Machine learning techniques for glitch detection in Planck/HFI data

Results

Paolo Galli

1 General outcomes and best model

Final results are reported in pages from 2 to 4. I do all tests on a 4-core desktop CPU.

Globally, the best algorithm turned out to be SVC when trained with the sorted dataset, with an accuracy of 0.9993 ± 0.0012, a training time of 0.77 s, and a rate of predictions of about 21 142 samples per second; lastly, SVC took 21 min and 8.05 s to finish the real-life test.

When trained with the sorted dataset, each algorithm reached at least a score of 0.99. Therefore, as the PCA had hinted, the sorting of the dataset helped in glitches recognition. Thus, limiting the analysis only to the sorted training dataset and comparing SVC with the other algorithms:

- Each algorithm has a comparable accuracy score within 2σ (see Table 3 and Fig. 1, page 2).
- SVC is not the fastest algorithm in terms of training time, although the difference is of about one or two tenths of a second (see Table 5, page 3).
- Excluding LGB[1], SVC is the fastest algorithm in terms of prediction time, closely followed by RFC; as expected, KNC is the slowest. See Table 5, page 3.
- In the real-life test, SVC turned out to be the fastest algorithm, closely followed by LGB. RFC and KNC took lots of days to finish the test. See Table 7, page 4.

[1]RFC and LGB prediction times need to be investigated because of some discrepancies in the obtained times: this will be done in Sec. 2.3 and 2.4.

Evaluation tables

Table 1: Models trained with the standard training dataset.

	SVC	KNC	RFC	LGB
Score	0.9805 ± 0.0063	0.9003 ± 0.0150	0.9143 ± 0.0113	0.9118 ± 0.0041

Table 2: Models trained with the augmented training dataset.

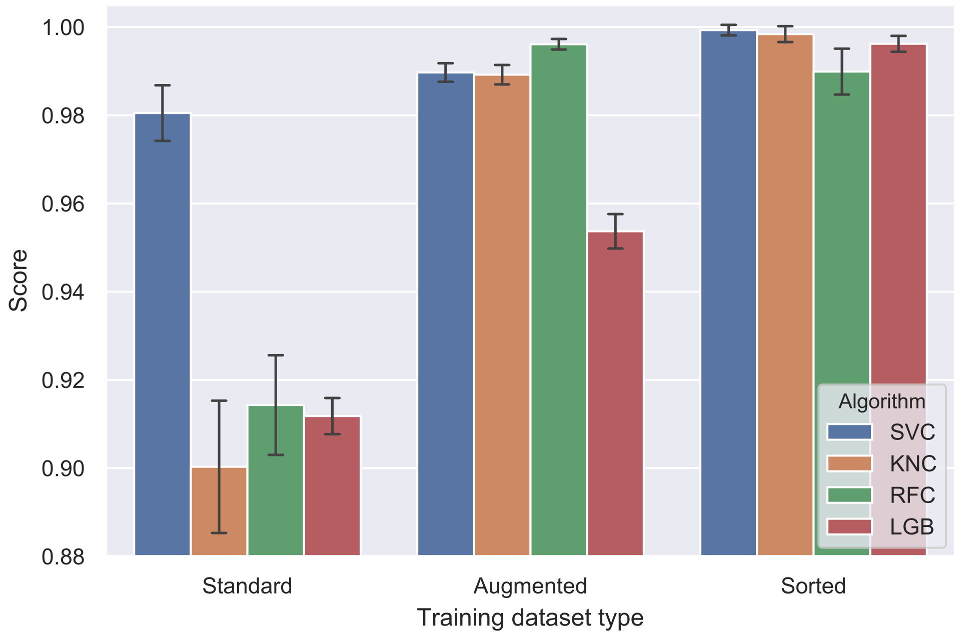
	SVC	KNC	RFC	LGB
Score	0.9897 ± 0.0021	0.9892 ± 0.0022	0.9961 ± 0.0012	0.9537 ± 0.0039

Table 3: Models trained with the sorted training dataset.

	SVC	KNC	RFC	LGB
Score	0.9993 ± 0.0012	0.9984 ± 0.0018	0.9899 ± 0.0052	0.9962 ± 0.0018

Evaluation plot

Figure 1: Score comparison between the four ML models.



Timing tables

Table 4: Models trained with standard and augmented training dataset.

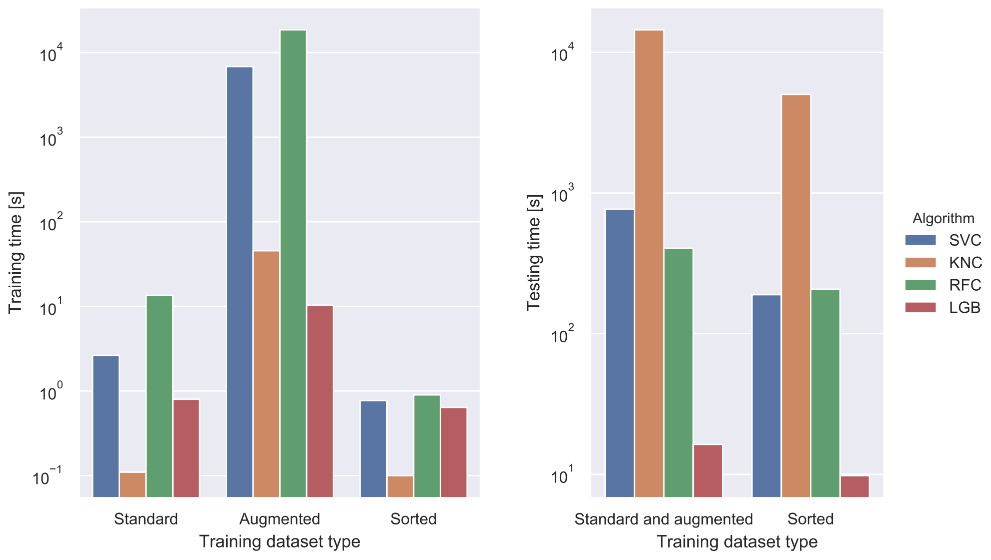
	SVC	KNC	RFC	LGB
Training time (standard dataset)	2.64 s (1 thread)	0.11 s (4 threads)	13.52 s (4 threads)	0.80 s (4 threads)
Training time (augmented dataset)	1 h 53 m 50.24 s (1 thread)	45.48 s (4 threads)	5 h 9 m 18.71 s (4 threads)	10.33 s (4 threads)
Testing time	12 m 47.23 s (1 thread)	4 h 0 m 39.05 s (4 threads)	6 m 44.65 s (4 threads)	16.33 s (4 threads)

Table 5: Models trained with the sorted training dataset.

	SVC	KNC	RFC	LGB
Training time	0.77 s (1 thread)	0.10 s (4 threads)	0.90 s (4 threads)	0.64 s (4 threads)
Testing time	3 m 9.20 s (1 thread)	1 h 23 m 32.29 s (4 threads)	3 m 26.80 s (4 threads)	9.79 s (4 threads)

Timing plots

Figure 2: Training and testing time comparison between the four ML models. Note the logarithmic scales on the y-axis.



Real-life test tables

Table 6: Models trained with the standard training dataset.

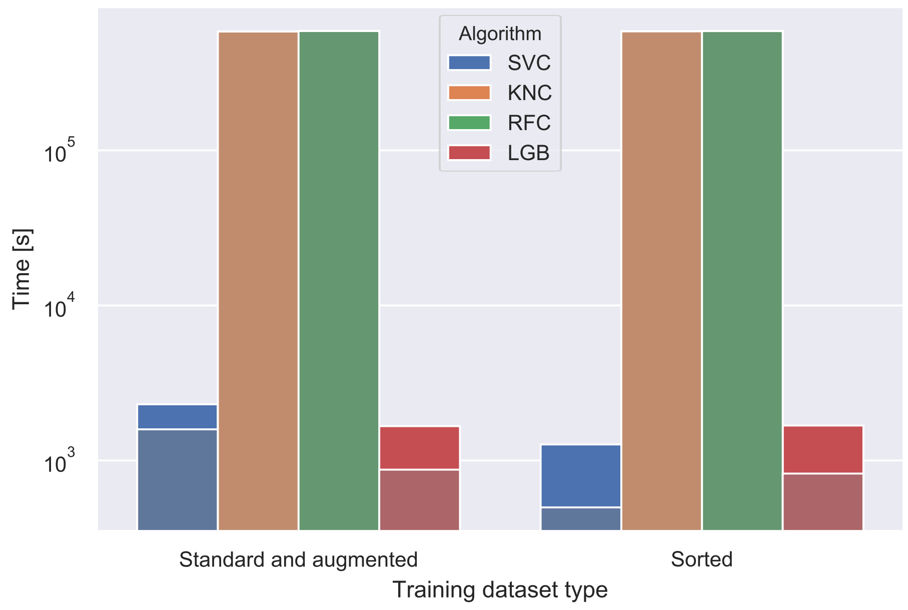
	SVC	KNC	RFC	LGB
Model time	26 m 27.21 s (1 thread)	6 d 18 h 20 m 54.57 s (4 threads)	6 d 19 h 29 m 5.18 s (4 threads)	14 m 32.23 s (4 threads)
Total time	38 m 22.22 s (1 thread)	6 d 18 h 46 m 23.87 s (4 threads)	6 d 20 h 9 m 27.41 s (4 threads)	27 m 43.87 s (4 threads)

Table 7: Models trained with the sorted training dataset.

	SVC	KNC	RFC	LGB
Model time	8 m 17.89 s (1 thread)	6 d 18 h 43 m 17.68 s (4 threads)	6 d 19 h 19 m 5.91 s (4 threads)	13 m 42.72 s (4 threads)
Total time	21 m 8.05 s (1 thread)	6 d 19 h 13 m 18.17 s (4 threads)	6 d 19 h 58 m 32.62 s (4 threads)	27 m 59.18 s (4 threads)

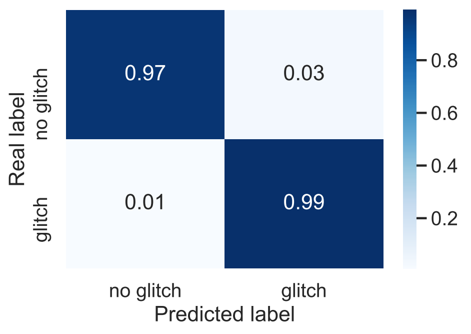
Real-life test plot

Figure 3: Real-life test comparison between the four ML models. Light colors refer to the model time; saturated colors refer to the total time. Note the logarithmic scales on the y-axis.

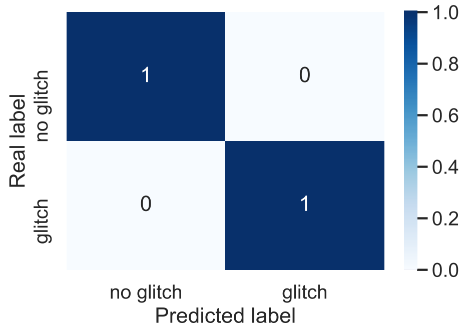


SVC evaluation plots

Figure 4: Confusion matrices.

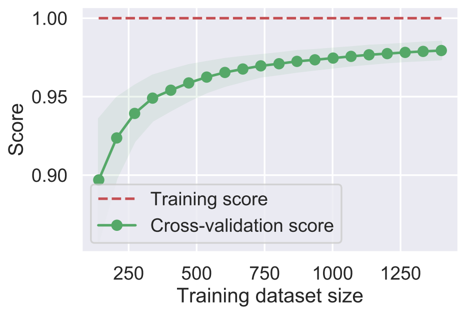


(a) Standard model.

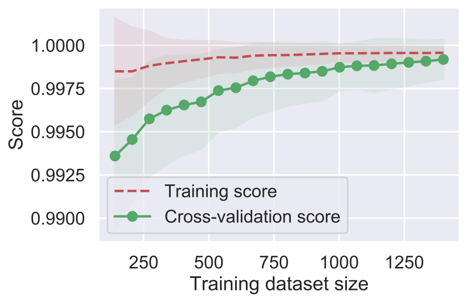


(b) Sorted model.

Figure 5: Learning curves.

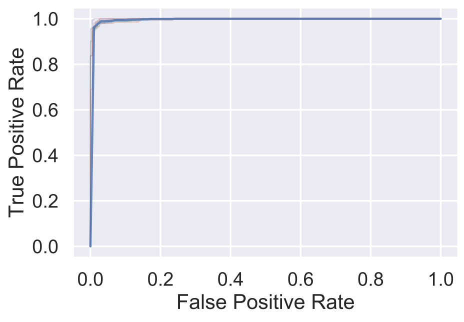


(a) Standard model.

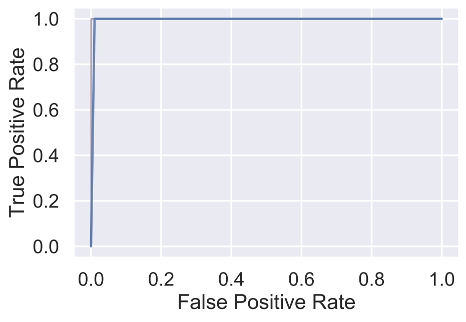


(b) Sorted model.

Figure 6: ROC curves.



(a) Standard model.

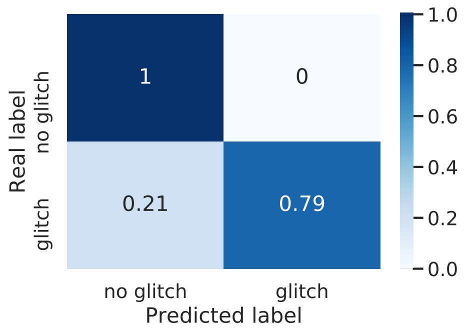


(b) Sorted model.

KNC evaluation plots

Figure 7: Confusion matrices.

(a) Standard model.



(b) Sorted model.

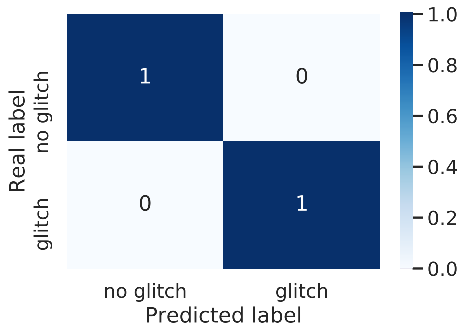
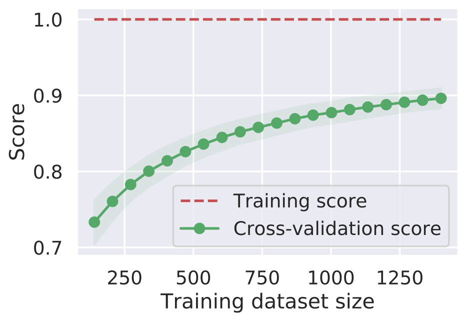


Figure 8: Learning curves.

(a) Standard model.



(b) Sorted model.

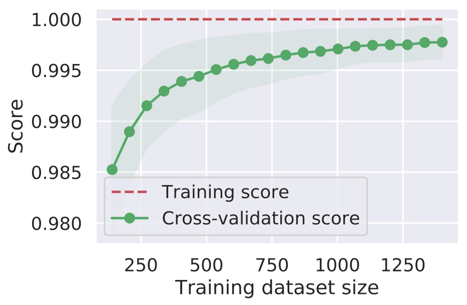
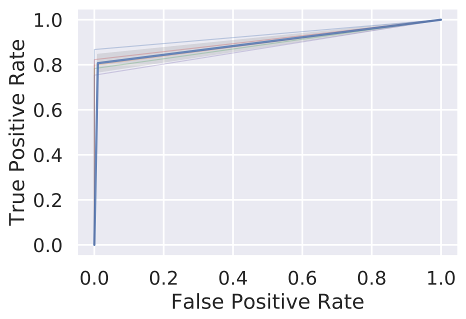
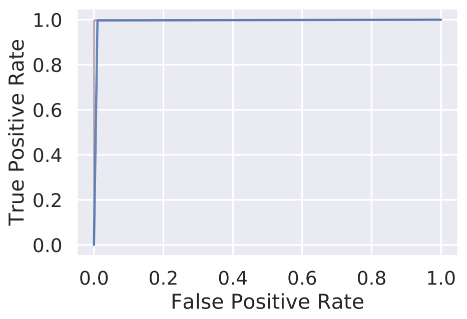


Figure 9: ROC curves.

(a) Standard model.

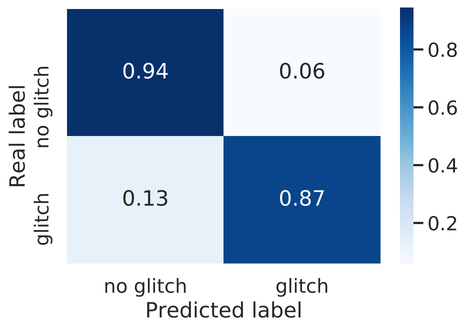


(b) Sorted model.

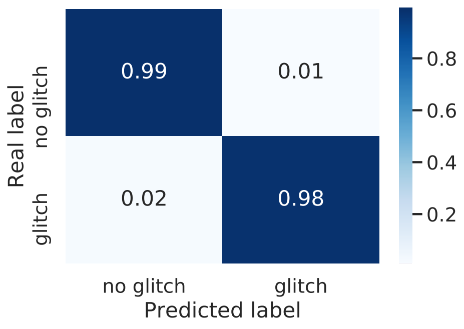


RFC evaluation plots

Figure 10: Confusion matrices.

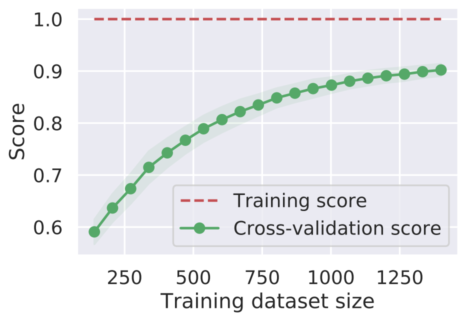


(a) Standard model.

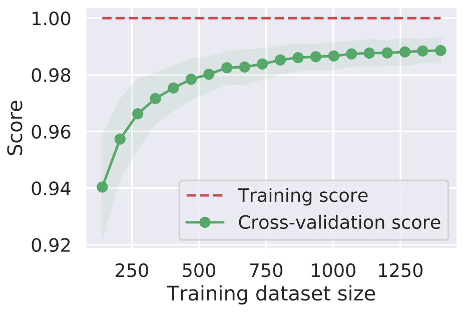


(b) Sorted model.

Figure 11: Learning curves.

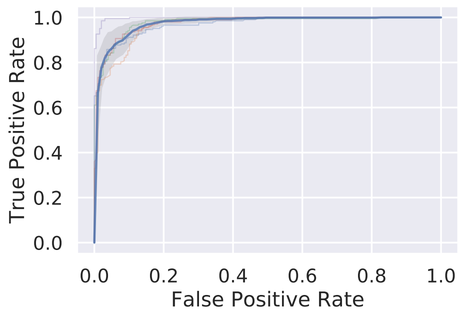


(a) Standard model.

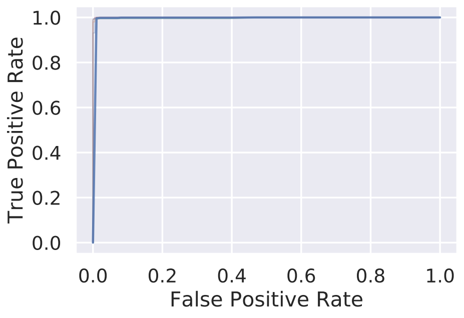


(b) Sorted model.

Figure 12: ROC curves.



(a) Standard model.

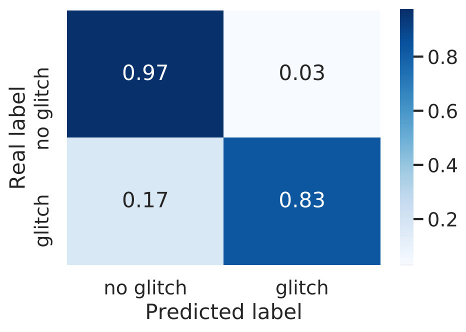


(b) Sorted model.

LGB evaluation plots

Figure 13: Confusion matrices.

(a) Standard model.



(b) Sorted model.

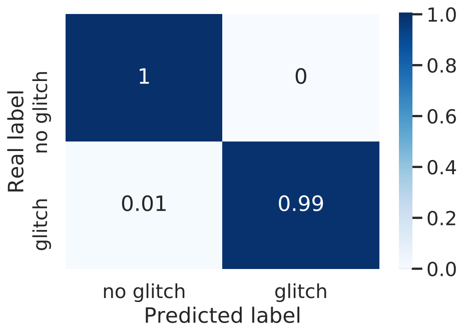
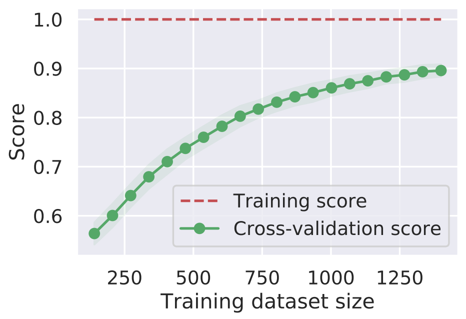


Figure 14: Learning curves.

(a) Standard model.



(b) Sorted model.

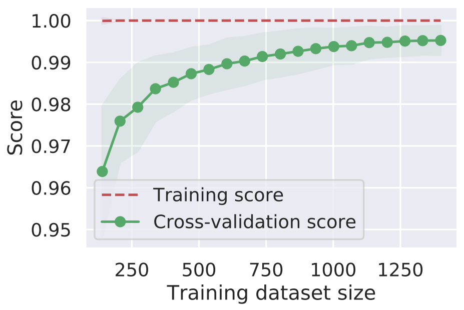
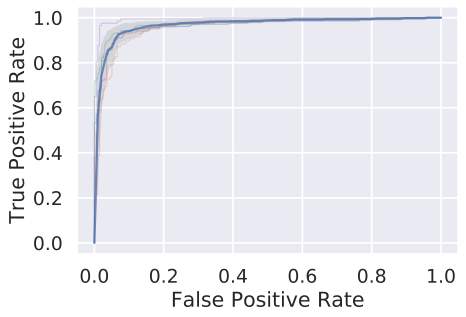
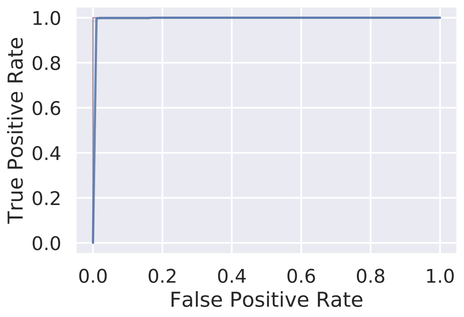


Figure 15: ROC curves.

(a) Standard model.



(b) Sorted model.



2 Results per model

Here I refer to a model trained with the x training dataset with *x model*, e.g., a standard model is a model trained with the standard training dataset. I only list the hyperparameters which value is different from the default value[2].

2.1 Support-Vector Machine

SVC evaluation plots can be seen on page 5. Best hyperparameters sets:

- Standard and augmented models:
  - `kernel = 'rbf'`
  - `gamma = 0.0151`
  - `C = 1.45`
- Sorted model:
  - `kernel = 'linear'`
  - `C = 0.15`

As said in the previous section, the SVC sorted model turned out to be the best ML model for glitches classification both for evaluation and timing results.

SVC results in having high accuracy scores also when trained with the standard training dataset (although the relatively small number of samples), and data augmentation only slightly increases the accuracy score: it means that the results obtained with the sorted model probably cannot be obtained with a standard model unless utilizing an enormous number of training samples. This behavior is confirmed by the learning curve 5a.

From the confusion matrix 4a, it's clear that the model behavior is the wanted behavior: when it fails, it tends to predict false-positives rather than false-negatives.

The training and testing times are always minor for the sorted model: this is due to the different kernel functions. Indeed, the sorted model uses a linear kernel, which means that there is no modification to the dataset space. The standard model uses instead a radial-based functions kernel, which slows the algorithm.

In the real-life test, the sorted model took only twenty minutes to analyze data from three bolometers and fifteen ODs, an excellent result.

[2]For HPs default values, refer to the packages' version in Sec. 3.1.

A negative point of the algorithm, intrinsic to its functioning, is the impossibility of being natively parallelized on multiple cores. Nevertheless, by using multithreading libraries such as `threading`, `multiprocessing`, or `mpi4py`, it is possible to parallelize predictions employing multiple classifiers. I used the `threading` library to do some tests on an 8-core CPU, obtaining a significant performance increase: this should be considered in future analysis.

2.2 K-Nearest Neighbors

KNC evaluation plots can be seen on page 6. Best hyperparameters sets:

- Standard and augmented models:
  - `n_neighbors = 1`
  - `algorithm = 'ball_tree'`
  - `leaf_size = 10`
  - `p = 8`
- Sorted model:
  - `n_neighbors = 1`
  - `algorithm = 'ball_tree'`
  - `leaf_size = 10`
  - `p = 3`

Final results are a fast train and slow predictions. Furthermore, the confusion matrix 7a reveals that KNC tends to predict false-negatives rather than false-positives when it fails: this is the opposite of the wanted behavior.

For these reasons, KNC is not the best algorithm for the classification of cosmic rays glitches. Although its scores are comparable to SVC scores (except for the standard model), KNC is a slow model: the shorter training time does not compensate for this difference, especially considering that SVC is a single-threaded model and KNC a multi-threaded model. The real-life test is indicative of that.

2.3 Random Forest

RFC evaluation plots can be seen on page 7. Best hyperparameters sets:

- Standard and augmented models:

  - `n_estimators = 400`
  - `min_samples_split = 3`
  - `bootstrap = False`

- Sorted model:
  - `n_estimators = 200`
  - `max_depth = 40`
  - `bootstrap = False`

RFC exhibits an opposite behavior if compared to other models: the model achieved its best accuracy score when trained with the augmented dataset. A hypothesis can be made by looking at the learning curves in Fig. 11. The LC 11a has a marked tendency to rise even at the end of the plot, indicating that the standard dataset is too small: indeed, if trained with the augmented dataset (200 000 samples vs. the 2000 samples of the standard and sorted training dataset), RFC reaches a higher accuracy score. The LC 11b shows the same behavior, although in a much less marked way. Probably, the number of samples in the augmented dataset made the difference.

As KNC, RFC too tends to predict false-negatives rather than false-positives when it fails (see confusion matrices in Fig. 10, page 7).

The long training time is due to the high number of meta-classifiers in the model: 400 for the standard and augmented models, 200 for the sorted model. A CPU with a higher core number can reduce the training time and boost the predictions rate.

Real-life test results are unexpected. As can be seen from Tables 4 and 5 on page 3, RFC has a comparable or shorter testing time than SVC. However, RFC took more than six days to complete the real-life test. The reason is probably in the method adopted for the test (see Sec. **??**): maybe, the prediction instruction is optimized for a large input dataset. Indeed, the input dataset utilized in the measurement of the testing time contains 2000 samples, while, in the real-life test, it contains only one sample.

2.4 Light Gradient Boosting Machine

LGB evaluation plots can be seen on page 8. Best hyperparameters sets:

- Standard and augmented models:
  - `learning_rate = 0.112`
  - `min_data_in_leaf = 7`

  - `num_leaves = 30`

- Sorted model:

  - `learning_rate = 0.177`
  - `min_data_in_leaf = 24`
  - `num_leaves = 120`

LGB reaches accuracy scores comparable to SVC only when trained with the sorted dataset, but turned out to be the fastest model, with incredibly low training and testing times (see Table 4 and 5, page 3). Indeed, LGB is a ML algorithm developed an optimized to handle and analyze large datasets. Despite that, real-life test results are comparable to those of SVC: as for RFC, the hypothesis is that the prediction instruction is optimized for a large input dataset, and consequently, the model is slower when used with a small input dataset.

Like KNC and RFC, it tends to predict false-negatives rather than false-positives when it fails (see confusion matrices in Fig. 13, page 8).

Thanks to the high accuracy score and the low training and testing times, LGB could potentially be a useful ML algorithm for glitches detection, if accurately studied: indeed, there is a need to reduce the number of false-negatives to reach the same performance of SVC. Furthermore, LGB is an advanced ML algorithm with lots of integrated features, hyperparameters[3], and evaluation metrics.

3 Conclusions

In this work, I investigated the possibility of recognizing cosmic rays signals in Planck/HFI data using machine learning techniques, powerful tools for analyzing and finding correlations among data. I examined four machine learning algorithms: Support-Vector Machine, K-Nearest Neighbors, Random Forest, and Light Gradient Boosted Machine.

I demonstrated that machine learning could effectively address the problem of cosmic rays glitches in data acquired from space missions that use bolometers.

Between all the analyzed models, *Support-Vector Machine Classifier* (SVC) turned out to be the best machine learning algorithm for glitches detection in Planck/HFI data, thanks to the combination of high accuracy scores, low

[3]LGB has lots of hyperparameters to tune: a tuning algorithm especially useful for the optimization of the LGB algorithm is `hyperopt`.

false-negatives rate, low training and prediction times, and excellent results in the real-life test. Even *Light Gradient Boosted Machine Classifier* (LGB) has the potential to become a valuable algorithm for the detection of glitches, but there is a need for more studies.

This preliminary study could open to a real possibility of utilizing these techniques in future space missions analysis pipelines. This work cover only one point of all the glitches analysis: my models can only find if a sample contains one or more glitches without removing them or reporting their position in the sample. However, models like sorted SVC and sorted LGB are accurate and fast (see tables and figures on pages from 2 to 4). They can be used to do a global scan of the data and pass only the positive samples to a more sophisticated algorithm (like a neural network), which has the task of actually removing the glitches from the sample; this can help to speed up the data analysis process.

3.1 Notes about used implementations

In this work I used the Anaconda Distribution environment with Python 3.7 and the following ML implementations:

- SVM: `SVC` (SVC) from package scikit-learn 0.21.3.
- KNN: `KNeighborsClassifier` (KNC) from package scikit-learn 0.21.3.
- RF: `RandomForestClassifier` (RFC) from package scikit-learn 0.21.3.
- LightGBM: `LightGBM` (LGB) from package lightgbm 2.3.0.