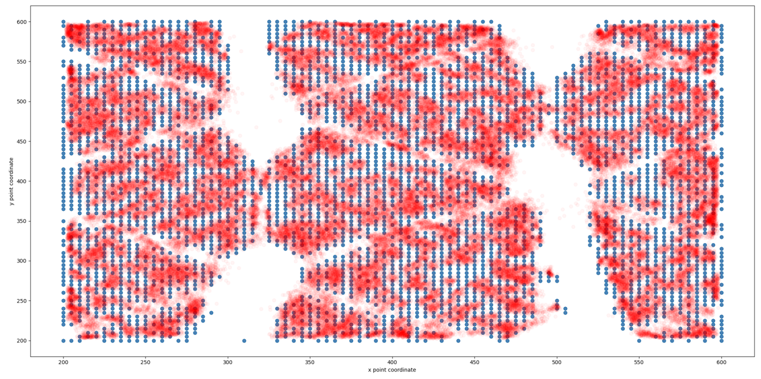

Fig. 6. Actual (blue) vs Predicted (red) (*x*,*y*)

by 5 pads, recognisable for the white hexagonal regions void of coordinates nor predicted neither trained.

Another fascinating finding is the strict correlation between *area* and *pmax*. For the non-outlier pads it's clear that the highest contribution for the *area* is given by the first peak, while the negative peak is usually not so relevant.

References

[1] M. Krammer, "Silicon detectors in High Energy Physics experiments," *Scholarpedia*, vol. 10, no. 10, p. 32486, 2015. revision #197033.

[2] W. McKinney, "pandas: a foundational python library for data analysis and statistics," *Python for High Performance & Scientific Computing*, vol. 14, no. 9, pp. 18–22, 2011.

[3] Python Software Foundation, *Python Software Foundation*, 2001–2022.

[4] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[5] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, Polat, Y. Feng, E. W. Moore, J. Vand erPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, A. Vijaykumar, A. P. Bardelli, A. Rothberg, A. Hilboll, A. Kloeckner, A. Scopatz, A. Lee, A. Rokem, C. N. Woods, C. Fulton, C. Masson, C. Häggström, C. Fitzgerald, D. A. Nicholson, D. R. Hagen, D. V. Pasechnik, E. Olivetti, E. Martin, E. Wieser, F. Silva, F. Lenders, F. Wilhelm, G. Young, G. A. Price, G. L. Ingold, G. E. Allen, G. R. Lee, H. Audren, I. Probst, J. P. Dietrich, J. Silterra, J. T. Webber, J. Slavič, J. Nothman, J. Buchner, J. Kulick, J. L. Schönberger, J. V. de Miranda Cardoso, J. Reimer, J. Harrington, J. Rodner, J. Kuczynski, J. O'Donnell, J. Burns, J. C. Oliveira, K. Tritz, M. Thoma, M. Edwards, M. Gori, M. Weber, N. Sumner, N. T. Aman, J. Rozbicki, O. Moloney, P. Vázquez-Benítez, P. Bidinger, P. Fuhrman, T. Grosskopf, R. Chen, R. Bubner, S. Hübner, S. Lyons, T. Feldbauer, T. O'Kane, W. Ning, W. Janning, W. Meyer, and S. v. d. Walt, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020.

[6] 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 Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.