ParticleTrack: Predictive Modeling for Particle Trajectory Reconstruction with Resistive Silicon Detectors

Mirkomil Egamberdiev
Politecnico di Torino
Student id: s315875
s315875@studenti.polito.it

Abdirashid Chorshanbiyev
Politecnico di Torino
Student id: s314800
s314800@studenti.polito.it

Abstract

In this paper we propose our approach to the problem of predicting the coordinates (x, y) of the particle trajectories within a Resistive Silicon Detector (RSD). First we focus on data exploration in order to manage possible issues in the dataset, then we compare the performance of basic regression models so as to choose a starting optimal estimator. Finally we improve our regressor through feature selection and we perform the predictions on the evaluation set.

I. PROBLEM OVERVIEW

Particle tracking within the field of particle physics is a fundamental task with significant implications for understanding the behavior and properties of subatomic particles, such as electrons. The challenge lies in accurately detecting and predicting the trajectory of these particles as they pass through specialized sensors. One such sensor, the Resistive Silicon Detector (RSD), offers a unique capability with its 2-dimensional surface and 12 snowflake-shaped metallic pads, enabling precise measurement of signals associated with particle events.

The core problem addressed in this project is the development of a robust data science pipeline to predict the (x, y) coordinates of particles based on the signals recorded by the RSD pads during each event. This predictive modeling task is crucial for advancing our understanding of particle interactions and contributes to the broader goals of particle physics research. The given dataset is a collection of 514,000 events consisting of signals recorded by the RSD pads. It is divided into two parts:

- the *development set* with 385,500 rows. This part is used for training and validation so it contains the target columns;
- the *evaluation set* with 128,500 rows. This part must be used as unseen data so it does not contain the target columns.

Each event is associated with a set of signals measured by all pads, from which several relevant features have been extracted. These features include the magnitude of *positive and negative peaks, time delays*, *area under the signal*, and the *rms value* of the signal. For each type of feature, there are 18 readings, resulting in a total of 90 attributes for each event. Therefore, we divided the features in categories for a better exploaration. However, among these attributes, 6 relate to noise features for each type due to the presence of 12 pads. After examining the datasets, it was found that there are no missing values. Hence, the task includes identifying and removing outliers, as well as recognizing and dropping the noise features.

	Development	**Evaluation**
.isna()	False	False

Table 1: Checking missing values

II.PROPOSED APPROACH

A. Preprocessing

In this section, we conduct data exploration to enhance the data quality. For each group of attributes, we generate boxplots to identify potential noise features from their distribution. For instance, the boxplots presented below effectively illustrate outliers for positive peak features and area features.

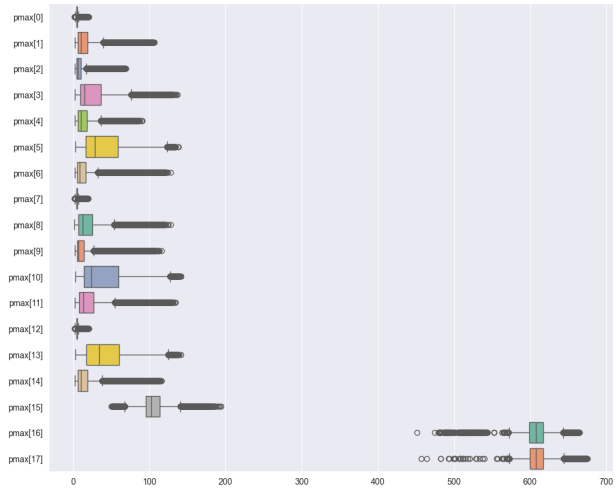

Figure 1: Boxplot of the *positive peak features* to determine noise

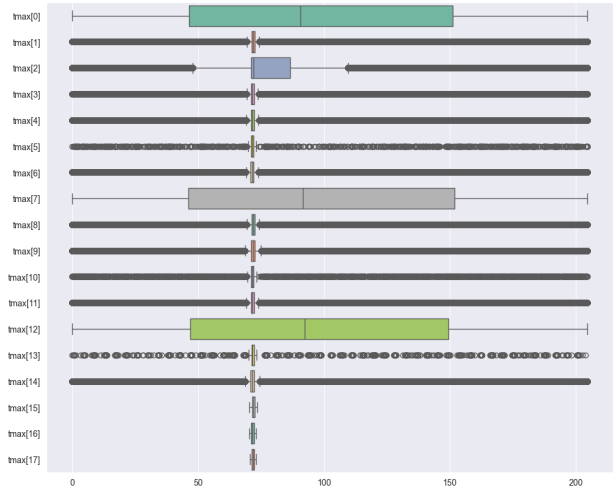

Figure 2: Boxplot of the *time delay features* to determine noise

As observed, features with indices 0, 7, 12, 15, 16, and 17 exhibit notably different distributions compared to others within *positive peak* and *time delay* feature groups shown in the **Figure 1** and **Figure 2** respectively. This pattern is consistent across other groups as well. Therefore, we identify these features as noise and proceed to eliminate them.

After getting rid of noise labels, we compute some statistics for each attribute that help us identify outliers along the row axis. For example, here is the output for certain features belonging to the category of positive peak features **Table 2**.

As observed for the provided features, there exist outliers where values beyond the 99th percentile significantly deviate from the rest of the data. These outlier values are indicative of anomalies that need to be addressed. To handle this, we utilize the boxplot method once again, which offers a convenient visualization of the distribution of numerical attributes[2]. By identifying outliers at both lower and higher percentiles, we aim to bring them into a more suitable range of values. It's important to note that we only address extreme outliers, as a certain level of noise can actually improve the model's generalization ability. Interestingly, we observe that retaining some outliers in the target feature leads to significantly better performance on the public leaderboard, despite a slightly lower score on the development set. This suggests that our model may have been overfitting to the development set.

	pmax[1]	pmax[8]
count	385500.000000	385500.000000
mean	16.510543	20.503279
std	16.781018	20.175681
min	2.028134	1.440403
1%	3.432467	3.506769
25%	5.619728	6.471845
50%	10.068513	12.660268
75%	19.031319	25.451260
99%	78.574631	90.952839
max	106.649066	128.185635

Table 2: Distribution of 'pmax[1]' and 'pmax[8]' labels

Another important step involves applying feature selection through correlation analysis and feature reduction using principal component analysis (PCA). For feature selection, we calculate the pairwise correlation among all attributes. For any pair of attributes with a correlation coefficient greater than or equal to 90%, we discard one of the two columns involved to reduce redundancy. In the following figure, we present a heatmap of the correlation matrix (**Figure 3**). Lighter cells indicate higher correlation between the corresponding features.

To apply PCA, we aim to determine the number of components that explain at least 95% of the total variance in the data. As depicted in the graph below, which illustrates the cumulative explained variance plotted against the number of components, we find that 31 components are required to explain 95% of the variance (**Figure 4**).

We found that feature selection and feature reduction techniques actually worsened our performance. Therefore, we decided to stick with the original set of features.

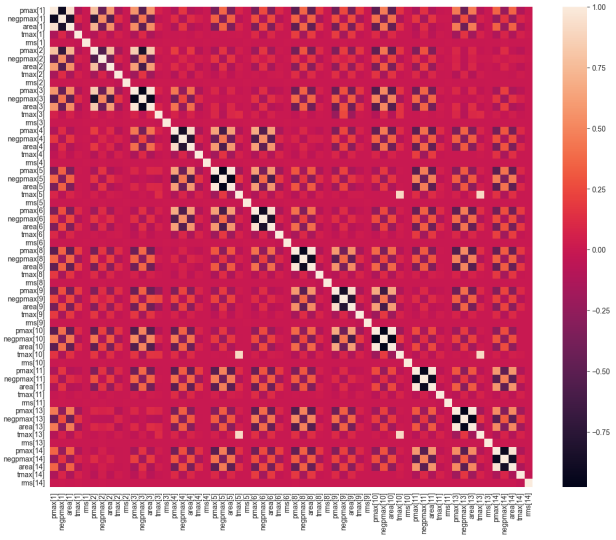

Figure 3: Heatmap of the correlation matrix

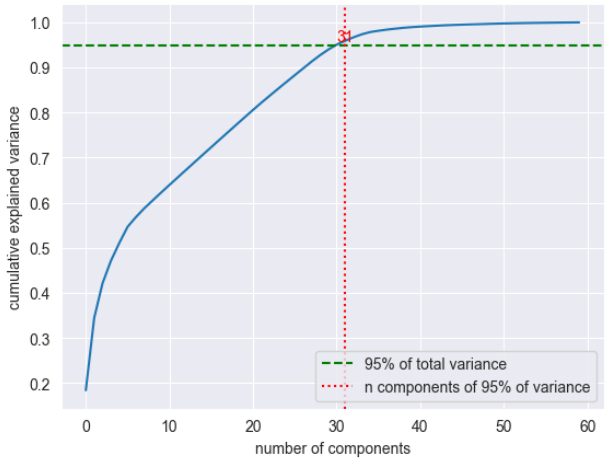

Figure 4: The graph of the cumulative explained variance as a function of the number of components.

B. Model Selection

After completing the data cleaning process, we begin basic model evaluation. The dataset is divided into three subsets: training, validation, and test sets. Model selection is performed using the training and validation sets only. Next, we standardize the features and assess the performance of five models using their default parameters: *Linear Regressor, Ridge, Lasso, Random Forest Regressor* and *Histogram based Gradient Boosting Regressor.* The evaluation is based on **average Euclidean distance** of our predictions from the targets[3].In other words, for all n events (x_1, y_1), (x_2, y_2), ... , (x_n, y_n) with predictions $(\hat{x}_1, \hat{y}_1), (\hat{x}_2, \hat{y}_2), ..., (\hat{x}_i, \hat{y}_i)$, the metric used will be

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

There are resulting scores:

Regression model	RMSE
Linear Regressor	14.309313
Ridge	14.309312
Lasso	14.983859
Random Forest Regressor	7.049470
HistGradient Boosting Regressor	6.095578

Since the *HistGradientBoostingRegressor* model achieves the best Root Mean Squared Error (RMSE) score, we have chosen to use this model to obtain the final result. *HistGradientBoostingRegressor* is a variant of the Gradient Boosting Machine (GBM) algorithm that is specifically optimized for efficiently handling large datasets. It introduces the use of histograms to improve training speed and memory efficiency compared to traditional GBM algorithms.

C. Hyperparameters tuning

In this step, we employ a grid search cross-validation technique to optimize the parameters of the *HistGradientBoostingRegressor* model. This involves testing all possible combinations of parameters to determine the best configuration. The dataset is divided into five folds, and for each fold, the model is trained on the remaining folds and evaluated on the current fold. The average score across all folds is computed for each parameter combination[1]. The parameters that are tuned and the corresponding tested values are as follows:

Hyperparameter	Values
max_iter	200, 500, 1000
max_leaf_nodes	200, 500, 1000
early_stopping	True, False

Where *'max_iter'* determines the maximum number of iterations or boosting stages the gradient boosting algorithm will run, *'max_leaf_nodes'* controls the maximum number of leaf nodes or terminal nodes in each tree of the ensemble and *'early_stopping'* prevents overfitting by monitoring the performance of the model on a validation set during training.

III.RESULT

The resulting best configuration is the one with *early_stopping* = False, *max_iter* = 500, *max_leaf_nodes* = 500. Now we fit the model on the development set and we make the prediction

on the evaluation set. The followings are respectively the obtained average Euclidean distance scores on the development set and on the public part of the evaluation set:

- development set score: 4.217
- public score: 4.698

IV.DISCUSSION

In conclusion, we employed various strategies to maximize the effectiveness of the pipeline. We found that the most critical aspect is the data preprocessing step, so we explored several techniques to enhance the dataset's quality. Additionally, we conducted feature selection to identify the most relevant ones for the problem. Ideally, a more thorough approach would involve selecting the best model through grid search cross-validation and tuning hyperparameters. However, we encountered computational limitations preventing the algorithm from converging. Nevertheless, based on our performance on the public leaderboard, we are satisfied with the presented pipeline.

References

[1] scikit-learn.org, "Comparing Random Forests and Histogram Gradient Boosting models." https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_hist_grad_boosting_comparison.html/

[2] github.com, "Regression Analysis Visualization.", https://juanitorduz.github.io/lm_viz/ (June 26, 2020).

[3] towardsdatascience.com, "9 Distance Measures in Data Science", https://towardsdatascience.com/9-distance-measures-in-data-science-918109d069fa.