UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI FISICA "ALDO PONTREMOLI"
Corso di Laurea triennale in Fisica (L-30)

Machine learning techniques for glitch detection in Planck/HFI data

Tesi di laurea di Paolo Galli
Matricola 885692

Relatore: Prof. Maurizio Tomasi

Anno Accademico 2018-2019

Contents

1 Introduction **5**

2 The Cosmic Microwave Background **7**
2.1 The discovery 7
2.2 The first light of our universe 8
2.3 The CMB polarization 10
2.4 Preset and future missions and experiments 12
2.5 Bolometers and cosmic ray glitches 13
2.5.1 Classification of glitches in Planck/HFI 14
2.5.2 Detection of glitches in Planck/HFI 15

3 Machine Learning **17**
3.1 Introduction 17
3.2 The typical ML workflow 18
3.2.1 Preprocessing of data and feature engineering 18
3.2.2 Model building 19
3.2.3 Model evaluation 20
3.2.4 Model optimization 23
3.2.5 Predictions on new data 24
3.3 ML algorithms used in this work 24
3.3.1 Support-Vector Machine 25
3.3.2 K-Nearest Neighbors 25
3.3.3 Random Forest 26
3.3.4 Light Gradient Boosting Machine 26
3.3.5 Notes about used implementations 27

4 ML glitches classification in Planck/HFI data **29**
4.1 Preprocessing of data 29
4.2 Classification 33
4.3 Building and evaluation of ML models 34
4.3.1 Preprocessing of data and model building 34

4.3.2 Validation methods 36
4.3.3 Optimization method 37
4.3.4 Timing methods 37

5 Results **41**
5.1 General outcomes and best model 41
5.2 Results per model 49
5.2.1 Support-Vector Machine 49
5.2.2 K-Nearest Neighbors 50
5.2.3 Random Forest 51
5.2.4 Light Gradient Boosting Machine 52

6 Conclusions **53**

Chapter 1

Introduction

The Planck spacecraft is the most recent mission for the study of the Cosmic Microwave Background, whose study is significant for understanding the birth and the evolution of the Universe. Planck/HFI (High-Frequency Instrument) measured this radiation utilizing bolometers, which, however, are also sensitive to cosmic rays: when one of these impacts on a detector, a peak (*glitch*) appears in the outcoming signal, which must be discarded in the analysis. The complication is that these peaks have an extremely high recurrence in the data: considering samples of 100 consecutive measurements, about half of them have at least a glitch. In future space missions, like LiteBIRD, the number of bolometers will be significantly larger. Therefore, it is necessary to find fast and efficient glitch detection techniques.

My work aims to detect glitches using machine learning algorithms, which, if well trained, could be high-speed and powerful tools for this type of operation.

Chapter 2

The Cosmic Microwave Background

Detected only fifty years ago, the Cosmic Microwave Background (CMB) has become an interesting object of study, thanks to its importance in understanding and quantitatively describing the history of the Universe. Indeed, by looking at this radiation, we can observe how our universe was only 375 000 years after the Big Bang: so, the CMB is one of the most important keys to understand the evolution of the Universe.

In this chapter, I first describe the origin of the CMB and its main properties. Then, I explain how we are studying the CMB, focusing on the most recent space mission, Planck, and the problems encountered: in the context of this work, I am interested in the problem of cosmic rays, which can cause glitches in the signal measured by CMB space instruments that use bolometers, and how this problem has been faced.

2.1 The discovery

The CMB was predicted for the first time[1] by George Gamow in 1946 [14] as a consequence of the Big Bang Model proposed by Friedmann and Lemaître. If in the past galaxies were closer than now, then at some moment their distance should have been so small that they were virtually in contact: at this time, the matter was ionized and in thermal equilibrium. This radiation should follow a black body spectrum of about 5000 K but, due to the expansion of the Universe, today it would be detected as a *microwave radiation of a few Kelvin*. However, this prediction remained unconfirmed for two decades.

[1] Robert Dicke predicted cosmic matter radiation at $T < 20\,\mathrm{K}$ in the same year, but he did not refer to any background radiation. [11]

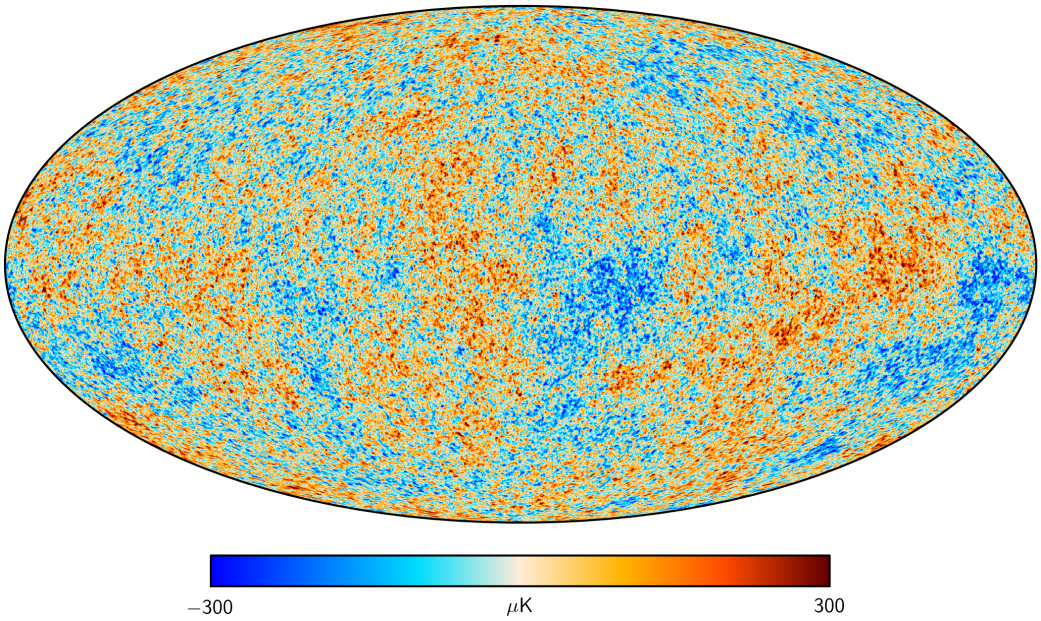


Figure 2.1: The CMB temperature anisotropies measured by the Planck satellite (ESA). The analysis of these anisotropies permits high-precision measurement of cosmological parameters and leads to a better understanding of phenomena that occurred in the early Universe.

The first detection[2] was carried out by Arno Penzias and Robert Wilson in 1964-1965 [22]: they measured the CMB as a noise in the signal of their antenna. Although they did not understand its nature at first, this noise was finally recognized as the relic radiation described by Gamow [10].

2.2 The first light of our universe

According to the Standard Cosmological Model, our universe originated about 14 billion years ago from a mathematical singularity called *Big Bang* and began to expand into a hot and dense plasma made by matter and radiation in thermal equilibrium. In the first 10^{-12} s, the Universe underwent a rapid and short-lived exponential expansion that enlarged it and brought quantum fluctuations in the primordial plasma at macroscopic scales: this period is called *inflation* and lasted for only 10^{-33} s or so. After a few seconds, the decreasing temperature and kinetic energy of particles caused by the progressive expansion of the Universe permitted the nucleosynthesis of hydrogen (at

[2] In 1941, Andrew McKellar detected an average bolometric temperature of 2.3 K [21], but Gamow's theory about the presence of a relic radiation did not exist yet.

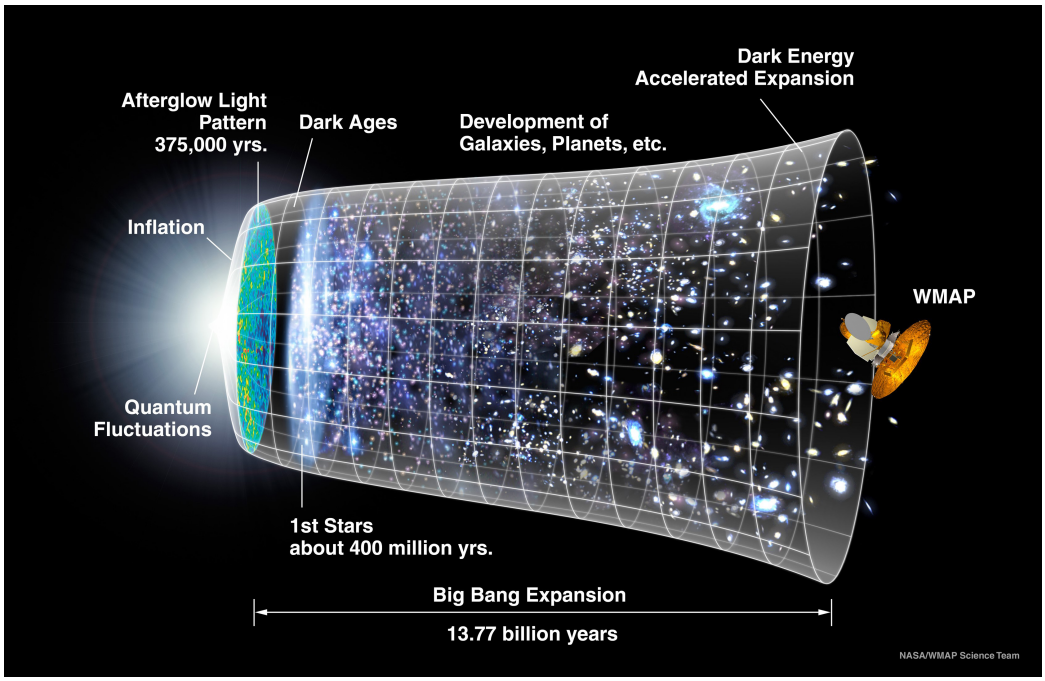


Figure 2.2: The evolution of the Universe in the Standard Cosmological Model. Representation made on the occasion of the WMAP mission (NASA).

$T \approx 10^{11}\,\mathrm{K}$) and helium (at $T \approx 10^{9}\,\mathrm{K}$). For the next 375 000 years, matter and radiation remained in thermal equilibrium thanks to the Thomson scattering[3] between photons and electrons. In the meantime, the Universe continued its expansion regularly, and therefore its temperature steadily decreased. This phase ended when the temperature of the Universe dropped to $T \approx 3000\,\mathrm{K}$. Indeed, this temperature is low enough to permit the combination of nuclei and electrons and thus permitted the formation of neutral atoms. The lower density of free electrons made the matter transparent to the radiation, and photons began to propagate freely with little or no interaction with matter[4]: this moment is called *last scattering epoch*, and the first escaped radiation is the *Cosmic Microwave Background*. Due to the Doppler effect caused by the expansion of the Universe, now we observe this radiation in the microwave with an almost perfect black body spectrum of $T \approx 2.73\,\mathrm{K}$.

[3]Thomson scattering is a classical elastic scattering between radiation and charged particles.

[4]The Thomson cross-section is proportional to m^{-2}, where m is the mass of the particle. Therefore, the interaction is governed almost by the lower-mass particle (e.g., electrons). The substantial reduction in the free-electron density after the last scattering epoch caused the collapse of the total cross-section, and thus the increase of the mean free path to a value higher than the size of the Universe.

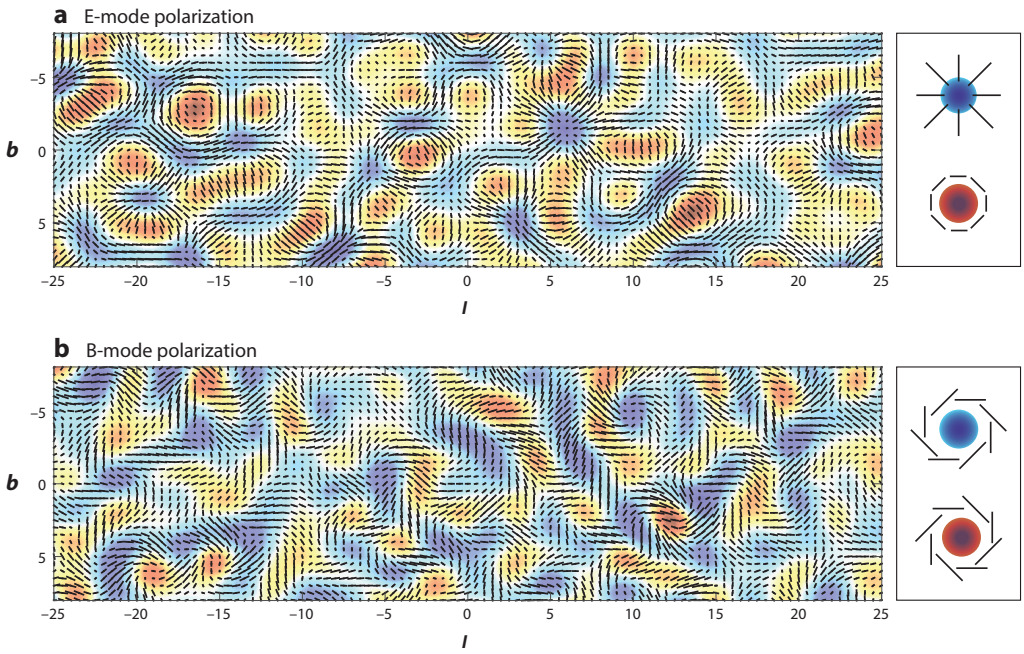


Figure 2.3: Representation of E-mode and B-mode polarization [19]. Note that, if reflected across a line going through the center of the pattern (i.e., under a parity transformation), E-mode patterns remain unchanged, while B-mode patterns swap their orientation.

One of the most significant characteristics of the CMB is its high homogeneity. This uniformity is explained by the inflationary paradigm: the exponential expansion proceeded so fast that, at the very first stages, kept together regions that today are no longer in causal contact.

2.3 The CMB polarization

In addition to the temperature anisotropies (see Fig. 2.1, page 8), another interesting property of the CMB is its polarization.

The CMB can be polarized if the radiation has a *quadrupolar configuration*[5]. Such configuration happens when the radiation field around an electron has a variation on an angle of 90°. Formally, polarization arises if and only if the radiation field possesses a quadrupolar component; quadrupoles are defined mathematically as the Laplace's spherical harmonics Y_ℓ^m

[5]Note also that the Thomson scattering preserves the polarization direction of the incoming radiation.

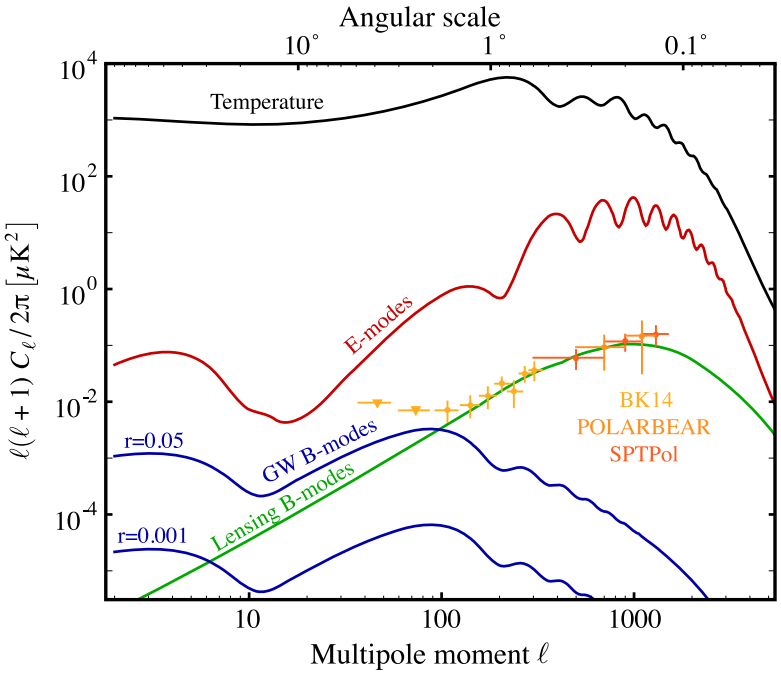


Figure 2.4: Theoretical predictions for temperature, E-mode, and B-mode polarization power spectra [1]. Today we have high-precision measurements of temperature and E-mode spectra. However, although the lensing spectrum has been measured (yellow and orange dots with their error bars), no experiment has detected the primordial GW B-modes spectrum because of its faintness.

with $\ell = 2$: so, there are five possible configurations, corresponding to $m = 0, \pm 1, \pm 2$. In 1997, Hu and White summarized the known explanations of the origin of each quadrupolar configuration [17]:

- Adiabatic pressure oscillations (sound waves) produce quadrupoles with $m = 0$;
- Vorticity in velocity fields causes Doppler shifts, which in turn produce quadrupoles with $m = \pm 1$;
- Gravitational waves (GW) stretch the wavelengths of photons, producing quadrupoles with $m = \pm 2$.

All of these effects polarize the CMB radiation. The first two effects are scalar perturbations (e.g., energy density perturbations) and produce the *E-mode* polarization pattern; the last effect is a tensorial perturbation (GW) and

produces the *B-mode* polarization pattern[6]. These patterns are sketched in Fig. 2.3, page 10. The CMB polarization fields would be a superimposition of these two modes, with different directions and wavelengths.

The GW contribution to polarization has significant importance because its measure can definitively prove the inflationary paradigm. Indeed, inflation predicts the amplification of the microscopic GW background originated in the first stages of the Universe expansion to macroscopic (cosmological) scales, thus potentially making it detectable. The predicted wavelength is too long for actual interferometers, but the effects of these GW on the CMB polarization can be measured as a tensorial perturbation. Thus, a measurement of B-mode polarization in the CMB is a direct measurement of the amplitude of tensorial perturbations, measuring the energy scale of relic GW.

As the CMB polarization took place during the matter-radiation decoupling, we can estimate that approximately 10% of the CMB signal should be polarized [5]. Therefore, measuring this polarization is challenging, especially for GW B-modes (see Fig. 2.4, page 11). For the latter, the figure shows that it is possible to measure GW B-modes only at large angular scale (small multipole moment ℓ); this motivates the use of spacecraft experiments alongside ground-based experiments: indeed, a satellite can explore a larger area thanks to the absence of the horizon and so observe the CMB at large angular scale.

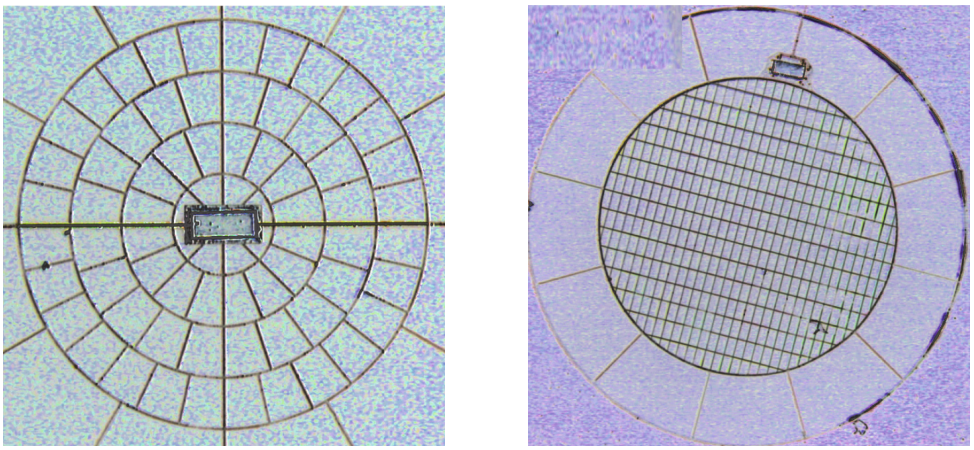
2.4 Preset and future missions and experiments

As sketched above, now the primary goal for CMB experiments is to detect GW B-modes in CMB polarization.

The next-generation space mission aimed at studying the Cosmic Microwave Background is *LiteBIRD*, the abbreviation of *Lite (Light) satellite for the studies of B-mode polarization and Inflation from cosmic background Radiation Detection* [16] [26]. It will observe the sky in radio and microwave frequencies (from 34 GHz to 448 GHz) with two telescopes. Each telescope is equipped with a half-wave plate system for polarization signal modulation and a focal plane filled with polarization-sensitive bolometers. Its main goal is to detect GW B-mode polarization patterns in the CMB at large angular scales[7]. It will be launched by the Japanese Space Agency (JAXA) in the middle of the 2020s.

[6] For a technical explanation of E-mode and B-mode polarization, refer to [19].

[7] The required angular coverage corresponds to $2 \leq \ell \leq 200$, where ℓ is the multipole moment (see Sec. 2.3).



(a) Spider-web bolometer. **(b)** Polarization-sensitive bolometer.

Figure 2.5: Planck/HFI bolometers.

Furthermore, also ground-based experiments are searching for gravitational waves B-modes in CMB polarization. Among the CMB ground experiments that are currently, the most important are *Adv-ACT* [13], *BICEP and Keck Array* [18], *CLASS* [12], *POLARBEAR and Simons Arrays* [27], and *SPT* [15]. Besides these missions, other experiments are planned and are under construction: *ALI* [20], *CMB-S4* [2] (one of the larger experiment), *LSPE* [28], *QUBIC* [4], and *Simons Observatory* [3] (which competes in size with CMB-S4).

2.5 Bolometers and cosmic ray glitches

Until today, three space missions have studied the Cosmic Microwave Background radiation: COBE, WMAP, and Planck.

COBE (Cosmic Background Explorer) was the first spacecraft launched to study the CMB and operated from 1989 to 1993 [7]. *WMAP (Wilkinson Microwave Anisotropy Probe)* succeeded COBE and operated from 2001 to 2010 [6]. Both these missions used radiometer-based technology in their detectors.

Planck is the most recent space mission and operated from 2009 to 2013 [9]. It considerably improved the WMAP measurements and produced the most detailed maps of the CMB anisotropies and polarization ever created until now. Planck consisted of two instruments: LFI (Low-Frequency Instrument, from 30 to 70 GHz) and HFI (High-Frequency Instrument, from 100 to 857 GHz), operating in a total of nine bands.

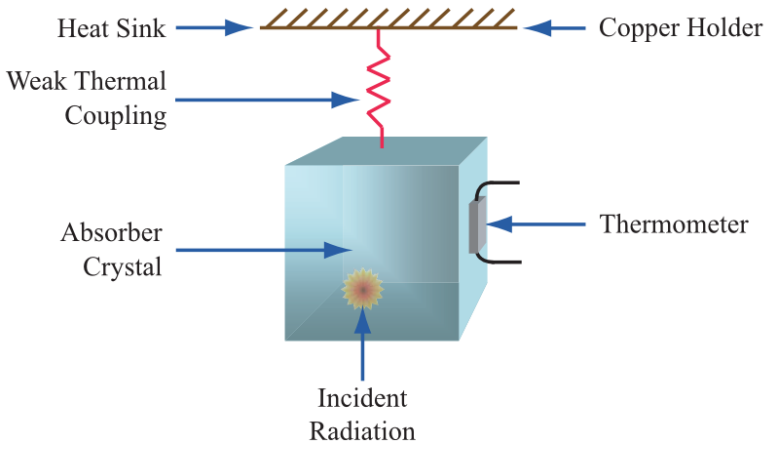


Figure 2.6: Scheme of a bolometer. The three principal components of a bolometer are the absorber crystal, the thermometer, and the heat sink.

Unlike the two previous spacecraft, Planck/HFI was the first space-based instrument that used *bolometers* instead of radiometers (see Fig. 2.5, page 13). Bolometers are phonon-mediated thermal detectors with finite response time to changes in the absorbed optical power: they measure the power of incident radiation via the heating[8] of a material with a temperature-dependent electrical resistance. The minimum excitation energy is proportional to $\kappa_B T$. So, by cooling bolometers to $T \approx 1\,\mathrm{K}$ or lower, it is possible to reach high energy resolution.

This choice helped in the improved sensitivities but caused a problem. Indeed, bolometers are sensitive to cosmic rays: when one of these impacts the spacecraft structure or a detector, a peak appears in the outcoming signal. This peak is called *glitch*.

2.5.1 Classification of glitches in Planck/HFI

The Planck Collaboration found three glitch families[9] in HFI data: short, long, and slow [24] (see Fig. 2.7, page 15).

- *Short* glitches are due to events in the optical absorbing grid or Ge thermistor. They show a rise time much shorter than the sampling period of 5 ms, followed by a fast decay. They are the highest-amplitude events.

[8] The bolometer heating is caused by the energy transferred from the radiation to the bolometer.

[9] The source of glitches in HFI is dominated by Galactic cosmic rays [24].

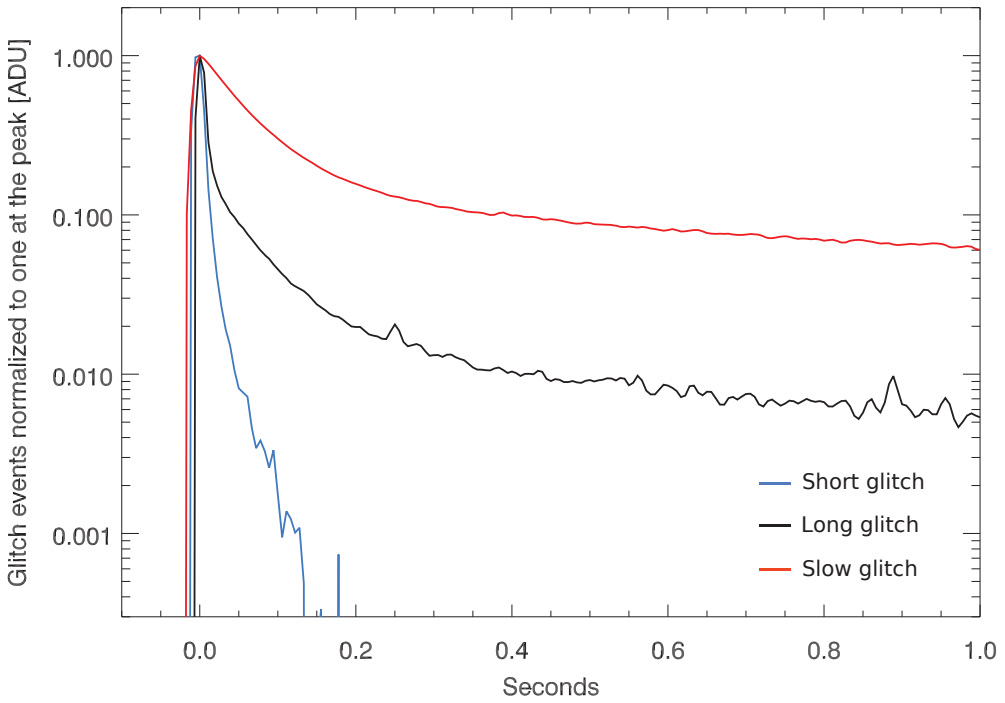


Figure 2.7: Examples of the three glitch families in Planck/HFI data [25].

- *Long* glitches are due to energy absorption events in the Si die. Like short glitches, the rise time is shorter than the sampling period, but the tail has a much larger amplitude. They are, on average, of lower amplitude than short glitches, but their rate is more than an order of magnitude higher: most of the glitches in the HFI data are these long events.
- *Slow* glitches have no explanation yet. They have a rise time much slower than the two other glitch families, and the tail is similar to that of a long glitch.

The population of glitches in HFI is unusual and significantly higher than in the other instruments: taking samples of 100 consecutive data, about half of these have at least a glitch inside. So, to use HFI data for scientific purposes, it is first necessary to identify and remove these glitches.

2.5.2 Detection of glitches in Planck/HFI

First, the Planck Collaboration modeled glitches with a sum of three or four exponentials: different types of glitches can be distinguished from each other

by the relative amplitudes and time constants of these exponential terms. The detection and removal of glitches were carried out using an *RMS (Root Mean Square) threshold algorithm* on a sliding window of 1000 samples and then performing a fit of the amplitudes of short and long glitch templates for all detected events in the window.

A problem in this method is the choice of the RMS threshold for the glitch identification: in HFI data, there are samples with glitches with a computed RMS value smaller than the maximum RMS value obtained for samples without glitches. So, the RMS threshold had to be chosen as the best compromise between false event detection and completeness of glitch detection. On LiteBIRD, this problem would be potentially bigger: indeed, LiteBIRD will use roughly 4000 bolometers (Planck/HFI had only 48 bolometers). There is a need to find a more scalable and straightforward detection method.

My work aims to find a fast, accurate, and scalable glitch detection method using various machine learning algorithms.

Chapter 3

Machine Learning

In the last years, machine learning (ML) had become a useful tool to analyze and extract hidden information and correlations from big datasets. Business companies use ML to analyze and plan marketing campaigns, predict consumer choices, and recommend their products. In science research, ML is useful for analyzing data, recognizing patterns, and extracting predictive models. Accordingly, ML algorithms are promising techniques for facing the problem of glitches in Planck/HFI data and, potentially, in future LiteBIRD data.

In this chapter, I first explain the basic concepts of ML and its main workflow. Then, I briefly describe the ML algorithms used in this work and the ideas under their implementation.

3.1 Introduction

Machine learning algorithms are mathematical and statistical models that can *find correlations* between data that usually humans can't find, because of the complexity of these correlations or the high amount of data to be analyzed. These algorithms learn from examples (the *training dataset*), creating a predictive model, and apply what they have learned to new and unseen events. So, one of the most powerful features of machine learning is the ability to *generalize* [8].

A ML model predicts a *target* variable Y from a set of *input features* $\boldsymbol{X}$ by creating a model. In other words: given the relation

$$Y = f(\boldsymbol{X}) + \mathcal{E}$$

between the input features $\boldsymbol{X}$ and the target Y, the goal and challenge of ML is to estimate f accurately while ignoring the noise $\mathcal{E}$.

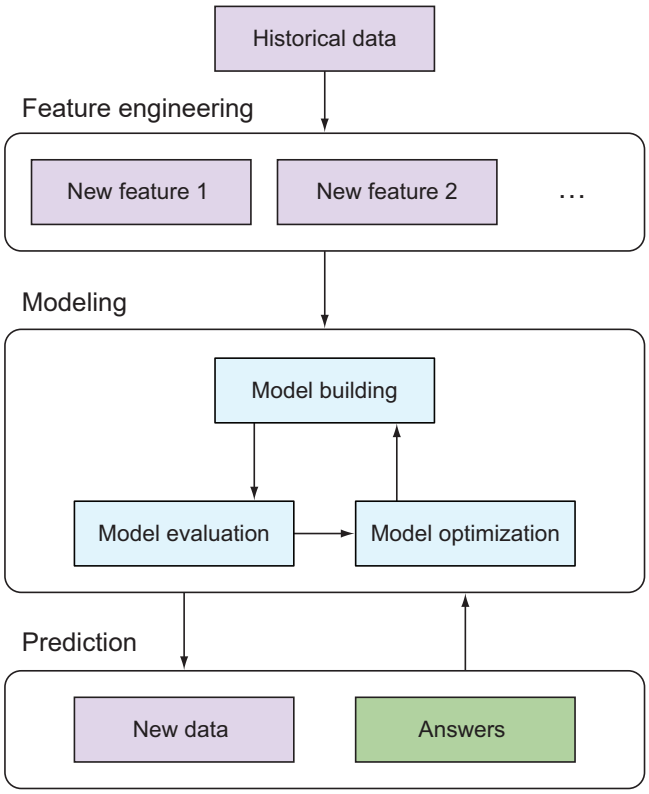


Figure 3.1: The main workflow of a machine learning model [8].

3.2 The typical ML workflow

The ML workflow has five main components: preprocessing of data and feature engineering, model building (or training), model evaluation, model optimization, and predictions on new data (see Fig. 3.1, page 18). Here I give only a brief explanation for each step; for a detailed discussion, see [8].

3.2.1 Preprocessing of data and feature engineering

This is one of the most critical parts of the entire workflow: indeed, the ML model results highly depend on training data.

The training dataset is the basis for every ML algorithm: historical data is used to learn and predict the best choice for each new and unseen input data. The training dataset consists of a table of N_{features} columns and N_{samples} rows: features are metrics that capture the relevant aspects of each considered event, called sample. The higher the number of samples, the greater

generalization power the model can have. Features can be of two types:

- *Numerical data* (e.g., age, instrument measurements), if values are numbers and represent a measure of something.
- *Categorical data* (e.g., the gender, the day of the week), if they can only take discrete values.

Sometimes, relevant information can be extracted from apparently uninformative features. It is possible to use the existing features to create new features that increase the prediction capability of the original data. This process is called *feature engineering*.

3.2.2 Model building

The ML model should be chosen on the type of problem that needs to be addressed. There are two main classifications for ML algorithms:

- Based on the functional form of f:
  - *Parametric models*. These models assume that f takes a specific functional form. The drawback is that most real-life problems do not have a simple form and so this assumption can be wrong.
  - *Non-parametric models*. These models don't make any assumption on the functional form of f: the form and complexity adapt to the complexity of data.
- Based on the training dataset labeling:
  - *Supervised machine learning*. The training dataset has a target variable[1] (labeled dataset). The two major categories of supervised machine learning models are:
    - *Classifier models*, which classify data into a discrete set of outcomes.
    - *Regression models*, which predict a numerical measurement.

[1]For example, if the goal is recognizing two different flowers A and B depending on their features (e.g., the length of the sepals, the number of petals), the training dataset can also contain the flower type. The ML model learns that the flower A has a typical set of features $\boldsymbol{X}_A$, and the flower B has a typical set of features $\boldsymbol{X}_B$. So, giving a set of features $\boldsymbol{X}$, the ML algorithm can predict the flower (the target) using the model created from the set of features $\boldsymbol{X}_A$ and $\boldsymbol{X}_B$.

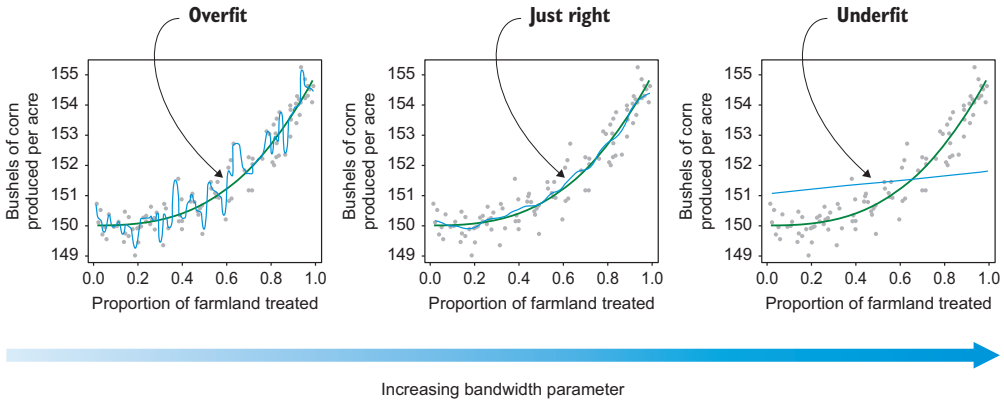


Figure 3.2: Three fits of a kernel-smoothing regression model to a corn production training dataset [8]. In the first case, the model has learned both the functional form f and noise $\mathcal{E}$: the model is overfitted. In the second case, the model has learned the trend f while ignoring almost all the noise $\mathcal{E}$: this model is well-fitted and can be used to predict events. In the third case, the model has not learned the trend f: the model is underfitted.

- *Unsupervised machine learning.* There is no identified target in the training dataset (unlabeled dataset). The two major categories of supervised machine learning models are:
  - *Clustering models*, which use input features to discover natural groupings in the dataset.
  - *Dimensionality reduction models*, which transform input features into a smaller number of new features that capture most of the variability of the dataset.

After the choice, the model needs to be built and trained on the training dataset to obtain a working model.

For the problem faced in this work, I used *binary supervised classification models*. Indeed, glitches shape is known, and so it is possible to teach the model to recognize glitches from a training dataset containing examples of glitches and non-glitches.

3.2.3 Model evaluation

Since I worked with classifier algorithms, I explain the validation metrics typically used for evaluating these models.

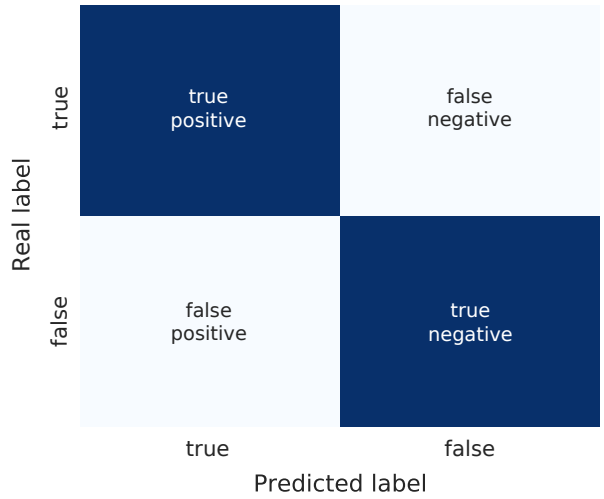


Figure 3.3: Example of a binary confusion matrix.

Cross-validation

The precision of a ML model can be evaluated with lots of metrics. The most common is the *accuracy*, computed as the ratio between the number of correct predictions and the total number of predictions:

$$Accuracy = \frac{N_{\text{correct}}}{N_{\text{TOT}}} \quad \in \quad [0, 1]$$

Some models fit the training dataset perfectly but fail when they are used to make predictions on new data. This lack of generalization is known as *overfitting* and occurs when the ML model also learns the noise $\mathcal{E}$ in the training dataset (see Fig. 3.2, page 20). The model is not predictive because noise is random.

Accordingly, the accuracy of the model predictions should not be evaluated on the training dataset. Indeed, using this dataset also for evaluation purposes isn't indicative of the model performance on new data and can lead to being too much optimistic about the real performance.

The solution is *cross-validation* (CV), a set of techniques that rigorously manipulates the training dataset to evaluate how the accuracy will be on new data. One of the best CV methods is *k-fold cross-validation*. First, the training dataset is randomly split into k disjoint subsets. Then, the model is trained on all folds except one: this one is used to test the model. After cycling all k folds, predictions are aggregated to assess accuracy. Note that CV methods (including k-fold method) assume that the training dataset forms a representative sample of the population.

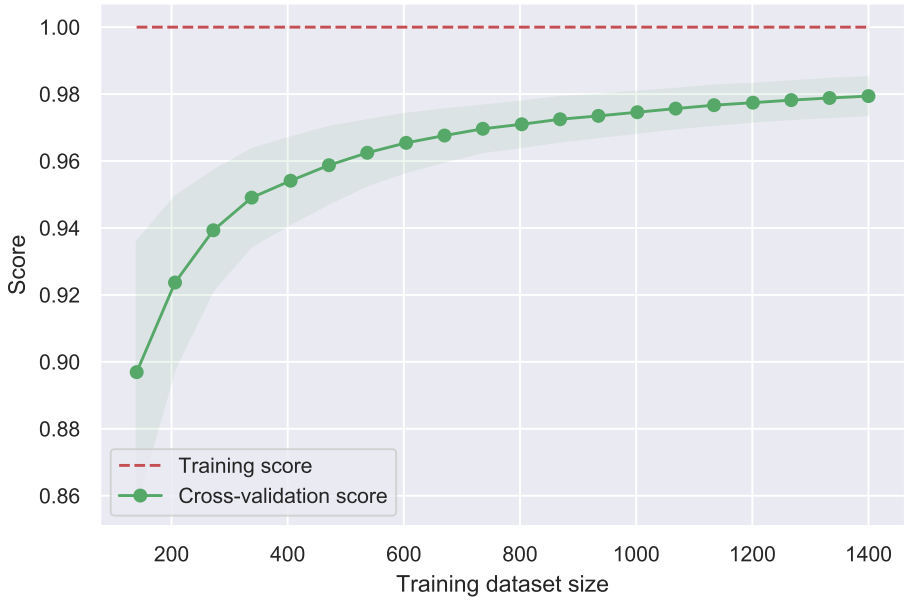


Figure 3.4: Example of a learning curve. From this plot, it is clear that enhance the size of the training dataset won't produce significant advantages.

Confusion matrix

In many classification problems, it's useful to look at this class-wise accuracy or class confusion. For this type of evaluation, a useful tool is the *confusion matrix* (CM). In the confusion matrix, rows are the real labels, and columns are the predicted labels; each element in the matrix shows the class confusion between the classes. So, the confusion matrix can help in evaluating how the model fails.

To understand, consider a binary classification - "true" or "false". A ML model can fail predictions in two ways: with a false-positive (prediction "true", real value "false") or with a false-negative (prediction "false", real value "true"). The CM for such a type of model would be like Fig. 3.3, page 21. In some cases, like in a hypothetical disease prediction, it is crucial to have a low false-negatives rate (and vice versa if considering other examples). The CM can help to achieve the needed model accuracy.

Learning curve

The *learning curve* (LC) shows the average accuracy (obtained with a CV method) as a function of N_{samples} (see Fig. 3.4, page 22). This is useful to understand if increasing the training dataset can improve predictions.

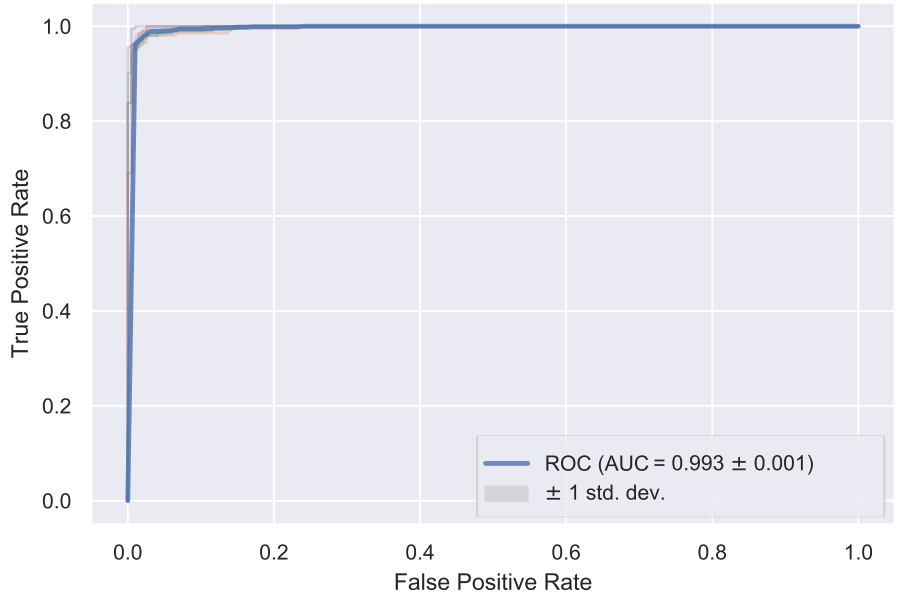


Figure 3.5: Example of a learning curve. From this plot, it's possible to assert that the model previsions have an excellent level of reliability.

ROC curve and AUC

ML predictions usually hold a degree of uncertainty, and many classifiers can output the full prediction probability. Usually, the accuracy threshold for a class prediction is 0.5. Any other threshold value can be chosen and would consequently lead to different predictions.

The *ROC curve* shows the true-positives rate vs. the false-positives rate at different thresholds from 0 to 1 (Fig. 3.5, page 23). It is an indicator of the overall classifier performance: indeed, a perfect classifier would have no false-positives so that the curve would be pushed to the top-left corner, similar to an inverted L.

An estimator linked to the ROC curve is the *AUC (Area Under the ROC Curve)*: the larger this area, the better the classification performance.

3.2.4 Model optimization

This is another critical part of the workflow. Most ML algorithms have one or more tuning parameters, called *hyperparameters* (HPs), that control the inner work of the learning algorithm. These parameters typically control the complexity of the relationships between the training dataset features and the target variable. Consequently, HPs can have a substantial impact on the

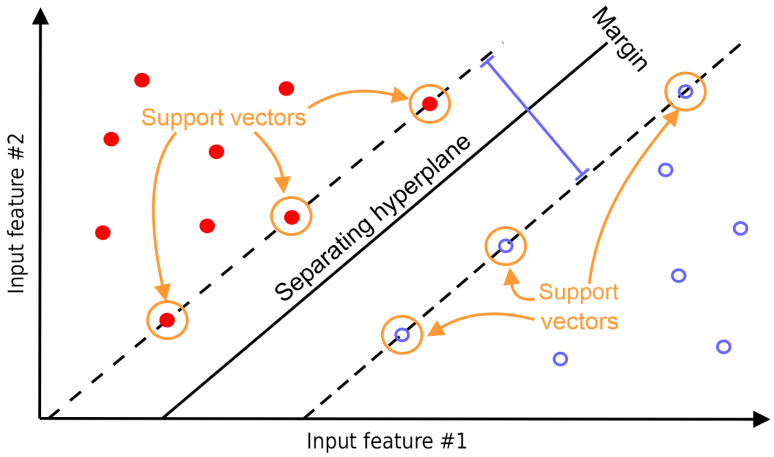


Figure 3.6: Example of Support-Vector Machine algorithm; the training dataset is divided into two categories, represented by red and blue dots.

final model and its predictive precision on new data.

The correct HPs set depends on the problem in hand and can be tuned using a brute-force method called *grid search*. First, with a random search on a vast grid in the HPs space, it is possible to find the best HPs neighborhood by creating and evaluating a model for each random HPs set extracted from the grid. Then, a full grid search in this neighborhood can reveal the best HPs set. This technique is computationally expensive but is crucial for the success of the ML model.

Note that grid search does not ensure that the chosen HPs set is the best HPs set possible.

3.2.5 Predictions on new data

Lastly, the model can be used to predict events from unseen data. Using the model to make predictions can be seen as filling out the blank target column of the new data. New input features must be in the same form as the training dataset features.

3.3 ML algorithms used in this work

The reason why I have chosen the following algorithms is that they are some of the best and most versatile ML algorithms developed until now. They are a good starting point to analyze which algorithm is the best for the problem of cosmic rays glitches.

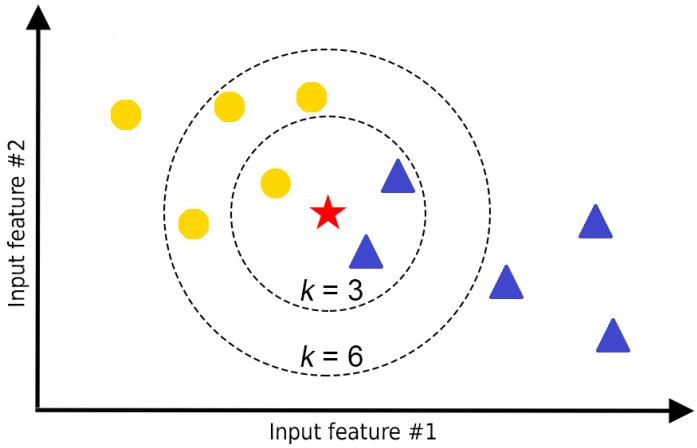


Figure 3.7: Example of the K-Nearest Neighbors algorithm. The training dataset is divided into two categories, represented by yellow dots and blue triangles; the red star is the new unseen sample.

3.3.1 Support-Vector Machine

Support-Vector Machine (SVM) is a supervised ML algorithm used for both classification and regression; it has its maximum efficiency in binary classifications.

The main idea behind SVM is to find the hyperplane that best divides the dataset into target classes in the space of the training dataset. This plane is identified by support-vectors, the points closest to the hyperplane (see Fig. 3.6, page 24). Thus, points far from the support-vectors are meaningless of this model. The possibility of using *kernel functions* make SVM a versatile model. A kernel function maps the training dataset into a different hyperspace, making predictions available also for non-linear datasets (e.g., two concentric classes).

3.3.2 K-Nearest Neighbors

K-Nearest Neighbors (KNN) is a non-parametric supervised ML algorithm primarily used for classification.

The main idea behind the KNN is that the class of a sample is the class of its nearest k samples in the training dataset space (see Fig. 3.7, page 25). It has a typical fast training because the model only has to make the training dataset map. On the contrary, the prediction time is high because the model has to compute the distance between the data point and all the points in the training set.

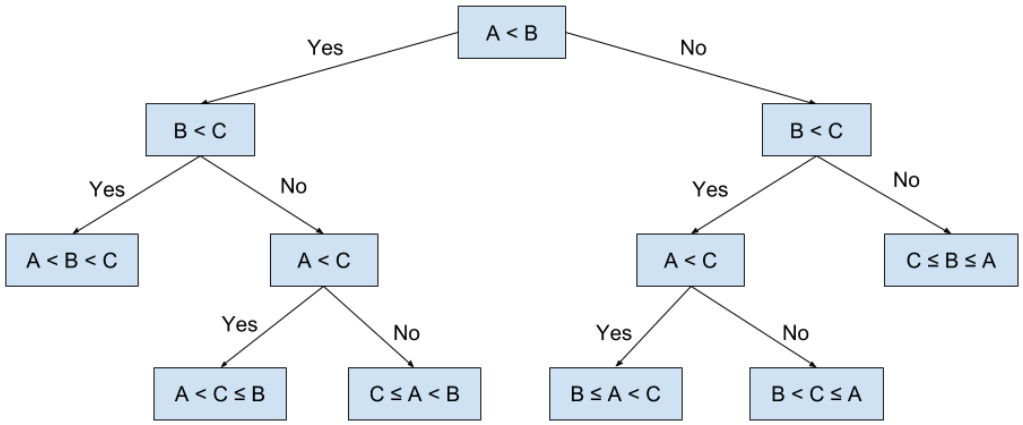


Figure 3.8: Example of a decision tree for the sorting of three numbers: by dividing data into increasingly specific categories, the algorithm succeeds in determining the relationship in which the three numbers are.

3.3.3 Random Forest

Random Forest (RF) is a non-parametric supervised ML algorithm used for both classification and regression. RF bases his functioning on decision trees and bagging ensemble (or bootstrap aggregation).

A *decision tree* is a tree-like model used to divide data into categories (see Fig. 3.8, page 26): in each division (node), data is split into subsets based on a question (e.g., "Is the number odd?", "Is the number divisible by five?"). Decision trees are sensitive to the noise of the dataset on which they are trained. Therefore, if the training dataset is modified (e.g., if a tree is trained on a subset of the training dataset), the resulting decision tree can be quite different.

Bagging (or *bootstrap aggregation*) is an ensemble method that permits combining different meta-classifier previsions into a single, more accurate prevision (see Fig. 3.9, page 27).

RF merges lots of decisional trees with the bagging ensemble method to make a single and more accurate model. This technique considerably reduces the model sensitivity to data noise.

3.3.4 Light Gradient Boosting Machine

Light Gradient Boosting Machine (LightGBM), is a non-parametric supervised ML algorithm used for both classification and regression.

LightGBM is a *gradient boosting* model. Gradient boosting is based on the idea that boosting (the technique of iteratively training an ensemble of

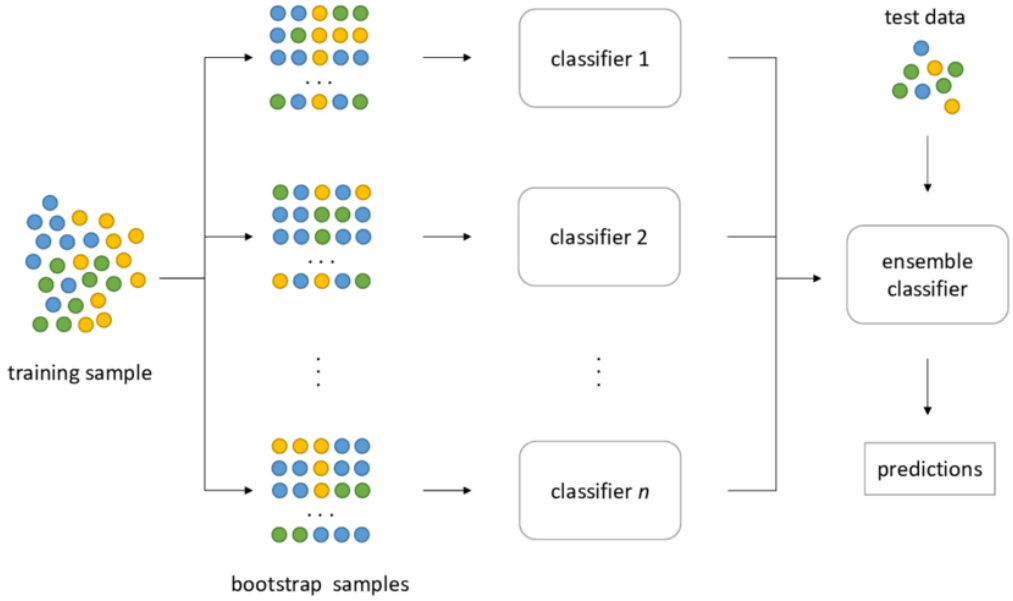


Figure 3.9: Scheme of a bagging ensemble.

weak learners and turning them into a single strong learner) can be viewed as an optimization algorithm on a suitable loss function. The algorithm optimizes a loss function over the functional space by iteratively choosing a function that points in the negative gradient direction.

3.3.5 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.

Chapter 4

ML glitches classification in Planck/HFI data

In this chapter, I explain the workflow I have followed in the preparation, creation, and optimization of my ML classification models.

4.1 Preprocessing of data

Planck/HFI data can be downloaded from the Planck Legacy Archive[1] (PLA). I choose to work on the first fifteen operating days (ODs), using 143 GHz data from bolometers 143-5, 143-6, and 143-7. I used two types of data: RAW (raw data from the instruments) and SCI (scientific data already cleaned and analyzed).

Calibration

First, raw data needs to be *calibrated*. Calibration is important in the context of this work because it allowed me to compare data from different detectors: to do this, I used the calibration constants extracted from SCI data. The complete calibration equation is:

$$S_{\text{calibrated}} = \frac{(S - C_{\text{zero-point}}) \cdot C_{\text{V-to-W}}}{C_{\text{calibration}}}$$

$$\text{with} \quad S = \frac{S_{\text{corrected}} - \text{movingAverage}(S_{\text{corrected}}, N_{\text{data-in-one-hour}})}{N_{\text{data-in-one-hour}}}$$

[1] `http://pla.esac.esa.int/pla/`

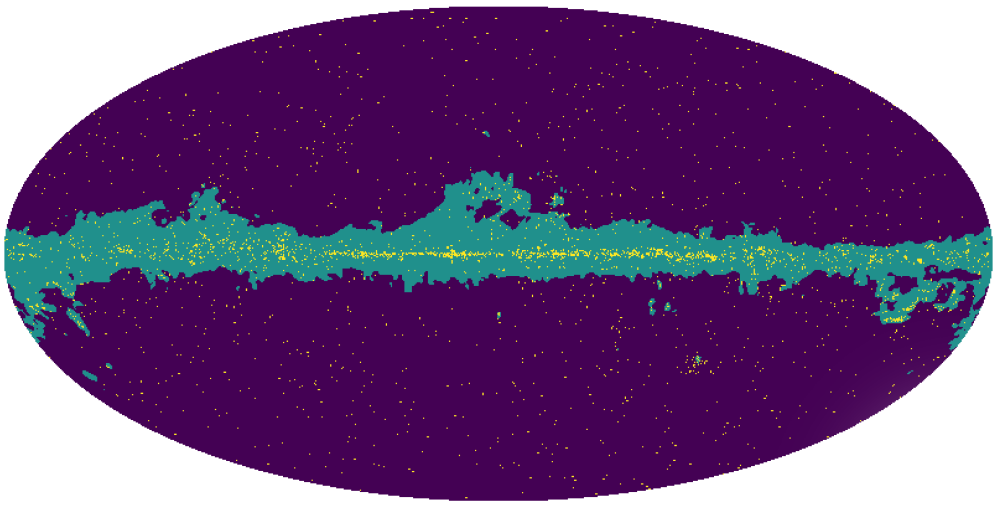


Figure 4.1: Galactic plane signal (green) and point sources (yellow) mask. Since SCI data follows the satellite data collection, the preview of the total mask cannot be performed starting from that data. However, the PLA provides the utilized masks, and thus it's possible to have a global view of the total mask.

where C_x represents the calibration constants, S_x represents the signal, and $S_{\text{corrected}} = S_{\text{raw}}$, but with the odd components of the signal array with a changed sign[2].

Cleaning

Then, I cleaned up data from the principal interferences. The two significant effects to be removed are:

- *Galactic dipole*, a result of the combined motion of the satellite with respect to the Sun and the Sun with respect to the Milky Way. The galactic dipole signal is [23]:

$$D(\boldsymbol{x}, t) = T_{\text{CMB}} \left(\frac{1}{\gamma(t)\,(1 - \boldsymbol{\beta}(t) \cdot \boldsymbol{x})} - 1 \right)$$

$$\text{with} \quad \boldsymbol{\beta} = (\boldsymbol{v}_{\text{Sun}} + \boldsymbol{v}_{\text{Planck}})/c, \quad \gamma = \left(1 - \boldsymbol{\beta}^2\right)^{-1/2}$$

  where T_{CMB} is the temperature of the CMB monopole, $\boldsymbol{v}_{\text{Sun}}$ is the velocity of the solar system with respect to the CMB rest frame, and $\boldsymbol{v}_{\text{Planck}}$ is the spacecraft velocity with respect to the solar system barycentre.

[2]This is due to a data acquisition technique used by HFI instruments.

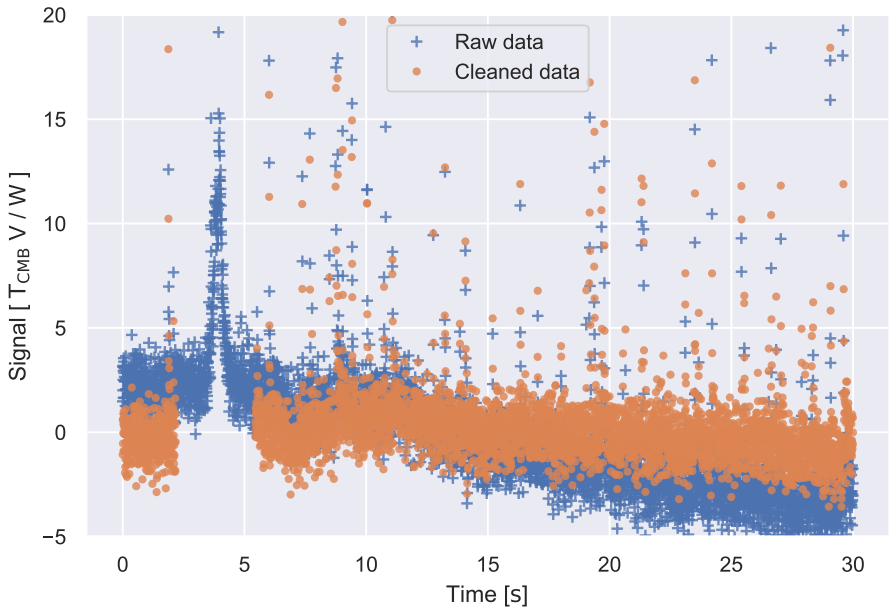


Figure 4.2: RAW data before and after the cleaning process. Both raw and cleaned signals are already calibrated.

- *Galactic plane signal* and *point sources.* These interferences can be eliminated using the flags in SCI data. SCI data are the so-called scientific data, i.e., data already processed and cleaned up from glitches. Each measurement has a flag that indicates a peculiarity in the spatial position of that sample, i.e., if the sample coincides with a point object, a planet, or the galaxy plane. For every measurement, there are 8 flags encoded in a 8 bit integer. To access flags, the number needs to be decomposed in binary format. In particular, the flags of interest are those concerning the galactic plane and the point sources:

  - `bit 4: StrongSignal; 1 = In Galactic plane`
  - `bit 5: StrongSource; 1 = On point source`

  Data with these flags must be discarded. Indeed, since the purpose of this work is detecting glitches and not cleaning up the raw signal from these effects, all points on the galactic plane or coinciding with a point source can be ignored without any consequences.

I applied the least-squares method to remove the galactic dipole; I chose this method because the calibrated RAW data and the computed galactic

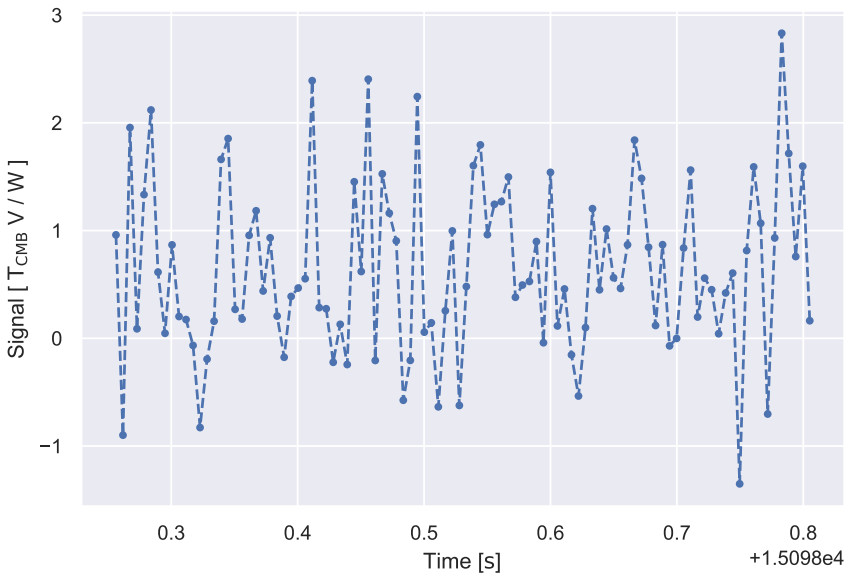


Figure 4.3: Example of a sample containing no glitch. No cosmic ray has impacted the bolometer, and data can be used for scientific purposes.

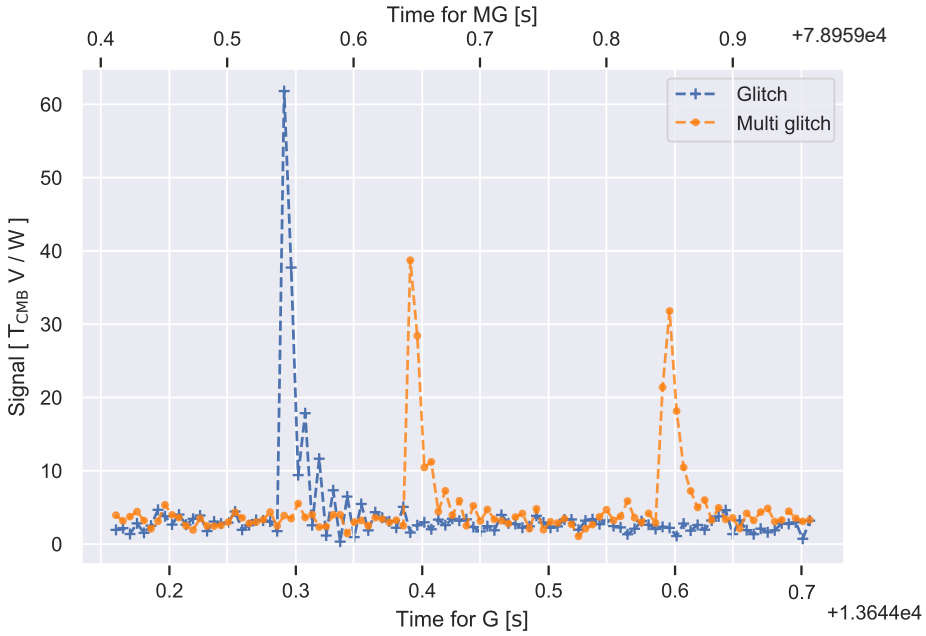


Figure 4.4: Example of two samples containing a glitch or multiple glitches. One ore more cosmic rays have impacted the bolometer, and data need to be processed to remove glitches. Note the different scales on the y-axis between this image and the one above.

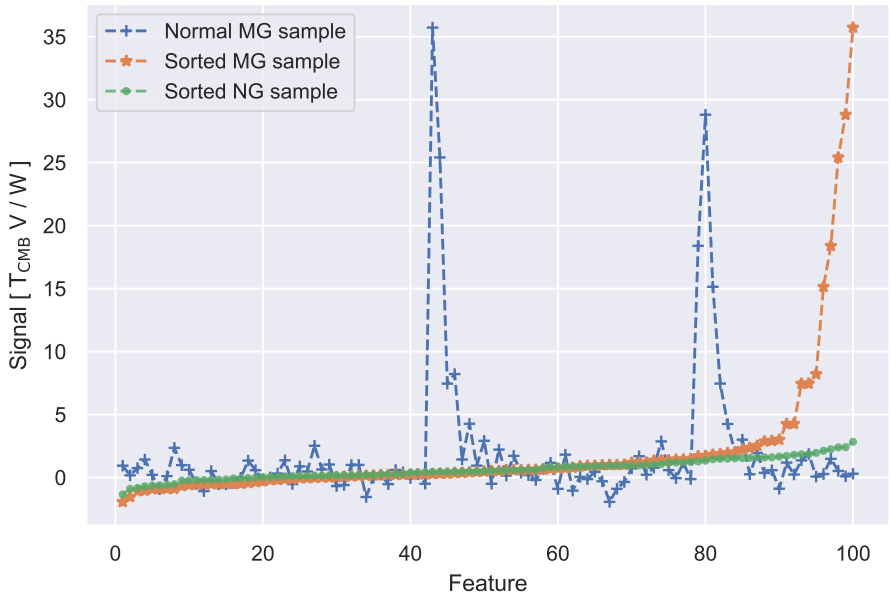


Figure 4.5: Example of a sorted sample. The blu line represents a standard time-ordered MG sample (the same in Fig. 4.4, page 32), the orange line represents the same sample after the sort, and the green line represents a sorted NG sample (the same in Fig. 4.3, page 32). The difference between the two samples is enclosed in the latest features, not at a random point of the sample as for the normal MG.

dipole do not have the same units of measurement[3]. Since the dipole modulates the RAW data, it's possible to find a correlation of the type:

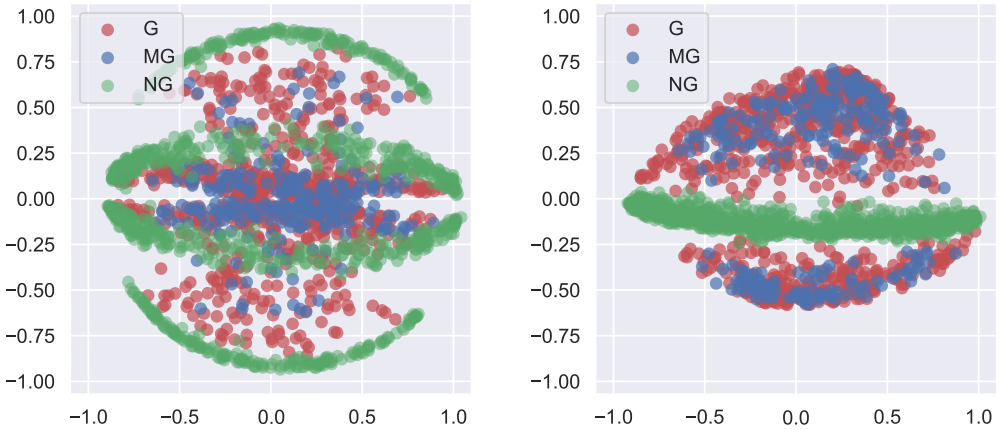
$$S_{\text{calibrated}} = m \cdot D + q$$

where $S_{\text{calibrated}}$ is the signal after the calibration process and D is the galactic dipole. Eventually, I applied the mask extracted from bits 4 and 5 in SCI data and obtained the cleaned data.

4.2 Classification

To train algorithms to recognize glitches, I created the training dataset by classifying 2000 data samples of 100 consecutive instrument measurements,

[3]This is because the constant $C_{\text{V-to-W}}$ is missing in the SCI data.



(a) Standard training dataset PCA. **(b)** Sorted training dataset PCA.

Figure 4.6: PCA results for the standard and the sorted training dataset. The two axis represent the two new dimensions extracted by the PCA algorithm.

each one taken from a random OD and a random bolometer among those considered. I made this classification by hand because, as discussed in the previous chapter, the correctness of the training dataset is of fundamental importance. I classified the following numbers of glitches:

- Samples without any glitch (*no glitch, NG*): 984
- Samples with only a glitch (*glitch, G*): 751
- Samples with multiple glitches inside (*multi glitch, MG*): 265

An example of each category can be seen in Fig. 4.3 and 4.4, page 32.

4.3 Building and evaluation of ML models

4.3.1 Preprocessing of data and model building

Feature engineering

Usually, different sample features represent different characteristics of the sample itself. In this case, the 100 features are the 100 consecutive measurements. Starting from the standard training dataset, I created two other training datasets:

- *Augmented training dataset.* As discussed above, the greater the training dataset, the higher is the accuracy that the model can have. A technique that permits incrementing the training dataset is *data augmentation*: it consists in increasing the original dataset by first applying translations, rotations, and scaling to its data, and lastly, by adding these transformed data to the original dataset. In the context of glitches detection, I applied only translations: for each sample in the original dataset, the augmented dataset contains the sample itself plus 99 translations of that sample. So, the augmented dataset has 200 000 samples. This technique also permits to achieve translational equivalence - i.e., the property that a glitch can be recognized regardless of its position within the 100 features in a sample).

- *Sorted training dataset*, in which I applied a *sorting operation* to the 100 features of each sample. Since glitches can be both positives (upwards peaks) and negatives (downwards peaks) and their absolute value is higher than regular data, a sort shifts the peaks in the firsts of latest features: these features become identifiers of the presence of glitches (see Fig. 4.5, page 33).

I trained every ML algorithm with each dataset.

Binary and multiclass classification analysis

I tried to figure out if a multiclass classification between NG, G, and MG was possible[4]. To understand this, I used *Principal Components Analysis* (PCA), a dimensionality reduction algorithm. In my training dataset, I have 100 dimensions (the 100 features). I used PCA to reduce my dataset into two dimensions so that I could plot them. The results of the PCA are reported in Fig. 4.6, page 34. There are two main outcomes:

- *No multiclass classification can be done* because of the superimposition of red points and blue points: this means that there is no characteristic feature to distinguish between G and MG. For SVM, I tried both binary (NG and G+MG) and multiclass (NG, G, and MG) classifications: the outcomes confirmed this prevision. So, I used only *binary classifiers* (NG and G+MG): so, from here, I refer to G+MG simply as G.

- The sorted training dataset groups and distinguishes better NG from G. So, models trained with this dataset could potentially have better accuracy.

[4]Obviously, the sorted training dataset loses the distinction between G and MG.

4.3.2 Validation methods

Cross-validation

To evaluate the accuracy of the models, I used a variant of the k-fold CV, called *stratified k-fold CV*. Stratified means that the ratio between the categories in the training dataset is maintained in the generated k folds, ensuring consistent partial results. I used 5 folds and repeated the stratified k-fold CV 50 times with a random shuffle seed. All results are aggregated to asses the final accuracy.

Confusion matrix

I made the CM by randomly splitting with the stratified method the original training dataset into two subsets: the training subset (70% of the training dataset) and the testing subset (30% of the training dataset). The algorithm is trained on the training subset and tested on the testing subset. Then, by comparing the predictions with the real labels, I created the partial CM. The final CM is obtained by repeating this process 100 times, merging all the partials CM and normalizing the values on the total number of predictions.

Below, I report a pseudocode example:

```
clf = classifier(hyper_parameters)
cm = np.zeros(shape=(2,2), dtype=int)

data, target = training_dataset['features'], training_dataset['labels']

# Partial confusion matrices
for i in range(100):
    # Split the data into a training set and a test set
    X_train, X_test, y_train, y_test = train_test_split(data, target,
    test_size=0.3, random_state=None, stratify=target)
    # Train and predict
    y_pred = clf.fit(X_train, y_train).predict(X_test)
    # Compute confusion matrix
    labels = [0., 1.]
    labels_text = ['no glitch', 'glitch']
    cm += confusion_matrix(y_test, y_pred, labels=labels)
# Final confusion matrix
plot_confusion_matrix(cm, labels_text, normalize=True)
```

For the problem in hand, it's essential to reduce the number of false-negative. A false-positive introduces a statistical error: indeed, removing a non-glitch reduces the quantity of data used in the analysis and impacts the sensitivity of the measurement. However, a false-negative introduces a *systematic error*. The CM helps to understand how different models misclassify data, and so which models are the most suitable for the problem of cosmic rays glitches.

Learning curve

To plot the learning curve, I started from 5% of the total training dataset, and I incremented it step by step by 5%. The subset is randomly split with the stratified method into two subsets: the training subset (70% of the subset) and the testing subset (30% of the subset). The algorithm is trained on the training subset and tested on the testing subset. Then, by computing the accuracy for each incrementing subset, I built the learning curve.

Below, I report a pseudocode example:

```
clf = classifier(hyper_parameters)
cv = ShuffleSplit(test_size=0.3, random_state=None)

# Plot learning curve
plot_learning_curve(clf, data, target, n_steps=20, cv=cv)
```

ROC curve and AUC

I made the ROC curve using the stratified k-fold CV (with $k = 5$) and computing the probabilities for the predictions. Then, by changing the threshold, I built the ROC curve. The AUC is obtained with its definition - i.e., by computing the area under the ROC curve.

4.3.3 Optimization method

I used the optimization technique reported in Sec. 3.2.4.

4.3.4 Timing methods

A ML model is not useful if it analyzes an entire OD in too much time. Indeed, the models I built only analyze data to search for the presence of a glitch: in a real application, glitch detection is only a little step of the analysis pipeline.

Training and testing time

The testing time is much more important than the training time: indeed, in a functioning model, the training is done only once at the beginning.

I measured the training time by merely timing the training instruction and the testing time by timing, for 2000 times, the prediction instruction on the entire training dataset (therefore, the total testing dataset has 4 million elements). I did not use any CV method because here I am interested only in the time needed to make predictions, not on the accuracy of these predictions.

Below, I report a pseudocode example:

```
clf = classifier(hyper_parameters)

# Measure training time
t_b = time.time()
clf.fit(training_dataset['features'], training_dataset['label'])
t_e = time.time()
# Total training time
training_time = t_e - t_b

# Measure testing time
t = 0.
for i in range(2000):
    t_b = time.time()
    clf.predict_proba(training_dataset['features'])
    t_e = time.time()
    t += t_e - t_b
# Total testing time
testing_time = t
```

Real-life test

The final test is a simulation of a real-life application of the models I have created: it's a timing of the classification of all the data in the considered ODs and bolometers. This simulation is a litmus test of the real performances of my models. Indeed, if an accurate model is also quick to make predictions, it can be taken into consideration for future researches.

Below, I report a pseudocode example:

```
clf = classifier(hyper_parameters)
clf.fit(training_dataset['features'], training_dataset['label'])

T1 = time.time()

# List for intermediate times
time_list = []

# Test all the ODs and detector
for OD in OPERATING_DAY_LIST:
    for bolometer in BOLOMETER_LIST:

        # Load data of the selected OD/bolometer
        current_dataframe = OD + '/' + bolometer
        with pandas.HDFStore(filename, mode='r') as in_data:
            data_dataframe = in_data[current_dataframe]
        # Number of sequences in the selected OD/bolometer
        n_sequences = data_dataframe.index[::100].shape[0]

        # Start test
        t = 0
        for i in range(0, n_sequences-1):
            # Reshape dataframe in sklearn input format
            d = data_dataframe.iloc[i*100 : i*100 + 100].to_numpy().
    transpose()
```

```
25              # Predict
26              t_b = time.time()
27              clf.predict_proba(d)
28              t_e = time.time()
29              t += (t_e - t_b)
30
31          # Append partial time into the time list
32          t_list.append(t)
33
34  T2 = time.time()
35
36  # Total time
37  # Only total prediction time
38  model_time = np.sum(t_list)
39  # Total time
40  total_time = T2 - T1
```

I computed two times (see lines 36-40 in the pseudocode above):

- *Model time* (in the pseudocode, `model_time`, line 38). It estimates only the prediction time without considering the time necessary to do the other operations, like load and format the input data.
- *Total time* (in the pseudocode, `total_time`, line 40). It estimates the total time needed to predict glitches, considering every operation.

Chapter 5

Results

5.1 General outcomes and best model

Final results are reported in pages from 42 to 44. I do all tests on a 4-core desktop CPU.

Globally, the best algorithm turned out to be SVC (see Sec. 3.3.1) 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 in Sec. 4.3.1, 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 5.3 and Fig. 5.1, page 42).
- 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 5.5, page 43).
- Excluding LGB[1] (see Sec. 3.3.4), SVC is the fastest algorithm in terms of prediction time, closely followed by RFC (see Sec. 3.3.3); as expected, KNC (see Sec. 3.3.2) is the slowest. See Table 5.5, page 43.
- 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 5.7, page 44.

[1] RFC and LGB prediction times need to be investigated because of some discrepancies in the obtained times: this will be done in Sec. 5.2.3 and 5.2.4.

Evaluation tables

Table 5.1: 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 5.2: Models trained with the augmented training dataset.

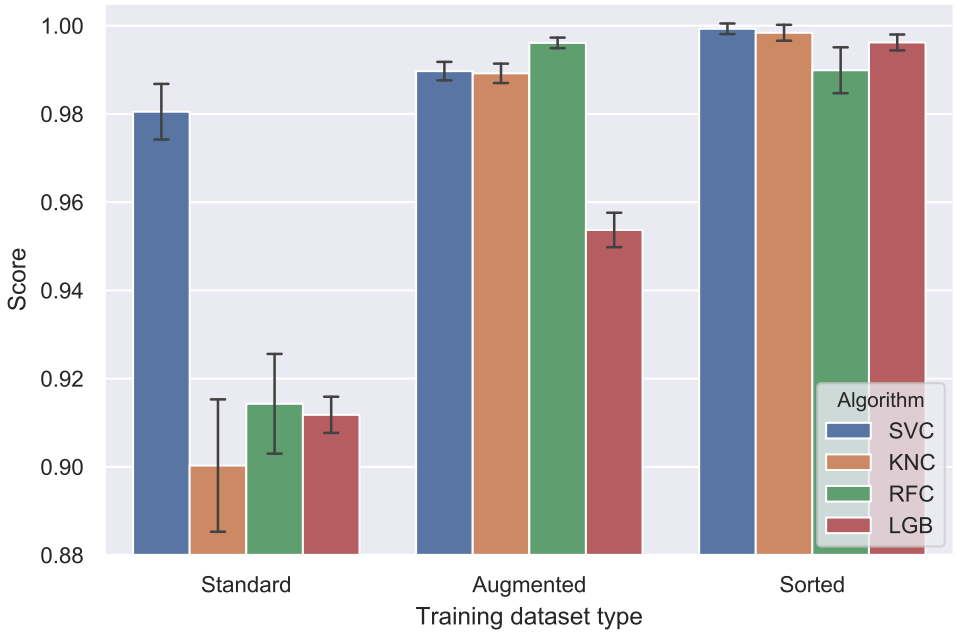
	SVC	KNC	RFC	LGB
Score	0.9897 ± 0.0021	0.9892 ± 0.0022	0.9961 ± 0.0012	0.9537 ± 0.0039

Table 5.3: 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 5.1: Score comparison between the four ML models.



Timing tables

Table 5.4: Models trained with standard and augmented training dataset.

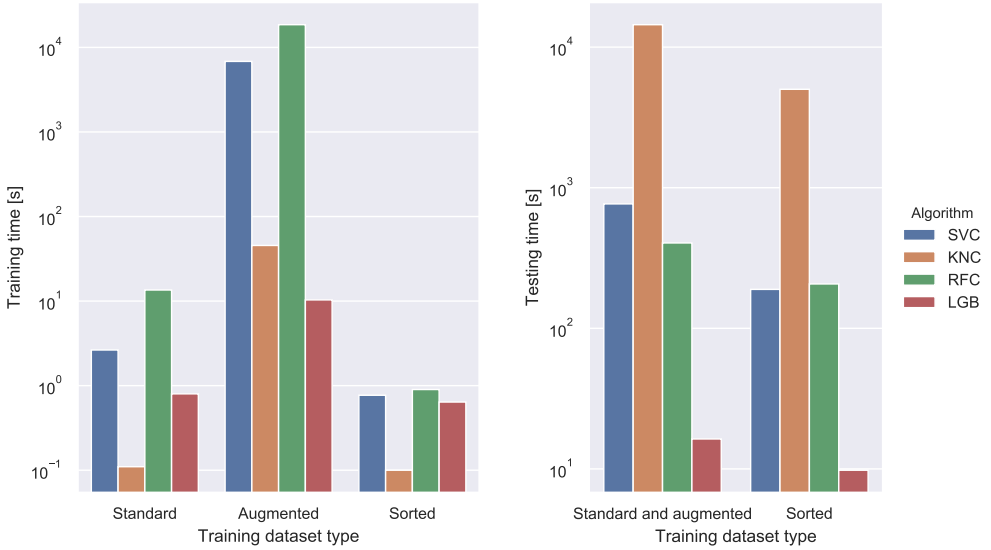
	SVC	KNC	RFC	LGB
Training time (standard dataset)	2.64 s (1 thread)	0.11 s (4 threads)	13.52 s (4 threads)	0.80 s (4 threads)
Training time (augmented dataset)	1 h 53 m 50.24 s (1 thread)	45.48 s (4 threads)	5 h 9 m 18.71 s (4 threads)	10.33 s (4 threads)
Testing time	12 m 47.23 s (1 thread)	4 h 0 m 39.05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

Table 5.5: 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 5.2: Training and testing time comparison between the four ML models. Note the logarithmic scales on the y-axis.



Real-life test tables

Table 5.6: Models trained with the standard training dataset.

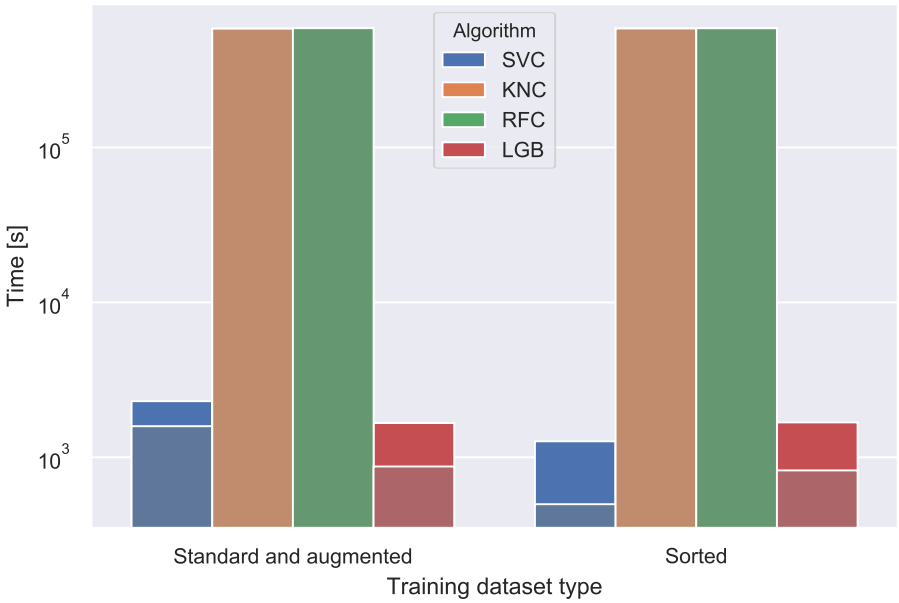
	SVC	KNC	RFC	LGB
Model time	26 m 27.21 s (1 thread)	6 d 18 h 20 m 54.57 s (4 threads)	6 d 19 h 29 m 5.18 s (4 threads)	14 m 32.23 s (4 threads)
Total time	38 m 22.22 s (1 thread)	6 d 18 h 46 m 23.87 s (4 threads)	6 d 20 h 9 m 27.41 s (4 threads)	27 m 43.87 s (4 threads)

Table 5.7: 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

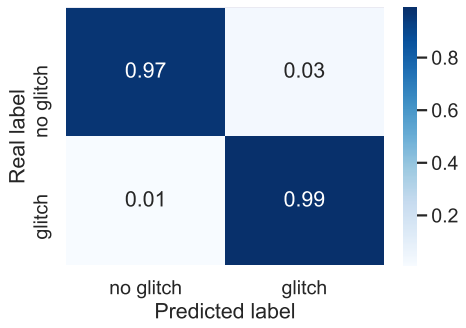
Figure 5.3: Real-life test comparison between the four ML models. Light colors refer to the model time; saturated colors refer to the total time. Note the logarithmic scales on the y-axis.



SVC evaluation plots

Figure 5.4: Confusion matrices.

(a) Standard model.



(b) Sorted model.

Figure 5.5: Learning curves.

(a) Standard model.

(b) Sorted model.

Figure 5.6: ROC curves.

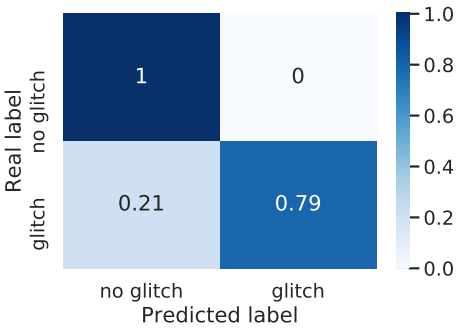
(a) Standard model.

(b) Sorted model.

KNC evaluation plots

Figure 5.7: Confusion matrices.

(a) Standard model.



(b) Sorted model.

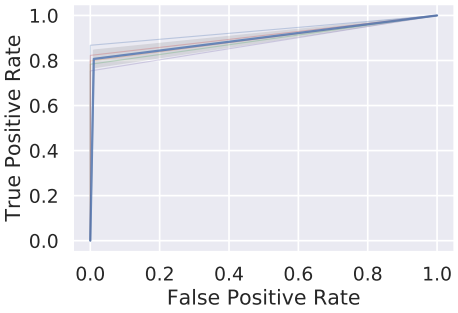
Figure 5.8: Learning curves.

(a) Standard model.

(b) Sorted model.

Figure 5.9: ROC curves.

(a) Standard model.

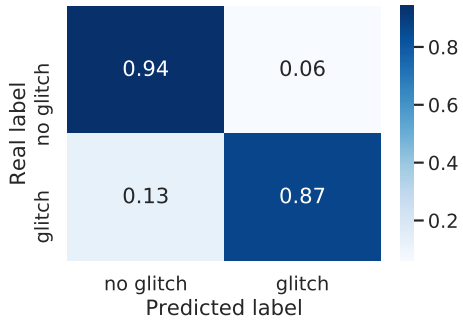


(b) Sorted model.

RFC evaluation plots

Figure 5.10: Confusion matrices.

(a) Standard model.



(b) Sorted model.

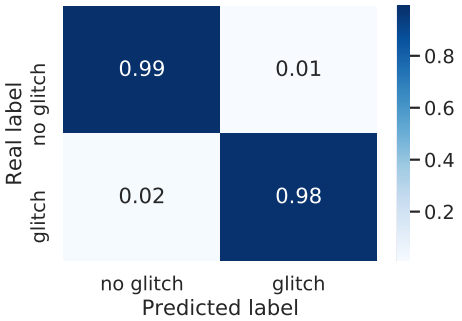


Figure 5.11: Learning curves.

(a) Standard model.

(b) Sorted model.

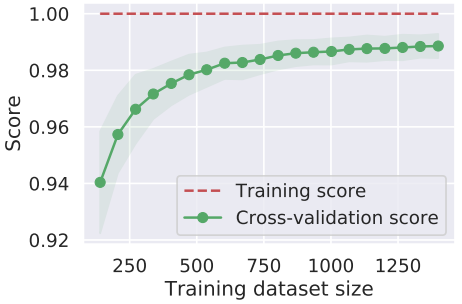
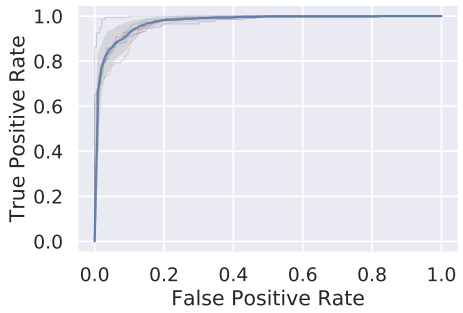
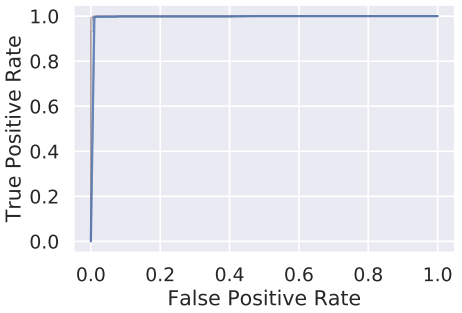


Figure 5.12: ROC curves.

(a) Standard model.



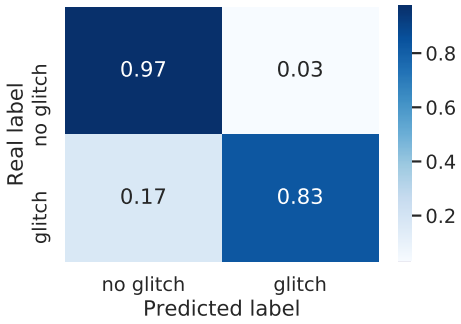
(b) Sorted model.



LGB evaluation plots

Figure 5.13: Confusion matrices.

(a) Standard model.



(b) Sorted model.

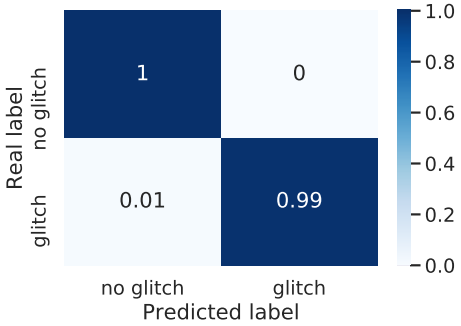


Figure 5.14: Learning curves.

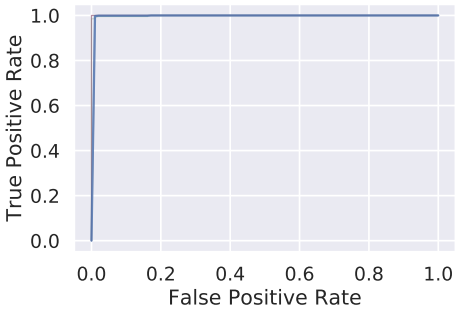
(a) Standard model.

(b) Sorted model.

Figure 5.15: ROC curves.

(a) Standard model.

(b) Sorted model.



5.2 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].

5.2.1 Support-Vector Machine

SVC evaluation plots can be seen on page 45. 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 5.5a.

From the confusion matrix 5.4a, 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. 3.3.5.

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.

5.2.2 K-Nearest Neighbors

KNC evaluation plots can be seen on page 46. 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 consistent with predictions in Sec. 3.3.2: a fast train and slow predictions. Furthermore, the confusion matrix 5.7a 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.

5.2.3 Random Forest

RFC evaluation plots can be seen on page 47. 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. 5.11. The LC 5.11a 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 5.11b 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. 5.10, page 47).

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 5.4 and 5.5 on page 43, 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. 4.3.4): 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.

5.2.4 Light Gradient Boosting Machine

LGB evaluation plots can be seen on page 48. 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 5.4 and 5.5, page 43). 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. 5.13, page 48).

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

Chapter 6

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, explained in Sec. 3.3: 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 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 42 to 44). 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.

Bibliography

[1] Kevork N. Abazajian et al. "CMB-S4 Science Book, First Edition". In: *arXiv e-prints*, arXiv:1610.02743 (Oct. 2016), arXiv:1610.02743. arXiv: 1610.02743 [astro-ph.CO].

[2] Maximilian H. Abitbol et al. "CMB-S4 Technology Book, First Edition". In: *arXiv e-prints*, arXiv:1706.02464 (June 2017), arXiv:1706.02464. arXiv: 1706.02464 [astro-ph.IM].

[3] Peter Ade et al. "The Simons Observatory: science goals and forecasts". In: J. Cosmology Astropart. Phys. 2019.2, 056 (Feb. 2019), p. 056. DOI: 10.1088/1475-7516/2019/02/056. arXiv: 1808.07445 [astro-ph.CO].

[4] J. Aumont et al. "QUBIC Technical Design Report". In: arXiv:1609.04372 (Sept. 2016), arXiv:1609.04372. arXiv: 1609.04372.

[5] Amedeo Balbi, Paolo Natoli, and Nicola Vittorio. "The CMB polarization: status and prospects". In: *arXiv e-prints*, astro-ph/0606511 (June 2006), astro–ph/0606511. arXiv: astro-ph/0606511 [astro-ph].

[6] C. L. Bennett et al. "The Microwave Anisotropy Probe Mission". In: ApJ 583.1 (Jan. 2003), pp. 1–23. DOI: 10.1086/345346. arXiv: astro-ph/0301158 [astro-ph].

[7] N. W. Boggess et al. "The COBE Mission: Its Design and Performance Two Years after Launch". In: ApJ 397 (Oct. 1992), p. 420. DOI: 10.1086/171797.

[8] Henrik Brink, Joseph Richards, and Mark Fetherolf. *Real-World Machine Learning*. 1st. USA: Manning Publications Co., 2016. ISBN: 1617291927.

[9] The Planck Collaboration. *The Scientific Programme of Planck*. 2006. arXiv: astro-ph/0604069 [astro-ph].

[10] R. H. Dicke et al. "Cosmic Black-Body Radiation." In: ApJ 142 (July 1965), pp. 414–419. DOI: 10.1086/148306.

[11] Robert H. Dicke et al. "Atmospheric Absorption Measurements with a Microwave Radiometer". In: *Physical Review* 70.5-6 (Sept. 1946), pp. 340–348. DOI: 10.1103/PhysRev.70.340.

[12] Thomas Essinger-Hileman et al. "CLASS: the cosmology large angular scale surveyor". In: vol. 9153. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. July 2014, p. 91531I. DOI: 10.1117/12.2056701. arXiv: 1408.4788.

[13] J. W. Fowler et al. "Optical design of the Atacama Cosmology Telescope and the Millimeter Bolometric Array Camera". In: Appl. Opt. 46.17 (June 2007), pp. 3444–3454. DOI: 10.1364/AO.46.003444. arXiv: astro-ph/0701020 [astro-ph].

[14] G. Gamow. "Expanding Universe and the Origin of Elements". In: *Physical Review* 70.7-8 (Oct. 1946), pp. 572–573. DOI: 10.1103/PhysRev.70.572.2.

[15] D. Hanson et al. "Detection of B-Mode Polarization in the Cosmic Microwave Background with Data from the South Pole Telescope". In: 111.14, 141301 (Oct. 2013), p. 141301. arXiv: 1307.5830.

[16] M. Hazumi et al. "LiteBIRD: A Satellite for the Studies of B-Mode Polarization and Inflation from Cosmic Background Radiation Detection". In: *Journal of Low Temperature Physics* 194.5-6 (Mar. 2019), pp. 443–452. DOI: 10.1007/s10909-019-02150-5.

[17] Wayne Hu and Martin White. "A CMB polarization primer". In: New A 2.4 (Oct. 1997), pp. 323–344. DOI: 10.1016/S1384-1076(97)00022-5. arXiv: astro-ph/9706147 [astro-ph].

[18] Howard Hui et al. "BICEP Array: a multi-frequency degree-scale CMB polarimeter". In: vol. 10708. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. July 2018, p. 1070807. DOI: 10.1117/12.2311725. arXiv: 1808.00568.

[19] Marc Kamionkowski and Ely D. Kovetz. "The Quest for B Modes from Inflationary Gravitational Waves". In: *Annual Review of Astronomy and Astrophysics* 54.1 (2016), pp. 227–269. DOI: 10.1146/annurev-astro-081915-023433.

[20] Yong-Ping Li et al. "Tibet's Ali: A New Window to Detect the CMB Polarization". In: arXiv:1709.09053 (Sept. 2017), arXiv:1709.09053. arXiv: 1709.09053.

[21] Andrew McKellar. "Molecular Lines from the Lowest States of Diatomic Molecules Composed of Atoms Probably Present in Interstellar Space". In: *Publications of the Dominion Astrophysical Observatory Victoria* 7 (Jan. 1941), p. 251.

[22] A. A. Penzias and R. W. Wilson. "A Measurement of Excess Antenna Temperature at 4080 Mc/s." In: ApJ 142 (July 1965), pp. 419–421. DOI: 10.1086/148307.

[23] Planck Collaboration et al. "Planck 2013 results. V. LFI calibration". In: A&A 571, A5 (Nov. 2014), A5. DOI: 10.1051/0004-6361/201321527. arXiv: 1303.5066 [astro-ph.IM].

[24] Planck Collaboration et al. "Planck 2013 results. X. HFI energetic particle effects: characterization, removal, and simulation". In: A&A 571, A10 (Nov. 2014), A10. DOI: 10.1051/0004-6361/201321577. arXiv: 1303.5071 [astro-ph.CO].

[25] Planck HFI Core Team et al. "Planck early results. VI. The High Frequency Instrument data processing". In: A&A 536, A6 (Dec. 2011), A6. DOI: 10.1051/0004-6361/201116462. arXiv: 1101.2048 [astro-ph.CO].

[26] A. Suzuki et al. "The LiteBIRD Satellite Mission: Sub-Kelvin Instrument". In: 193.5-6 (Dec. 2018), pp. 1048–1056. DOI: 10.1007/s10909-018-1947-7. arXiv: 1801.06987.

[27] A. Suzuki et al. "The Polarbear-2 and the Simons Array Experiments". In: 184.3-4 (Aug. 2016), pp. 805–810. DOI: 10.1007/s10909-015-1425-4. arXiv: 1512.07299.

[28] The LSPE collaboration et al. "The Large-Scale Polarization Explorer (LSPE)". In: arXiv:1208.0281 (Aug. 2012), arXiv:1208.0281. arXiv: 1208.0281.

Acknowledgments

I would like to thank my supervisor, Prof. Maurizio Tomasi, for his help in the realization of this thesis work. His guidance helped me to better develop this work in all its phases, from the initial preparation to the final writing.

I thank all my friends and classmates for their support in these years of study: Leonardo, Eleonora Fe., Paola, Marcello, Andrea, Irene, Eleonora Fo., Denise, Samuele C., Filippo, Samuele A., Gabriele, Daniele, Giovanni, Liliana, Sofia, Rossella, and many others. List them all would be impossible.

Special thanks go to my friend Marilisa, for always encouraging and helping me during these years and in the most demanding moments.

Finally, big thanks go to my parents Elena and Giorgio and my brother Marco for their love, support, and patience in these years of work.

Paolo Galli
February 2020