**ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA**
**CAMPUS DI CESENA**

Scuola di Ingegneria e Architettura
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

# Interpretable Prediction of Galactic Cosmic-Ray Short-Term Variations with Artificial Neural Networks

Tesi di laurea in
Sistemi Autonomi

*Relatore*
**Chiar.mo Prof. Andrea Omicini**

*Candidato*
**Federico Sabbatini**

*Correlatore*
**Prof.ssa Catia Grimani**
**Dott. Giovanni Ciatto**

Sessione di Laurea Unica
Anno Accademico 2019-2020

# Abstract

Monitoring the galactic cosmic-ray flux variations is a crucial issue for all those space missions for which cosmic rays constitute a limitation to the performance of the on-board instruments. If it is not possible to study the galactic cosmic-ray short-term fluctuations on board, it is necessary to benefit of models that are able to predict these flux modulations.

Artificial neural networks are nowadays the most used tools to solve a wide range of different problems in various disciplines, including medicine, technology, business and many others. All artificial neural networks are black boxes, i.e. their internal logic is hidden to the user. Knowledge extraction algorithms are applied to the neural networks in order to obtain explainable models when this lack of explanation constitutes a problem.

This thesis work describes the implementation and optimisation of an explainable model for predicting the galactic cosmic-ray short-term flux variations observed on board the European Space Agency mission LISA Pathfinder. The model is based on an artificial neural network that benefits as input data of solar wind speed and interplanetary magnetic field intensity measurements gathered by the National Aeronautics and Space Administration space missions Wind and ACE orbiting nearby LISA Pathfinder. The knowledge extraction is performed by applying to the underlying neural network both the ITER algorithm and a linear regressor. ITER was selected after a deep investigation of the available literature. The model presented here provides explainable predictions with errors smaller than the LISA Pathfinder statistical uncertainty.

**Keywords:** Artificial neural networks, knowledge extraction, explainable artificial intelligence, interpretable prediction, LISA Pathfinder, cosmic rays

*To myself.*
*Again.*

# Acknowledgements

Throughout the writing of this work I have received a great deal of support and assistance.

I would first like to thank my supervisor, Professor Andrea Omicini, and assistant supervisors, Professor Catia Grimani and Dr. Giovanni Ciatto, whose help and expertise were fundamental and invaluable for the realisation of my thesis. Working with you has increased both my personal and scientific formation and your precious feedback pushed me to give my best, bringing my work to a higher level. Thank you!

I would like to thank Mother for her enthusiasm during the preparation of my thesis.

I would like to thank Paolen for having read my work. Next time we'll speak about $(a + b)^4$!

I would like to thank Pedro for having (not so) patiently tolerated to be overlooked during these months in which I worked to this thesis.

I would like to thank my Big Hamster for his ideas and the time he spent with me. *Sparpack!*

I am greatly indebted with the LISA Pathfinder Collaboration for providing the cosmic-ray data gathered during the mission elapsed time. The data can be also publicly downloaded from the website `https://www.cosmos.esa.int/web/lisa-pathfinder-archive/home`.

The sunspot number and data about the solar modulation parameter have been taken from the websites `http://solarscience.msfc.nasa.gov/SunspotCycle.shtml` and `http://cosmicrays.oulu.fi/phi/Phi_mon.txt`, respectively. Data gathered by Wind and ACE experiments are available on the NASA-CDAWeb Internet website `ftp://cdaweb.gsfc.nasa.gov/pub/data/omni/omni_cdaweb/hourly/2016`. Neutron monitor data have been taken from `http://cosray.phys.uoa.gr`, while those about the heliospheric current sheet crossing are available here: `https://omniweb.sci.gsfc.nasa.gov./html/polarity/polarity_tab.html`.

# Contents

**Abstract** **iii**

**List of Figures** **xi**

**List of Tables** **xv**

**List of Algorithms** **xvii**

**1 Introduction** **1**

**2 Machine Learning and Artificial Neural Networks** **3**
2.1 General aspects of machine learning models . . . . . 3
2.1.1 Typical problems and inputs . . . . . 3
2.1.2 Splitting of the data set . . . . . 5
2.1.3 Under-fitting and over-fitting . . . . . 6
2.1.4 Time series processing . . . . . 7
2.2 Artificial neural networks . . . . . 9
2.2.1 Artificial and biological neurons . . . . . 10
2.2.2 Activation functions . . . . . 11
2.2.3 Multi-layer perceptrons . . . . . 15
2.2.4 Hidden layers and hidden units . . . . . 16
2.2.5 Training . . . . . 17
2.2.6 Epochs and iterations . . . . . 17
2.2.7 Early stopping . . . . . 18
2.2.8 Loss function . . . . . 18
2.2.9 Learning rate . . . . . 18
2.3 Pruning in neural network models . . . . . 19

**3 Knowledge Extraction** **21**
3.1 Interpretable models . . . . . 21
3.1.1 Black box problems . . . . . 21
3.1.2 Explanator types . . . . . 22

3.2 Extracting knowledge from black boxes . . . 23
3.2.1 Explanator classification. . . . 24
3.2.2 Translucency of the model . . . 24
3.2.3 Quality of the extracted rules . . . 25
3.3 Extracting knowledge from neural networks . . . 25
3.3.1 Knowledge extraction from classifiers . . . 25
3.3.2 Knowledge extraction from regressors . . . 29
3.3.3 Comparison between the described algorithms . . . 32

**4 The LISA Pathfinder Mission 35**
4.1 The LISA Pathfinder mission . . . 35
4.1.1 Characteristics and orbit . . . 36
4.1.2 The particle detector . . . 36
4.2 Galactic cosmic-rays . . . 39
4.2.1 Long-term variations of galactic cosmic-rays . . . 39
4.2.2 Short-term variations of galactic cosmic-rays . . . 44
4.2.3 Forbush decreases . . . 47
4.3 Cosmic-ray flux short-term variations observed with LISA Pathfinder 48

**5 Predicting LISA Pathfinder Data 53**
5.1 Model design . . . 53
5.2 Implementation . . . 55
5.2.1 Data set creation . . . 55
5.2.2 Neural network implementation and optimisation . . . 56
5.2.3 Knowledge extraction from the neural network . . . 72
5.3 Programming language . . . 84

**6 Results 87**
6.1 Comparison between the implemented models . . . 87
6.2 Prediction of LISA Pathfinder galactic-cosmic ray flux variation missing data . . . 98

**7 Conclusions 113**

**Bibliography 117**

# List of Figures


2.1 Data set splitting into training, validation and test sets. . . . . 6
2.2 Under-fitting and over-fitting. . . . . 7
2.3 Over-fitting example. . . . . 8
2.4 Biological neuron. . . . . 11
2.5 Artificial neuron. . . . . 12
2.6 Examples of activation functions. . . . . 13
2.7 Multi-layer perceptron. . . . . 16

4.1 Orbits of the LISA Pathfinder, ACE and Wind satellites. . . . . 37
4.2 Schematic representation of the instrumentation on board the LISA Pathfinder spacecraft. . . . . 38
4.3 Schematic representation of the particle detectors on board LISA Pathfinder. . . . . 40
4.4 Solar modulation parameter and LISA Pathfinder particle detector galactic cosmic-ray single count rate during the LISA Pathfinder mission. . . . . 42
4.5 Galactic cosmic-ray proton and helium energy spectra measurements. 45
4.6 Moscow neutron monitor counting rate in December 2006. . . . . 48
4.7 Comparison of LISA Pathfinder particle detector counting rate percent variations with ACE measurements of the solar wind speed and interplanetary magnetic field during the Bartels rotation 2496. . . . 50
4.8 Comparison of LISA Pathfinder particle detector counting rate percent variations with contemporary, analogous measurements of neutron monitors during the Bartels rotation 2496 . . . . . 51

5.1 Schematic representation of this thesis workflow. . . . . 54
5.2 Galactic cosmic-ray flux percent variations, solar wind speed and interplanetary magnetic field intensity measured during the Bartels rotation 2505. . . . . 57

5.3 Mean squared error calculated on neural networks with one only hidden layer with a different number of neurons and for various learning rates. . . . . 61
5.4 Refined region of minimum MSE following from Figure 5.3. . . . . 61
5.5 The same of Figure 5.3 for neural networks with two hidden layers. 64
5.6 Refined region of minimum MSE following from Figure 5.5. . . . . 64
5.7 Mean squared error calculated on neural networks with different time windows for the input variables. . . . . 66
5.8 Same as Figure 5.2 with the optimal time windows estimated for the input variables. . . . . 68
5.9 Same as Figure 5.8 with the optimal samplings estimated for the input variables. . . . . 70
5.10 Same as Figure 5.9 with average values. . . . . 71
5.11 Hyper-cube expansion during the iterations of the ITER algorithm. 76
5.12 Proposed variation of the ITER algorithm. . . . . 77
5.13 Mean absolute error measured on the predictions of a regression tree trained with the output of the presented neural network. . . . . 79
5.14 Proposed algorithm for the expansion in the feature space starting from a single data set example. . . . . 82

6.1 Galactic cosmic-ray flux percent variations, solar wind speed and interplanetary magnetic field intensity measured during the Bartels rotation 2491. . . . . 89
6.2 Same as Fig. 6.1 for the Bartels rotation 2492. . . . . 90
6.3 Same as Fig. 6.1 for the Bartels rotation 2493. . . . . 91
6.4 Same as Fig. 6.1 for the Bartels rotation 2494. . . . . 92
6.5 Same as Fig. 6.1 for the Bartels rotation 2495. . . . . 93
6.6 Same as Fig. 6.1 for the Bartels rotation 2496. . . . . 94
6.7 Same as Fig. 6.1 for the Bartels rotation 2497. . . . . 95
6.8 Same as Fig. 6.1 for the Bartels rotation 2498. . . . . 96
6.9 Same as Fig. 6.1 for the Bartels rotation 2499. . . . . 97
6.10 Same as Fig. 6.1 for the Bartels rotation 2500. . . . . 98
6.11 Same as Fig. 6.1 for the Bartels rotation 2501. . . . . 99
6.12 Same as Fig. 6.1 for the Bartels rotation 2502. . . . . 100
6.13 Same as Fig. 6.1 for the Bartels rotation 2503. . . . . 101
6.14 Same as Fig. 6.1 for the Bartels rotation 2504. . . . . 102
6.15 Same as Fig. 6.1 for the Bartels rotation 2505. . . . . 103
6.16 Same as Fig. 6.1 for the Bartels rotation 2506. . . . . 104
6.17 Same as Fig. 6.1 for the Bartels rotation 2507. . . . . 105
6.18 Same as Fig. 6.1 for the Bartels rotation 2508. . . . . 106

6.19 Same as Fig. 6.8 with missing data filled with the prediction of the implemented artificial neural network model. . . . . 107
6.20 Same as Fig. 6.13 with missing data filled with the prediction of the implemented artificial neural network model. . . . . 108
6.21 Same as Fig. 6.14 with missing data filled with the prediction of the implemented artificial neural network model. . . . . 109
6.22 Same as Fig. 6.16 with missing data filled with the prediction of the implemented artificial neural network model. . . . . 110
6.23 Same as Fig. 6.22 with artificial neural network predictions performed by considering as input flux normalisation the preceding predictions of the same model. . . . . 111

# List of Tables


4.1 Observed sunspot number and solar modulation parameter ($\phi$) during the LISA Pathfinder mission. . . . . 43
4.2 Parameterisations of proton and helium energy spectra at the beginning and the end of the LISA Pathfinder mission. . . . . 46
4.3 Neutron monitor station characteristics. . . . . 49
4.4 Average characteristics of galactic cosmic-ray recurrent variations observed with LISA Pathfinder. . . . . 52

5.1 Mean squared error calculated on neural networks with an only hidden layer with a different number of neurons and for various learning rates. . . . . 60
5.2 The same of Table 5.1 for neural networks with two hidden layers. . 63
5.3 Complexity and mean squared error measured on validation set for the neural networks trained with the best hyper-parameters found with the grid search. . . . . 65
5.4 Mean squared error calculated on the validation set for neural networks with different architectures and learning rates varying the sampling rate of the input variables. . . . . 69
5.5 Complexity and mean squared error measured on validation set for the best optimised neural networks described in the paragraph. . . . 72
5.6 Mean absolute error of the predictions made by the variation of ITER with respect to the underlying neural network and the real data. . . . . 78
5.7 Mean absolute error of the predictions of the proposed model with respect to the underlying neural network and the real data. . . . . 83

6.1 Mean absolute error and standard deviation of the predictions made with the three described models with respect to the data of the test set. . . . . 88

# List of Algorithms


1 The *Rule-extraction-as-learning* algorithm. . . . . 26
2 The TREPAN algorithm. . . . . 27
3 The REFANN algorithm. . . . . 30
4 The N2PFA algorithm. . . . . 31
5 The ITER algorithm. . . . . 32
6 Modified fourth step of the ITER algorithm. . . . . 33
7 Algorithm for the creation of an explainable model that uses the main ITER concepts and a linear regressor. . . . . 80

# Chapter 1

# Introduction

Monitoring the galactic cosmic-ray flux short-term variations is a main issue for all those space missions for which cosmic rays represent a limitation to the on-board instrumentation performance. For these missions the study of the instrument efficiency requires the analysis of the cosmic-ray flux variations. When no proper particle detectors are flown on the satellite, models are used to obtain estimates or predictions of galactic cosmic-ray fluctuations.

Nowadays, artificial neural networks are among the most used tools to solve a wide range of different tasks. Neural networks learn relationships between input and output data, modeling them as variables of a function that can be either linear or not. However, neural networks solve problems as black boxes, so it is not possible to understand how they work and how they combine data to obtain the final result. These tools are used in many application fields, such as predictions and decision making. Since there are critical applications in which human beings must know how to interpret the network response, different methods were developed to extract knowledge from the aforementioned black boxes.

Several works showing rule extraction methods applied to neural network models exist in the literature. The described neural networks are used to solve problems belonging to a wide range of areas, such as financial analysis or bio-medical field. For instance, in [1, 2, 3] the authors describe this technique for credit-risk evaluation. As for the medical field, neural networks are used to make early breast cancer prognosis predictions [4] and to help the diagnosis of hepatobiliary disorders [5], coronary artery disease or thyroid dysfunctions [6], as well as to determine the type of dermatological diseases or to take decisions about liver diseases or diabetes [7]. All these papers refer to rule extraction mechanisms adopted to explain neural networks solving medical tasks. The lists are not exhaustive: rule extraction algorithms are applied also to neural networks for credit card screening [8], detecting intrusions in computer networks [9], keyword extraction [10] and many other areas.

The goal of this work is to design and implement an explainable predictor of the

galactic cosmic-ray flux short-term variations. In the pursuit of this objective, this thesis provides manifold contributions. First of all, the design and implementation of a neural network model for time-series analysis aimed to the prediction of the cosmic-ray flux variations in an opaque way, by using observations of solar wind speed and interplanetary magnetic field intensity as input variables. This work also reports the best pre-processing method of the raw input data in order to obtain suitable data sets for the neural network training and testing phases. The model is trained to mimic the measurements performed on board the European Space Agency (ESA) LISA Pathfinder space mission [11, 12], a satellite carrying instruments for gravitational wave detection in space and a particle detector. The LISA Pathfinder spacecraft was orbiting around the first Lagrangian point at 1.5 millions km from Earth between the end of January 2016 and July 2017. The input variable observations of solar wind speed and interplanetary magnetic field intensity are carried out by other space missions.

The second contribution of this work is the the design and implementation of a knowledge extraction procedure aimed to replace the aforementioned neural network with a human-intelligible rule-based predictor. This implies a deep literature analysis in order to select the best extraction strategy, resulted in the choice of the ITER algorithm. Moreover, two novel variants of the ITER algorithm tailored on the prediction task at hand are proposed and discussed.

Finally, the predictive performances of the two predictors are numerically assessed and compared with respect to the test set, entirely composed of samples never used during the training phase of the neural network model. The robustness of the proposed predictors is then evaluated also with respect to the LISA Pathfinder missing data.

**Thesis structure.** This thesis work is structured as follows. In Chapter 2 the use of machine learning models, and in particular of neural networks, is discussed to solve different problems, such as classification and regression. In Chapter 3 methods and techniques to extract rules from black box models for understanding the model behaviour are described, with particular attention to neural networks. In Chapter 4 the ESA LISA Pathfinder space mission is presented as a case study for which neural networks are used to predict the cosmic-ray data gathered on board the satellite by using as input variables the solar wind speed and the interplanetary magnetic field intensity obtained from measurements taken by the National Aeronautics and Space Administration (NASA) Wind experiment orbiting also around the first Lagrangian point L1. The developed model is described in Chapter 5, together with the knowledge extraction methods applied to it. The results of this work are reported in Chapter 6. Finally, in Chapter 7 the main results of this thesis work are summarised.

# Chapter 2

# Machine Learning and Artificial Neural Networks

Machine learning is an application of artificial intelligence that provides systems of the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data, known as *training data*, and use them to automatically learn [13, 14, 15]. Machine learning approaches are traditionally divided into three broad categories, depending on the nature of the feedback available to the learning system. The categories are: *supervised learning*, where example inputs are given to the system as well as the expected outputs and the goal is to learn a general rule that maps inputs to outputs; *unsupervised learning*, where no labels are given to the learning algorithm, leaving the algorithm to find automatically the structure in its input; *reinforcement learning*, where a computer program interacts with a dynamic environment in which it must perform a certain goal.

Performing machine learning involves creating a model, which is trained on the training data and then can process additional unknown data to make predictions. Various types of models are used and researched for machine learning systems, such as artificial neural networks.

In the following general aspects of machine learning models are reported; artificial neural networks are discussed in particular.

## 2.1 General aspects of machine learning models

### 2.1.1 Typical problems and inputs

Typical problems resolved by using machine learning models are *classification* and *regression*. In a (multi-class) classification problem the goal is to assign a class label

to an object of interest. These objects are called examples, samples or instances and are composed of attributes or features. Features can be continuous or discrete; the former can take any value in an infinite domain, while the latter are from finite domains. In addition, discrete features can be ordinal, if their values can be ordered, or categorical. Formally, an instance $\boldsymbol{x}$ described by $k$ attributes can be defined as:

$$\boldsymbol{x} = (x_1, x_2, ..., x_k) \in \mathbb{D}, \tag{2.1}$$

where $\mathbb{D}$ is defined as $A_1 \times A_2 \times ... \times A_k$ and $A_i$ is the domain of the $i$-th feature.

In a classification problem the model assigns a category at every unknown instance. Formally, each instance $\boldsymbol{x}$ has a class label $y$, where $y$ is one of the $u$ elements of the set of the different classes of the problem $\mathbb{L} = \{y_1, y_2, ..., y_u\}$. Mathematically, a classification problem consists in the definition of a function that maps input examples to output classes [16]:

$$f : \mathbb{D} \to \mathbb{L}. \tag{2.2}$$

It is possible to split classification problems into two different categories: *multi-class classification* if there are more than two classes or *binary classification* if there are exactly two classes. A variation of the standard multi-class classification problem is the *multi-label classification problem*, where the output is a subset of the possible classes rather than a single one, i.e. each instance could belong to more than one class [17]. Another variant is the *probabilistic classification*, where the output is composed by a probability for each class, representing the probability of the input instance to belong to a specific class.

In regression problems the goal is to calculate a numeric output value rather than a discrete one (as for category labels in classification). In this case the mathematical formulation of the problem is a function that maps input examples to real values:

$$f : \mathbb{D} \to \mathbb{R}. \tag{2.3}$$

It is possible to have integer values in place of real ones; moreover, a multi-class classifier can be adopted to solve regression problems by splitting the output domain into disjoint classes. For instance, if the goal is to solve a regression in the [0, 1] interval, this interval can be split into $n$ disjoint classes, such as 10: [0, 0.1), [0.1, 0.2), ..., [0.9, 1.0]. Then each output value is replaced by the pertinent class and the problem is resolved as a multi-class classification. In this case the output cannot be an exact value, but an interval.

### 2.1.2 Splitting of the data set

Given a data set, it is a good practice to split it in subsets in order to better estimate the accuracy of the model. So, first of all it is necessary to select and keep separate a certain amount of examples, such as 20% of the total number of instances, and never use it during the training phase. This set is called test set and will be used only after the training to measure the accuracy of the model with completely unknown examples. Generally the test set is chosen randomly, so as to take instances with different characteristics that reflects the whole data set. By not using these examples for the learning phase it is possible to prevent the occurrence of over-fitting problems, because the instances of the test set are completely unknown for the model, so there it was no chance to memorise such data for a good performance in later predictions (see Section 2.1.3).

The remaining examples that are not part of the test set can be used as a unique training set or can be split again between training set and validation set. The validation set is used to check the optimal values of the model hyper-parameters and it could be chosen in different manners. The simplest way is to randomly take a portion of the training set and not use it in the learning phase. In this case the validation set is fixed and used during all the proofs. Another approach consists in applying the so-called $k$-fold cross-validation [18]: the training set is divided into $k$ disjoint folds of equal size $S_1, S_2, ..., S_k$, where the union of all the folds is the whole training set. Then each partition $S_i$ is selected as validation set while the remaining $k-1$ are the training set. The model is trained with this latter and the accuracy is measured on the former. This is repeated $k$ times, each one with a different partition as validation set, then the output accuracy averaged on the $k$ folds is calculated together with the standard deviation of all the accuracies measured during the cross-validation. An extension of regular cross-validation is stratified cross-validation [19]. In $k$-fold stratified cross-validation the training set is partitioned into $k$ folds such that each class is uniformly distributed among the $k$ folds. The result is that the class distribution in each fold is similar to that in the original training set. In this sense, the partition is *balanced* in terms of class distributions. In contrast, regular cross-validation randomly partitions the samples into $k$ folds without considering class distributions. A possible scenario with regular cross-validation is that a certain class could be distributed unevenly, i.e. some folds contain more cases of the class than other folds. This distortion in class distributions can cause a less reliable accuracy estimation. In [20] is reported that stratified cross-validation performs better than regular cross-validation, having smaller bias and variance. Stratification is used mainly for classification problems, but can be applied with regression, too.

In any case, it is necessary to repeat these operations for each possible combination of the hyper-parameter values (or, due to time issues, for a subset of these).

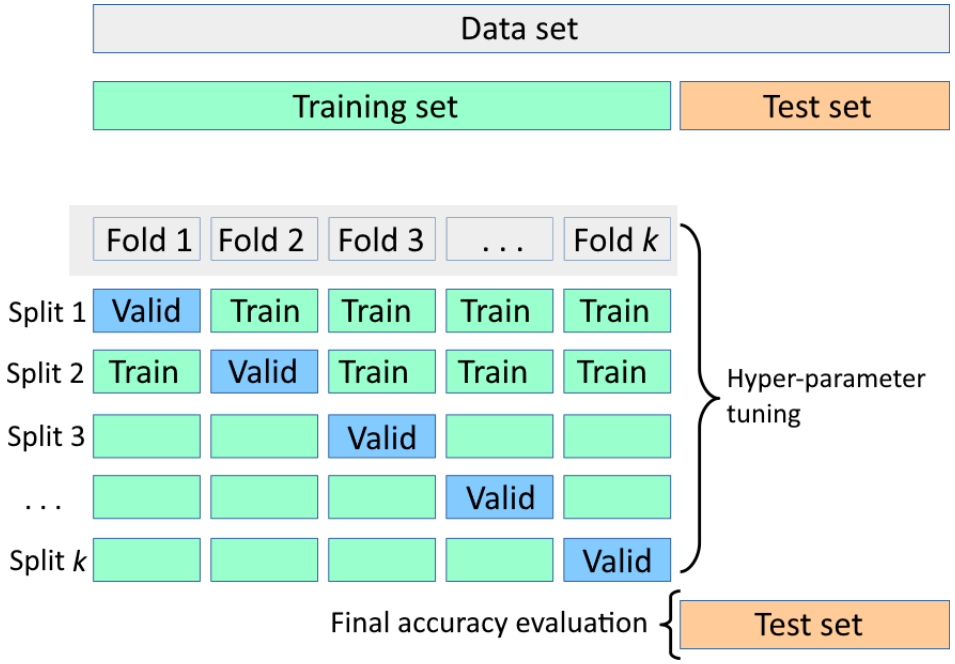



Figure 2.1: Representation of data set splitting into training and test sets. The training set is also used as validation set through the $k$-fold cross-validation.

As a final step, it is possible to select the best values for the hyper-parameters by checking which combination presents the best accuracy, then the model is trained again with the entire training set (no more validation set is needed) and the best selected hyper-parameters. Finally, the output accuracy is measured on the test set. This method is graphically resumed in Figure 2.1.

### 2.1.3 Under-fitting and over-fitting

By observing the prediction error on both the training and test sets it is possible to identify two common issues caused by a lack or an excess of model complexity, i.e. under-fitting and over-fitting [21]. A graphical representation of these problems is reported on Fig. 2.2.

**Under-fitting.** Under-fitting occurs when a machine learning model is unable to reduce the error for either the test or training set. This is caused by an insufficient capacity of the model; that is, it is not powerful enough to fit the underlying complexities of the data distributions. An endless decrease of the test prediction error over the iteration number highlights this kind of problem; it could be resolved with changes in the network architecture, such as the number of hidden layers and

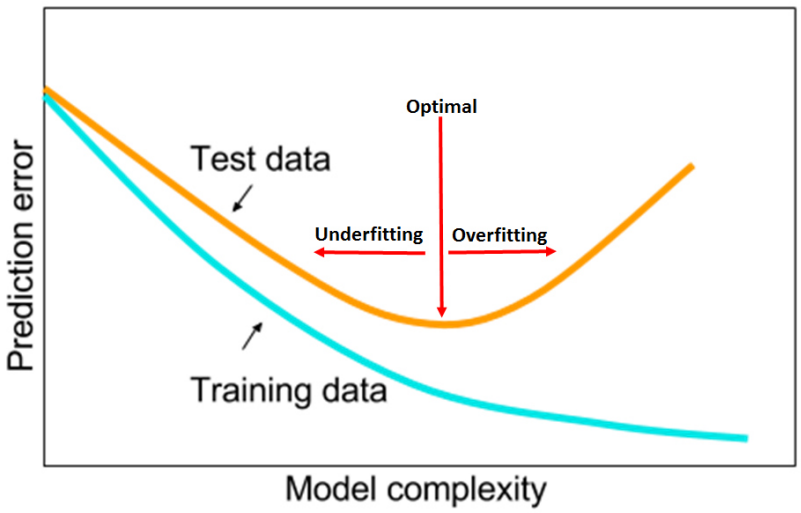



Figure 2.2: Effects of under-fitting and over-fitting with respect to the prediction error on training set (cyan line) and test set (orange line). Model complexity (reported on the $x$ axis) refers to the capacity or powerfulness of the model. The optimal capacity falls between under-fitting and over-fitting.

neurons, or by using different activation functions or data pre-processing.

**Over-fitting.** Over-fitting happens when the machine learning model is so powerful as to fit the training set so well that the noise and the peculiarities of the data are memorized. By defining the generalisation error as the loss measured on the validation or test set minus the loss measured on the training set, a large generalisation error implies good accuracy on training set and poor performance on validation or test set [22]. Over-fitting causes an increase of the generalisation error. In other words, if a neural network over-fits training data, there will be a lack of generalisation and thus low accuracy with different, unknown data. Too many hidden neurons or hidden layers, as well as a too small learning rate, can cause this problem. Over-fitting is pointed out by an increase of the test prediction error over the iterations number. An example is reported in Figure 2.3.

### 2.1.4 Time series processing

Some applications require input data in the form of variables extracted from a time series. A time series is defined as a series of data points time indexed. Generally, a time series is a sequence taken at successive equally spaced points in time. Thus it is a sequence of discrete-time data. Time series can exhibit different characteristics, such as a trend or a seasonality (i.e. an increasing or decreasing tendency with

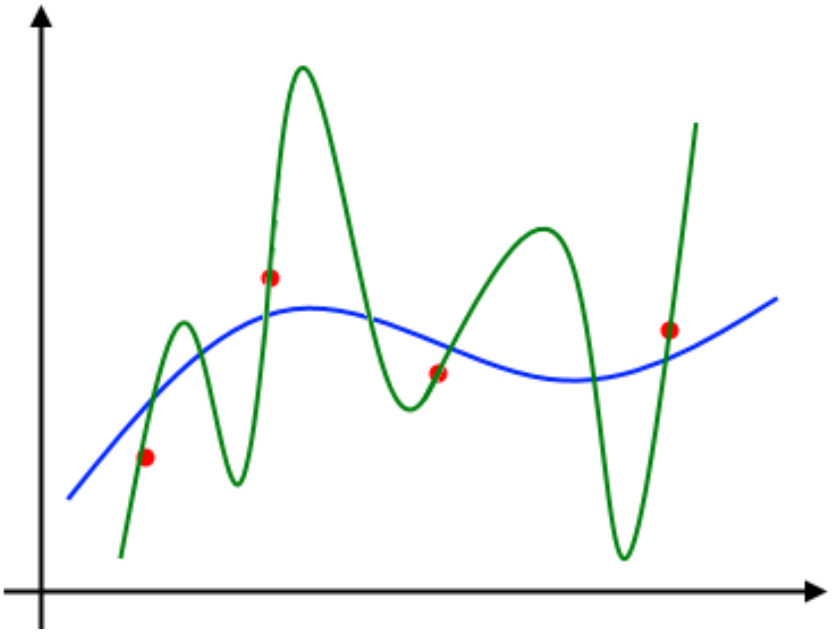


Figure 2.3: Example of an over-fitting situation. Red dots are the data points that have to be interpolated. The blue line is a good interpolation, while the green line is a bad one. However, the accuracy of the green line is better than the other, because it fits perfectly every data point.

respect to time or a fixed pattern repeated in time, respectively). Time series are useful when past observations of a variable of interest can be studied to make predictions about the future behaviour of the variable. This forecast is performed by using some function of the data collected in the past. The main issue to handle is to understand how many past observations enter into the forecast, since each variable has its own peculiarities.

If there is a single time-dependent variable, the time series is called *univariate*; otherwise it is called *multivariate*. In the univariate case the value of the studied variable is only related to its predecessors in time. Multivariate time series have more than one time-dependent variable and each variable depends not only on its past values but also depends on other variables. This dependence is used for forecasting future values [23].

Models for time series data can have many forms and represent different stochastic processes. When modeling variable variations, three broad classes of practical importance are the autoregressive models, the integrated models and the moving average models. Other models have been developed, merging the features of two or more of the aforementioned classes [24]. In addition, it is possible to adopt machine learning models to deal with time series. In any case, the variable of an

univariate time series needs to be unrolled in order to obtain a vector of fixed size. This step is required since the majority of machine learning models cannot accept as input vectors of variable size. Formally, this approach to forecast may be expressed as follows. Let $y_t$ denote the value of the variable of interest $y$ in a period $t$. Then a prediction $\hat{y}_T$ of the variable $y$ for period $T$, made with observations belonging to the time interval between the periods $T-h$ and $T-1$, may have the form:

$$\hat{y}_T = f(y_{T-1}, y_{T-2}, \ldots, y_{T-h}), \tag{2.4}$$

where $f(\cdot)$ denotes some suitable function of the past observations $y_{T-1}$, $y_{T-2}$, ..., $y_{T-h}$. For instance, $f(\cdot)$ can be a linear function. A similar unrolling is required for predicting variables belonging to multivariate time series. Let $y_{1,t}$, $y_{2,t}$, ..., $y_{K,t}$ denote the $K$ distinct variables of the time series in a period $t$. The prediction $\hat{y}_{k,T}$ of the variable $y_k$ for period $T$, made with observations belonging to the time interval between the periods $T-h$ and $T-1$, may have the form:

$$\hat{y}_{k,T} = f_k(y_{1,T-1}, y_{2,T-1}, \ldots, y_{K,T-1}, y_{1,T-2}, \ldots, y_{K,T-2}, \ldots y_{1,T-h}, \ldots, y_{K,T-h}). \tag{2.5}$$

In Equation (2.4) and Equation (2.5) the size of the time window including the past observations of the variables is not defined. The size depends on the problem to solve and on the variables involved. In the multivariate case it is possible to choose different sizes for distinct variables. The above equations assume that within a time window all the observations are used, but it is possible to choose a different granularity, for instance selecting one every $n$ observations and discarding the remaining $n-1$. With multivariate time series it is possible to choose a different granularity for each variable.

## 2.2 Artificial neural networks

Artificial neural networks are mathematical systems loosely modeled on the human brain and inspired by the organization and functioning of biological neurons. The main concept resides in learning from experience. They try to emulate biological neurons by multiple layers of simple processing elements called artificial neurons. Each neuron is linked to all or a part of its neighbours and each link (called connection) has a coefficient of connectivity (called weight) that represents the strength of the connection. Neural network weights are randomly initialised; the learning process consists in adjusting these values so that the final output of the network will be as much as possible close to the expected result [25].

There are numerous artificial neural network variations, related to the nature of the task assigned to the network. There are also numerous variations in how the neurons are modeled. Some models correspond closer than others to biological neurons [26, 27].

Artificial neural networks can approximate both linear and non-linear functions and can also estimate piece-wise approximations of functions [28]. In the last decades artificial neural networks became increasingly central in information technology research thanks to their ability to solve complex problems with high degree of accuracy, even in fields where traditional methods fail, such as forecasting and decision modeling [29].

Despite this advantage, neural networks often require a lot of experience to effectively choose optimal hyper-parameters, regularisation parameters and network architecture, which are all tightly coupled. Since there are no simple and easy ways to set hyper-parameters, currently this process is mostly based on extensive trial and error, because a grid or random search of the hyper-parameter space is often computationally expensive and time consuming [30]. Examples of hyper-parameters are activation functions, (see Section 2.2.2), network architecture (as discussed in Section 2.2.4) and learning rate (Section 2.2.9).

### 2.2.1 Artificial and biological neurons

Neural networks are composed by a large number of elemental components called neurons. Since their structure is modeled after that of the brain, neural networks have a strong similarity to the biological brain and therefore a great deal of the terminology is borrowed from neuroscience.

**Biological neurons.** The basic signaling unit of the nervous system is the neuron. The brain contains billions of them; the best estimates carried out for adult human brains are of the order of $10^{11}$ neurons. The interactions between them enable people to think, remember, move, maintain homeostasis and feel emotions. A neuron is a specialized cell allowing for different actions because of precise connections with other neurons, sensory receptors and muscle cells. Each neuron can connect with up to $2 \cdot 10^5$ other neurons [31, 32, 33]. The basic neuron consists of synapses, soma, axon and dendrites as shown in Fig. 2.4 [34]. Synapses are links between neurons that permit electric signals to pass from one to another. These electrical signals are then jumped across to the soma, which carries out some operations and sends its own electrical signal to the axon. The axon then disperses this signal to dendrites. Dendrites send the signals to the different synapses, and the process is reiterated [35]. In other words, a biological neuron receives inputs from other sources, combines them and performs a generally nonlinear operation and then outputs the final result.

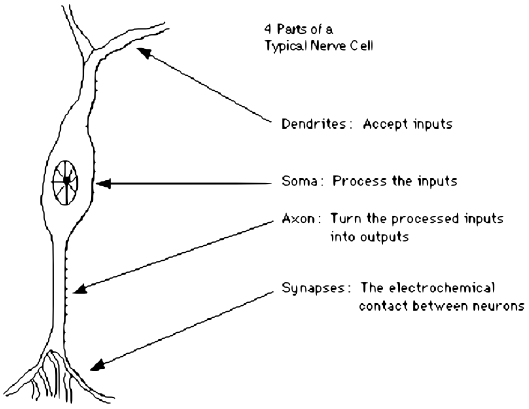



Figure 2.4: Simplified biological neuron and relationship of its four components.

**Artificial neurons.** The basic unit of neural networks is the artificial neuron and it simulates the four basic functions of natural neurons, even though they are much simpler than the biological counterpart. Figure 2.5 shows the basics of these components.

Every neuron of the network has different simultaneous inputs, each of which has a weight connection assigned to it. Weights are correlated with the significance of the correspondent input. The global input of the neuron, called net value, is the weighted sum of all the inputs multiplied by their respective weights. Each neuron has its own distinctive threshold value, and if the output is greater than the threshold, then the neuron is called fired, otherwise it is not fired. The output is then fed into all the neurons of the following layer.

There are alternative ways to calculate the net value instead of using the weighted sum. For example, the aggregation function can select minimum, maximum, majority or product.

Figure 2.5 shows $n$ inputs and weights of a generic neuron (represented by $x_i$ and $w_i$, respectively); each input $x_i$ is multiplied by the correspondent weight $w_i$. The net value is obtained by summing up all these products and an additional parameter called bias (represented in figure as $b$). The output value of the neuron is equal to the transfer function ($\sigma$) applied to the net value [36].

### 2.2.2 Activation functions

The activation function, or transfer function, is a core element of artificial neurons. It transforms the activation level of a neuron into an output signal, enabling nonlinear mappings of the data so that they can be more comprehensible for the network. For example, it can make the data linearly separable for a better classification.

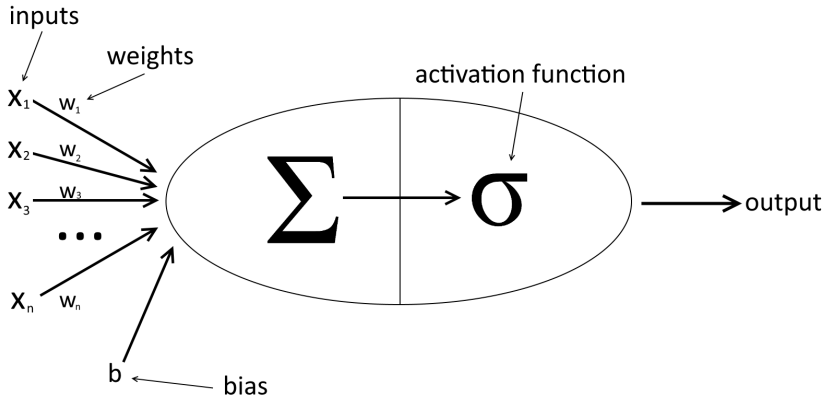



Figure 2.5: Artificial neuron.

The choice of the correct activation function depends on the kind of problem that has to be solved. It is well known that sufficiently elaborated networks can approximate any function of interest thanks to the activation function [37, 38]. In [39] the authors reported that it is possible to approximate any continuous function to any degree of accuracy if and only if the network adopts an activation function that is not polynomial. In the literature there are many works discussing the advantages of the sigmoid function [40, 41, 42, 43], although a wide selection of other useful functions exists, both nonlinear such as the hyperbolic tangent [44, 45] or the rectifier function [46, 47, 48], and linear [49]. Other non-conventional functions have been proposed in the literature, such as in [50].

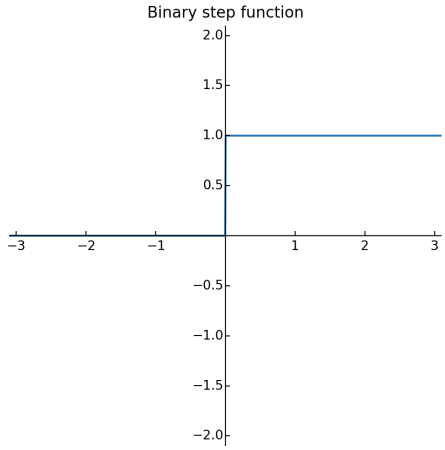
Interesting dissertations about the difference between the various activation functions can be found in [51] and [52].

**Binary step function.** The binary step function is the simplest activation function that can be implemented with a simple if-else statement. It represents a neuron that is activated only if its input is larger than a certain threshold. It is suitable for binary classification but not for multi-class classification. Its main drawback is that its gradient is zero; this could be a problem during the training back-propagation phase.

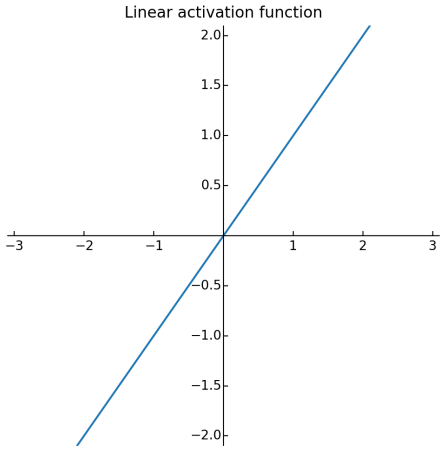
If $t$ is the threshold, this function can be defined as:

$$f(x) = \begin{cases} 1 & \text{if } x \geq t, \\ 0 & \text{otherwise.} \end{cases} \tag{2.6}$$

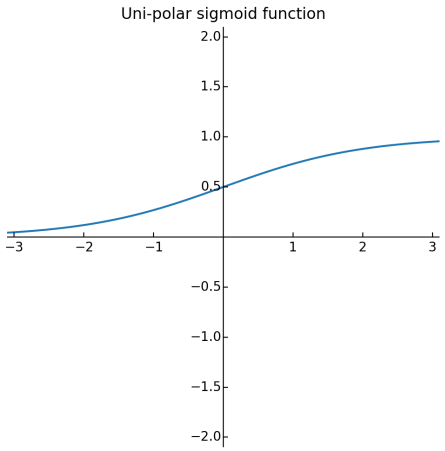
Figure 2.6a shows this function with a threshold $t = 0$.



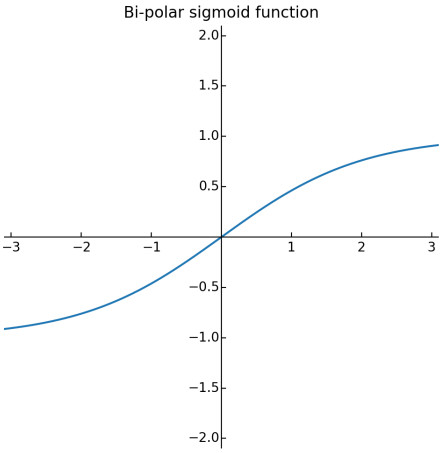

(a) Binary step function with $t = 0$.



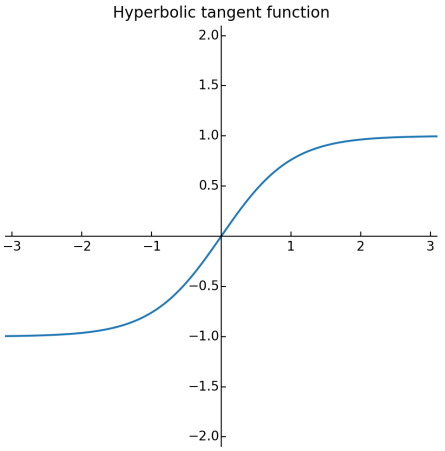

(b) Linear function with $a = 1$ and $b = 0$.



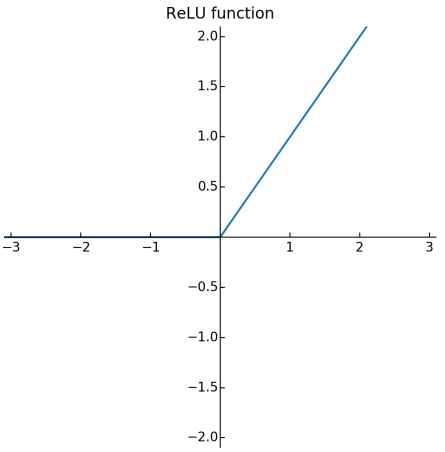

(c) Uni-polar sigmoid function.




(d) Bi-polar sigmoid function.




(e) Hyperbolic tangent function.




(f) ReLU function.

Figure 2.6: Examples of activation functions.

**Linear activation function.** The linear activation function is directly proportional to the input. It removes the zero gradient problem of the binary step function. It can be defined as:

$$f(x) = ax + b. \tag{2.7}$$

Figure 2.6b shows this function with parameters $a = 1$ and $b = 0$. With this values the linear function is an identity function.

**Sigmoid functions.** Sigmoids are the most widely used activation functions, since they are non-linear functions continuously differentiable. This feature makes them especially advantageous to be used in neural networks trained by back-propagation algorithms. The term sigmoid means 'S-shaped'.

**Uni-polar sigmoid function.** The uni-polar sigmoid function maps the interval $(-\infty, \infty)$ onto (0, 1) and is defined as:

$$f(x) = \frac{1}{1 + e^{-x}}. \tag{2.8}$$

The uni-polar sigmoid function is mainly used for models that have to provide an output prediction in terms of probability. Since the probability ranges between 0 and 1, uni-polar sigmoid is the most suitable choice. This function is shown in Figure 2.6c.

**Bi-polar sigmoid function.** This function differs from the uni-polar one because it is symmetric with respect to the origin and produces output in the range (-1, 1). It is used for applications that produce output values in the same range and it is defined as:

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}. \tag{2.9}$$

Figure 2.6d shows the bi-polar sigmoid function.

**Hyperbolic tangent.** The hyperbolic tangent is defined as the ratio between the hyperbolic sine and cosine functions or expanded as the ratio of the half-difference and half-sum of two exponential functions in the points $x$ and -$x$ as follows:

$$\tanh x = \frac{\sinh x}{\cosh x} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{2.10}$$

As shown in Fig. 2.6e, this function is similar to a sigmoid, but has a more steep, zero centered gradient. It has the same output range of the bi-polar sigmoid function and it is mainly used for classification between two classes.

**ReLU function.** ReLU stands for rectified linear unit and is a non-linear activation function widely used in neural networks. It is more efficient than other functions because not all the neurons are activated at the same time; in addition, in some cases the value of the gradient is zero, so weights and biases are not updated during back-propagation. It can be defined as follows:

$$f(x) = \max(0, x). \tag{2.11}$$

This function is reported in Fig. 2.6f.

### 2.2.3 Multi-layer perceptrons

Neural networks are composed of connected artificial neurons, but the clustering of these neurons is a fundamental issue in the network construction. Generally they are grouped in layers, where only neurons in successive layers are connected. The basic architecture, called perceptron, is composed of an input layer and an output one. The first has a number of neurons equal to the number of data set features, while the latter has a neuron for each output value of the network. For example, a simple classifier, that returns a label representing the class of a sample, will have a single output neuron, as also a regressor for predicting real values. The input value of input neurons is the feature value (opportunely scaled with respect to the problem, usually between 0 and 1, or -1 and 1). In this basic architecture, inputs are processed in the input layer and the relative output, after the application of the activation function, is fed to the output layer.

A more complex architecture consists of one or more additional layers put between the two described above. These are called hidden layers. In this case, the output of the input layers is sent to the first hidden layer. Hidden neurons do similar operations on their input (weighted summation of the values and application of an activation function) and provide output to successive hidden layers. The output of the last hidden layer is fed to the output layer. It can be seen that the propagation of the input throughout the layers creates a feed-forward path to the output [53]. Figure 2.7 shows a multi-layer perceptron with two hidden layers. In the example there are an input layer with eight neurons, two hidden layers with five and two hidden units, respectively, and finally an output layer with only one neuron. Biases are not reported in the figure to keep it clear, while all the connections are drawn. As examples, only some weights are reported, such as $w_{31}$ for the connection between the input unit 1 and the hidden unit 3, $w_{34}$ for the connection between the hidden units 3 and 4 and finally $w_{54}$ for the connection between the hidden unit 4 and the output unit 5.

It is possible to have connections starting from a neuron in a layer and terminating in neurons of a previous layer; these are called feedback. Inhibitions,

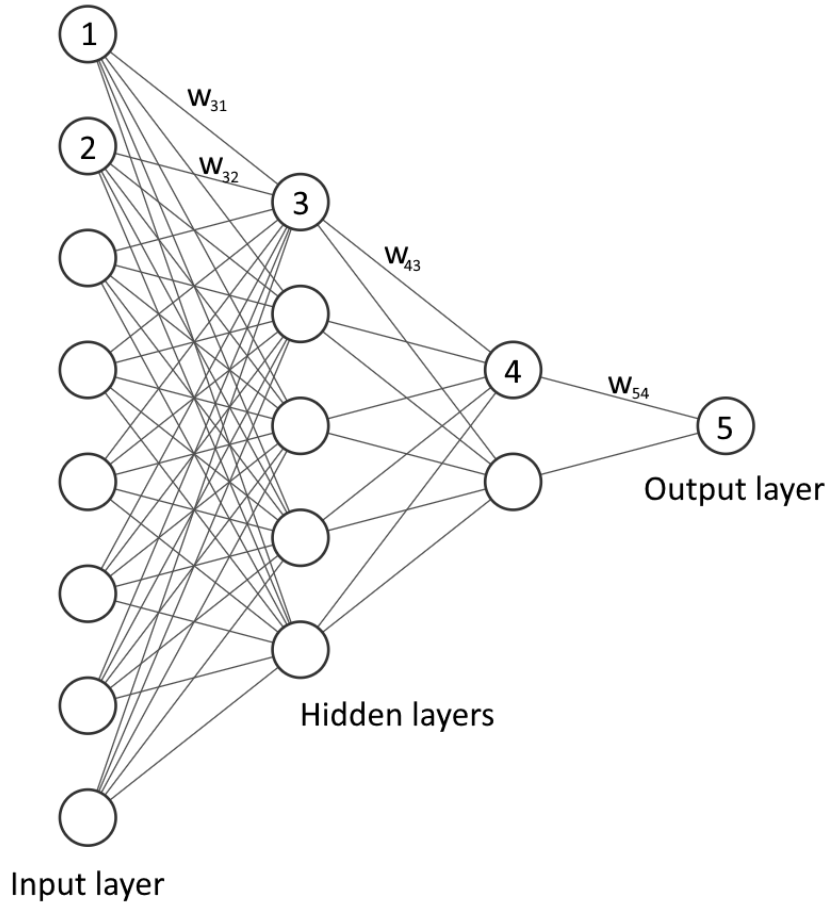



Figure 2.7: Representation of a multi-layer perceptron with two hidden layers. Biases are not reported to keep the figure clear.

or lateral inhibitions, are connections between neuron of the same layer, used to inhibit a neuron if there is another neuron with higher value. They are usually applied in output layers of classifiers, to deactivate neurons with non-maximum probability. This mechanism is also called competition.

### 2.2.4 Hidden layers and hidden units

The number of hidden layers and that of hidden neurons (or units) is crucial for different reasons. First of all, adding layers allows the network to learn more complex mappings between input and output. However, too many of them could lead to over-fitting problem (see Section 2.1.3) and protract the training time. Use only one hidden layer is faster, but the network could be not sufficiently complex to model the problem (see the under-fitting problem in Section 2.1.3). For example,

it has been shown that a single hidden layer feed-forward network, with sigmoid activation function in the hidden layer, can approximate an arbitrary mapping from one finite dimensional space to another [37]. Although it seems that multi-layer perceptrons can learn whichever mapping if complex enough, adding more than two or three hidden layers does not generally improve the output accuracy.

A similar conclusion is valid for hidden neurons with respect to under-fitting and over-fitting problems. The optimal number of units can be tuned with trial and error methods, even though different rules are suggested in the literature [54, 55]. There is not a valid formula that holds in every context.

### 2.2.5 Training

The training of a multi-layer perceptron is a supervised learning algorithm, consisting in two phases: the forward propagation of the samples of the training set and the successive update of the network parameters (connections weights and neurons biases) in order to reduce the prediction error, using back-propagation techniques [56]. The prediction error is calculated with respect to a defined loss function, such as the mean absolute error or the mean squared error (see Section 2.2.8). Network parameters are randomly initialised. The supervised nature of the process implies that both training input and output are provided to the model.

During the forward phase, every training sample is fed to the network, starting from the input layer. Each neuron of the layer produces an output value and this is propagated to every connected neuron all through the output layer [57]. Neurons in the output layer give the final result of the model. Since the expected value is known, it is possible to calculate the error of the model and to back-propagate such error in order to update the network parameters and reduce it [58].

### 2.2.6 Epochs and iterations

During the training phase of the neural network, an epoch represents the feeding process of each data sample of the training set to the network. An epoch comprises one or more batches, depending on the batch size. A batch is a subset of the training set. The batch size is a hyper-parameter. The number of epochs is another hyper-parameter and defines the number of times that the learning algorithm takes as input the entire training set. Too many epochs can lead to a lack of generalisation due to over-fitting.

In the training algorithm network parameters are updated after the feed of an entire batch to the algorithm, and the feeding followed by the update is called iteration. As a result, the number of iterations per epoch depends on the batch size: large sizes implies less iterations per epoch. Small batch sizes are recommended for having regularisation effects [59].

### 2.2.7 Early stopping

Over-fitting can be resolved by using a technique called early stopping. It implies the preliminary separation of the training data into two splits, the training and validation sets. The training set is used to train the network, while the validation set is only used to periodically test the accuracy of the model. During the training the prediction error is supposed to decrease on both splits; when validation error starts to grow up, the training is stopped [60]. It is suggested to use a large number of hidden neurons, slow learning rate and small random initial weights. It is possible to select a threshold and consider any improvement of the prediction error on the validation set smaller than this threshold as a marker of the worsening of the error.

### 2.2.8 Loss function

The loss function, or cost function, is a measure of how the output of a model diverges from the expected output. It can be formulated in different ways, depending on the problem to resolve. For example, cross-entropy is a loss function used in classification problems, where the output is a label, while mean absolute error or mean squared error are functions used in regression problems. The mean absolute error (MAE) is defined as the average absolute difference between the model output $\hat{\boldsymbol{y}}$ and the expected output $\boldsymbol{y}$:

$$MAE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i|, \tag{2.12}$$

where $n$ is the number of test samples $y_i$ and corresponding model outputs are $\hat{y}_i$. Conversely, the mean squared error (MSE) is defined as follows:

$$MSE(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2. \tag{2.13}$$

.

Of course, a perfect model would have a loss equals to zero.

### 2.2.9 Learning rate

This hyper-parameter determines the size of the step taken during the training to move toward a minimum of the used loss function. This value must be carefully tuned: if too large it is possible to jump over the minimum, but if too small there are different problems, such as a very long time for convergence or a stop in a suboptimal local minimum. The tuning of this hyper-parameter is intertwined

with that of the others; for example, it is found that it is possible to increase the batch size instead of reducing the learning rate without losses in test accuracy [61].

## 2.3 Pruning in neural network models

Artificial neural networks tend to have an elevate number of neurons and, thus, of trainable parameters. This implies high computation, energy and memory costs during the learning phase, but also higher probability of over-fitting the training data and a more difficult procedure to extract a set of rules from the model. There are different kinds of pruning, based on different criteria, aimed to make the network smaller and faster. The majority is based on the removal of neurons or input features that make little or no contribution to the output of a trained network. Obviously, by removing part of the network complexity there is a drawback regarding the model accuracy. The more the model is pruned, the more it can lose precision.

A method to perform pruning is described in [62] and consists in estimating the contribution of a neuron with respect to the final loss and the iterative removal of those with smaller scores. The algorithm applies an iterative fine-tuning process with small learning rate. During each epoch it alternates two steps. The first consists in computing the importance of each neuron for each batch of samples. After a predefined number of batches, the second step takes place: it consists in the removal of the less important neurons. The algorithm stops when the target number of neurons is pruned or when the accuracy drops below a certain threshold.

The method reported in [63] is based on the sensitivity estimation of the error function to the exclusion of each neuron. This estimation is performed during the network training in a non-interfering manner, so at the end of the process it is possible to prune the connections with smaller sensitivity.

In [64] connections (and thus neurons with no input or output connections) identified by small weights are removed without loss of accuracy.

Other approaches, called brute-force pruning, exclude one connection at a time and evaluate if the network without the inhibited connection have the same (or a comparable) accuracy. If it is the case, the connection is pruned [65].

Pruning algorithms are an alternative to constructive algorithms. While the former start with a sufficient number of neurons and then remove units iteratively until a criterion is met, the latter start with few neurons and then add units and layers until the obtained results are acceptable.

Some pruning methods require an *ad hoc* training. An example are magnitude-based approaches, in which the loss function is modified by adding a term to push the network towards weights near to zero during the training.

Pruning or constructive approaches can be applied also to input features, in

order to minimize the number of units in the input layer. From the point of view of the knowledge extraction it is an important issue, because less input variables implies simpler rules.

# Chapter 3

# Knowledge Extraction from Neural Networks

Decision support systems constructed as black boxes are more and more widely used. A black box model is a system that hides its internal logic to the user; this lack of knowledge constitutes a problem, especially in critical applications such as to the field of public health and economy. The literature reports approaches aiming to provide a possibility to interpret the model results which are preferred over those that do not allow any interpretation despite being characterised by a good accuracy.

## 3.1 Interpretable models

The term *interpretability*, applied to machine learning systems, is referred to the ability to explain or to present a model in an understandable way for humans [66].

Given a black box it is possible to distinguish between *reverse engineering* or *design* of explanations. In the first case given the decision records produced by a black box decision maker the problem consists in reconstructing an explanation for it. In the second case, given a data set of training decision records the task consists in developing an interpretable predictor model together with its explanations. In the reverse engineering field, there are different kinds of problems, described in the following using as a reference a black-box classifier [67].

### 3.1.1 Black box problems

**Black box model explanation problem.** Given a black-box classifier this problem consists in providing an interpretable and transparent model which is able to mimic the behaviour of the black box and which is also understandable by

humans.

**Black box outcome explanation problem.** Given a black-box classifier this problem consists in providing an interpretable outcome, that is a method for providing an explanation for the outcome of the black box. In other words, the interpretable model must return the prediction together with an explanation about the reasons for that prediction. It is not required to explain the whole logic behind the black box but only the reasons for the choice of a particular instance.

**Black box inspection problem.** Given a black-box classifier this problem consists in providing a representation for understanding either how the black box model works or why the black box returns certain predictions more likely than others.

### 3.1.2 Explanator types

Different kinds of explanators can be used to open a black box. Their nature differs on the basis of how the explanator is constructed and presents the output to the user. Decision rules and decision trees are the most widespread and human understandable models.

**Decision rules.** Decision rules are functions which map observations to appropriate actions and can be extracted by generating the so called classification rules, i.e. association rules that in the consequence have the output class label [68, 69]. The most common rules are *if-then rules* where the *if* clause is typically a conjunction of conditions on the input variables, event though is possible to use negations and disjunctions too. An improvement of these rules are *m-of-n rules*, where, given a set of $n$ conditions, if $m$ of them are verified then the consequence of the rule holds [70].

**Decision trees.** This model is based on a graph structured like a tree and composed of internal nodes that represent tests on attributes. Leafs are the output class labels of a black-box classifier or the output value of a black-box regressor. Each branch models a possible outcome and the entire paths from root to leaves represent classification rules [71]. Obviously it is possible to linearise the tree to obtain a list of *if-then rules* [72], in the form:

*if* body *then* outcome,

where the body is the conjunction of the clauses corresponding to the different node conditions in the path from the root to the outcome represented by a leaf. This must be done for each possible path.

The comprehensibility of decision trees is facilitated by several factors [73], such as their graphical structure and the fact that they typically contain a subset of attributes instead of all of them, helping users to focus on the most relevant ones. Third, the hierarchical tree structure provides information about the importance of different attributes; attributes close to the root are more relevant.

**Other explanators.** Beyond decision rules and trees there are other explanator models. *Features importance* is a very simple but effective solution consisting in returning as explanation the set of features used by the black box together with their weight [74, 75]. *Salient masks* are an efficient way of pointing out what causes a certain outcome, especially when images or texts are treated, and consist in using "masks" visually highlighting the determining aspects of the analyzed record. They are generally used to explain deep neural networks [76, 77]. *Sensitivity analysis* consists of evaluating the uncertainty in the outcome of a black box with respect to different sources of uncertainty in its inputs. It is generally used to develop visual tools for black box inspection [78, 79]. *Partial dependence plots* help in visualizing and understanding the relationship between the outcome of a black box and the input in a reduced feature space [80, 81]. *Prototype selection* consists in returning, together with the outcome, an example very similar to the classified record, in order to make clear for which criteria the prediction was returned. A prototype is an object that is representative of a set of similar instances and is part of the observed points, or it is an artifact summarizing a subset of them with similar characteristics [82, 83]. *Neurons activation* is the inspection of neural networks and deep neural network carried out by observing which are the fundamental neurons activated with respect to particular input records [84, 85].

## 3.2 Extracting knowledge from black boxes

Despite the versatility of black boxes, these models have an inherent inability to explain in a comprehensible form the process by which a given decision or a generated output has been obtained. However, in certain applications it is crucial to find an explanation for the black box response, so it is necessary to apply a method for knowledge extraction from the model. In addition, by applying these methods, it could be possible to learn unknown relationships between input and output [86].

### 3.2.1 Explanator classification.

Several methods exist for extracting knowledge from black box models and three different dimensions are used to classify them. One of these is the translucency, intended as the view of the model internal structure. Translucency is described in more detail in the next section. The other dimensions are the kind of the extracted rules (for instance *if-then* or *m-of-n* rules) and the kind of the problem to solve (i.e. a classification or a regression).

### 3.2.2 Translucency of the model

With respect to the translucency parameter, there are two possible approaches: *decompositional* and *pedagogical*. In addition there is a third approach, called *eclectic*, which combines elements of both categories.

**Decompositional approach.** The decompositional approach allows for the knowledge extraction considering the internal logic and structure of the underlying black box model. For this reason the extracting process is closely intertwined with all the peculiarities of the black box and each kind of black box requires specific extraction algorithms.

With respect to artificial neural networks, this kind of approach focuses on extracting rules at the level of individual network units, the hidden and output neurons weights and biases. Hence it provides a transparent view of the underlying network. The computed output from each hidden and output network unit must be mapped into a binary outcome (yes or no) which corresponds to the notion of a rule consequent. For this reason each hidden or output unit can be interpreted as a step function or a Boolean rule, which reduces the rule extraction problem to determine the situations in which the rule is true. The rules extracted at the individual unit level are then aggregated to form the composite rule set for the neural network as a whole.

**Pedagogical approach.** The pedagogical approach extracts rules from a black box model without making any architectural assumptions of the underlying model and it can therefore be applied to any kind of black box. This technique considers the rule extraction as a learning task where the target concept is the function computed by the black box model. The input features used for the extraction are the same input features of the underlying model.

In the case of artificial neural networks, the pedagogical approach treats the underlying neural network as a model with an opaque view of its units. The knowledge extraction is performed by learning rules that map inputs directly into outputs, regardless from the network parameters.

### 3.2.3 Quality of the extracted rules

To evaluate the quality of the rules extracted with knowledge extraction methods, different indicators can be kept in consideration: prediction performance evaluation, fidelity, consistency and comprehensibility [87]. Quality is intended as how well the extraction task has been performed, i.e. by comparing the generalization power of the interpretable model with respect to that of the underlying black box model [88].

The prediction performance evaluation is correlated to the goodness of the model output and it depends on the task to solve. In the case of classifications, the accuracy of the explanator can be used. Conversely, in regression tasks different metrics are adopted, such as the mean squared error or the coefficient of determination $R^2$.

Fidelity is about the ability of the model to mimic the behaviour of the underlying black box. Consistency holds if, under different training sessions, the model generates rule sets which produce the same classifications of unseen examples. Finally, the comprehensibility is determined by measuring the size of the model. For example, for a rule set is the number of rules and the number of antecedents per rule.

## 3.3 Extracting knowledge from neural networks

In the following only explanators producing decision rules or trees for solving the model explanation problem are reported.

### 3.3.1 Knowledge extraction from classifiers

In this section two algorithms for extracting knowledge from artificial neural network solving classification problems are described.

***Rule-extraction-as-learning.*** This algorithm is an eclectic method for extracting conjunctive rules from trained neural networks using a learning process driven by sampling and queries. Output rules can be *if-then* or *m-of-n* rules. The core idea is to view rule extraction as a learning task where the target concept is the function computed by the network and the input features are simply the network input features. As it can be seen in Algorithm 1 [89], the approach uses two different *oracles*, namely functions that are able to answer queries about the concept being learned. The first is the EXAMPLES oracle, which produces, on demand, training examples for the rule-learning algorithm. The second is the SUBSET oracle, which takes as arguments a class label $c$ and a conjunctive rule $r$ and returns

**Algorithm 1** The *Rule-extraction-as-learning* algorithm.

```
\* initialise rules for each class *\
for each class c
   R_c := ∅
repeat
   e := EXAMPLES()
   c := classify(e)
   if e not covered by R_c then
      \* learn a new rule *\
      r := conjunctive rule formed from e
      for each antecedent r_i of r
         r' := r but with r_i dropped
         if SUBSET(c, r') = true then r := r'
      R_c := R_c ∨ r
until stopping criterion met
```

*true* if all of the instances that are covered by the rule $r$ are members of the given class $c$. It returns $false$ otherwise.

The algorithm repeatedly queries the EXAMPLES oracle, each time determining the class of the returned example. If such example is not covered by the learned rule for its class, the rule is extended with an additional term in order to include the example. Class rules are disjunctive normal form expressions, while new terms are initialised as the conjunction of all of the feature values of the example. Then the rule is generalised by repeatedly dropping each antecedent and checking with the SUBSET oracle if the rule is still agreeing with the network. Dropping an antecedent means making one of the example features undetermined. If SUBSET returns *true* the antecedent is definitely removed from the rule. This is equivalent to say that the value of the feature is irrelevant in determining the output class of the example. The algorithm works until a stopping criterion is met, such as obtaining an accuracy greater than a predefined threshold with respect to a set of selected samples or when a certain number of consecutive iterations have resulted in no new rules.

The main drawbacks of this algorithm are the impossibility to handle real-valued features (it only works with categorical ones) and the low degree of readability with an high number of different classes and/or input features. This latter problem can be limited by extracting *m-of-n* rules instead of *if-then* rules.

**TREPAN.** TREPAN is a pedagogical algorithm for extracting comprehensible, symbolic representations from trained neural networks by using queries to induce a decision tree that approximates the concept represented by the given network.

**Algorithm 2** The TREPAN algorithm.

```
TREPAN(training_examples, features)
  \* sorted queue of nodes to expand *\
  Queue := ∅
  for each example E ∈ training_examples
    \* use net to label examples *\
    class label for E := ORACLE(E)
  initialise the root of the tree, T, as a leaf node
  put ⟨T, training_examples, {}⟩ into Queue
  while Queue is not empty and size(T) < tree_size_limit
    \* expand a node *\
    remove node N from head of Queue
    examples_N := example set stored with N
    constraints_N := constraint set stored with N
    use features to build set of candidate splits
    use examples_N and calls to ORACLE(constraints_N) to evaluate splits
    S := best binary split
    search for best m-of-n split, S', using S as a seed
    make N an internal node with split S'
    for each outcome s of S'
      \* make children nodes *\
      make C, a new child node of N
      constraints_C := constraints_N ∪ {S' = s}
      call ORACLE(constraints_C) to determine if C should remain a leaf
      otherwise
        examples_C := members of examples_N with outcome s on split S'
        put ⟨C, examples_C, constraints_C⟩ into Queue
  return T
```

It is able to produce decision trees that maintain a high level of fidelity to their respective networks while being comprehensible and accurate. It results to be general in its applicability and to scale well to large networks and problems with high-dimensional input spaces.

TREPAN, shown in Algorithm 2 [90], is based on the availability of an oracle that is able to answer queries during the learning process. Since the target function is simply the concept represented by the network, the oracle uses the network to answer queries. It learns directly from the training set. The role of the oracle is to determine the class (as predicted by the network) of each instance that is presented as a query. Queries to the oracle, however, do not have to be complete instances, but instead they can specify constraints on the values that the features

can take. In the latter case, the oracle generates a complete instance by randomly selecting values for each feature, while ensuring that the constraints are satisfied. In order to generate these random values, TREPAN uses the training data to model the marginal distribution of each feature. The oracle is used for three different purposes: to determine the class labels for the network training examples; to select splits for each of the tree internal nodes; to determine if a node covers instances belonging to only one class.

TREPAN grows trees using a best-first expansion. The best node is the one at which there is the greatest potential to increase the fidelity of the extracted tree to the network. The function used to evaluate node $n$ is:

$$f(n) = reach(n) \cdot (1 - fidelity(n)), \tag{3.1}$$

where $reach(n)$ is the estimated fraction of instances that reach $n$ when passed through the tree, and $fidelity(n)$ is the estimated fidelity of the tree to the network for those instances. The role of internal nodes in a decision tree is to make a partition of the input space in order to increase the separation of instances of different classes. This algorithm forms trees that use *m-of-n* expressions for its splits.

Split selection involves deciding how to partition the input space at a given internal node in the tree. A limitation of conventional tree-induction algorithms is that the amount of training data used to select splits decreases with the depth of the tree. Thus splits near the bottom of a tree are often poorly chosen because these decisions are based on few training examples. TREPAN, having an oracle available, is able to use as many instances as desired to select each split. It chooses a split after considering at least $S_{min}$ instances, where $S_{min}$ is a parameter of the algorithm. When selecting a split at a given node, the oracle is given the list of all of the previously selected splits that lie on the path from the root of the tree to that node. These splits serve as constraints on the feature values that any instance generated by the oracle can take, since any example must satisfy these constraints in order to reach the given node.

TREPAN uses two separate stopping criteria for the decision tree growth. First, a given node becomes a leaf in the tree if, with high probability, the node covers only instances of a single class. To make this decision, the algorithm determines the proportion of examples that fall into the most common class at a given node, and then calculates a confidence interval around this proportion. The oracle is then queried for additional examples. TREPAN also accepts a parameter that specifies a limit on the number of internal nodes in an extracted tree. This parameter can be used to control the comprehensibility of extracted trees, since some domains may require very large trees to describe networks to a high level of fidelity.

With respect to *Rule-extraction-as-learning*, TREPAN presents many advan-

tages. First of all, its decision trees are more readable for humans than decision rules. A second benefit is in the higher level of both fidelity and accuracy with respect to the underlying neural network and test data, respectively. In addition, TREPAN is computationally faster than *Rule-extraction-as-learning*. However, TREPAN cannot be applied to real-valued features but only to categorical ones, as the *Rule-extraction-as-learning* algorithm.

### 3.3.2 Knowledge extraction from regressors

This section is referred to algorithms for knowledge extraction from neural network used in regression problems.

**REFANN (Rule Extraction from Function Approximating Neural Networks).** REFANN is a decompositional rule extraction algorithm for regression problems [91]. The method is designed specifically for multi-layer perceptrons with one hidden layer. It assumes that the activation function used in the hidden layer is the hyperbolic tangent function (see Eq. (2.10)) and that no activation function was used for the output unit. The principal idea behind the algorithm is to approximate the hidden layer activation functions by multiple linear functions. Subsequently, these linear approximations are used to create the rules. REFANN is summarised in Algorithm 3.

Pruning of irrelevant input and hidden nodes is essential to create a network that generalizes well and to allow the extraction of a small set of comprehensible rules. The N2PFA algorithm (Neural Network Pruning for Function Approximation) (see Algorithm 4) is used for pruning the network. The main reason of selecting N2PFA over other pruning algorithms is that it removes complete neurons instead of only some connections. This will result in better comprehensibility because fewer inputs remain for inclusion in the rule conditions.

Regarding the rule condition simplification, as such conditions are based on scalar products involving the input weights, the resulting boundaries are oblique hyperplanes in the input space. To facilitate human interpretation, it might be better to replace these oblique boundaries by ones that are parallel with the axes of the input space. C4.5 algorithm can be adopted for this task by assigning each training observation a class label corresponding to the subregion it belongs to. C4.5 is then able to learn parallel decision boundaries from these labelled observations that can be used to replace the original oblique conditions.

This method is able to generate accurate results, however it presents some drawbacks too. First, the number of extracted rules grows exponentially with the number of hidden neurons. This means that the method is only applicable if the problem can be solved by a neural network that contains a limited number of hidden neurons. The second drawback is associated with all dependent approaches.

**Algorithm 3** The REFANN algorithm.

- train and prune a network with one hidden layer and one output unit
- **for each** hidden unit $i = 1, 2, ..., H$
  - determine $x_{im}$ from the training samples
  - find the interesting points of the $n$-piece approximating linear function $L_i(x)$, with $n \in \{3, 5\}$
  - define the $L_i(x)$ function
  - divide the input space into $n^H$ subregions
- **for each** nonempty subregion generate a rule
  - define a linear equation that approximates the network output $\hat{y}_p$ for input sample $\mathbf{I}_p$ in this subregion as the *consequent* of the extracted rule:

$$\hat{y}_p = \sum_{i=1}^{H} v_i L_i(s_{ip}), \tag{3.2}$$

    where:

$$s_{ip} = \sum_{j=1}^{N} w_{ij} I_{jp}. \tag{3.3}$$

    $I_{jp}$ is the $j$-th feature of the input sample $\mathbf{I}_p$ and $v_i$ is the weight of the connection between the hidden neuron $i$ and the output neuron, while $w_{ij}$ is that of the connection between input neuron $j$ and hidden neuron $i$.
  - create the condition for this rule
- **optional:** simplify the rule conditions

Whereas it is rather straightforward to alter the REFANN method for the approximation of the hidden unit activation function, it seems much more difficult to apply the algorithm when there are more drastic changes to the architecture of the neural network, e.g. networks with several hidden layers.

**ITER.** ITER is a pedagogical algorithm usable for building predictive regression rules from a trained regression model, such as neural networks [92]. The main idea of the algorithm is to iteratively expand a number of hypercubes until they cover the entire input space. Each of these cubes can then be converted into a rule of the following format:

**Algorithm 4** The N2PFA algorithm.

- Split the data into three subsets: training, cross-validation, and test sets.
- Train a network with a sufficiently large number of hidden units to minimize the error function.
- Remove redundant hidden units.
- Remove irrelevant inputs.
- Report the accuracy of the network on the test data set.

$$\begin{aligned} &\text{if } Var_1 \in [Value_1^{Low}, Value_1^{High}] \\ &\text{and } Var_2 \in [Value_2^{Low}, Value_2^{High}] \\ &\text{and ... and } Var_M \in [Value_M^{Low}, Value_M^{High}] \\ &\text{then predict some Constant,} \end{aligned} \tag{3.4}$$

where $M$ is the dimension of the input space.

The algorithm starts with the creation of a user-defined number of random starting cubes. These cubes are infinitesimally small and therefore correspond to points in the input space. Afterwards, these initial cubes are gradually expanded until they cover the entire input space or until they can no longer be expanded. During each update are executed the steps reported in Algorithm 5.

Before the first iteration of the algorithm, ITER calculates the size of the surrounding hypercube, i.e. the cube that surrounds all of the training observations. When calculating the allowed update size, ITER takes this surrounding cube into consideration and never creates cubes that lie outside it. The surrounding cube is also used to retrieve default values for the $MinUpdate$ values. Unless the user specifies otherwise, the default is equal to a twentieth of the size of each dimension. For example, if the values of the training observations for some dimension lie within the interval [0,1] then $MinUpdate = 0.05$.

A drawback of the algorithm is non-exhaustivity. While ITER creates rules that are non-overlapping, it is not always able to cover the entire input space. In other words, the rules created by ITER are exclusive but not necessarily exhaustive. It can happen when none of the cubes can expand because each cube is blocked by another cube. To guarantee exhaustivity, it is necessary to add a number of cubes that cover the remaining gaps. The authors report that generally the number of cubes to add remains relatively small.

By specifying the number of starting cubes, the user automatically indicates the desired number of rules because ITER will never create extra cubes during execution. Only to make the resulting rule set exhaustive, the algorithm is allowed to construct additional cubes. There are several disadvantages, such as a strong

**Algorithm 5** The ITER algorithm.

1. For each hypercube $i = 1, ..., N$ and for each dimension $j = 1, ..., M$ calculate how far the cube can be expanded to both extremes of the dimension before it intersects with another cube, call these distances $LowerLimit_i^j$ and $UpperLimit_i^j$.
2. For each hypercube $i = 1, ..., N$ and for each dimension $j = 1, ..., M$ calculate the size of the update. The update equals $MinUpdate_j$, a user-specified constant, unless this size would result in overlapping cubes. If this is the case then the update is smaller such that the two blocks become adjacent.
   Mathematically: $LowerUpdate_i^j = \min\{LowerLimit_i^j, MinUpdate_j\}$ and $UpperUpdate_i^j = \min\{UpperLimit_i^j, MinUpdate_j\}$.
3. For each hypercube $i = 1, ..., N$ and for each dimension $j = 1, ..., M$ create two temporary cubes adjacent to the original cube along the opposite sides of dimension $j$ with a width of respectively $LowerUpdate_i^j$ and $UpperUpdate_i^j$. For each of both cubes, create a number of random points lying within the cube and calculate the mean prediction for these points according to the trained continuous regression model. Call the difference between each of both means and the mean prediction for the original cube respectively $LowerDiff_i^j$ and $UpperDiff_i^j$.
4. Find the global minimum over all cubes of these differences and combine the temporary cube for which the difference was minimal with its original cube. Update the mean prediction for this cube and remove all other temporary cubes.

dependence of the results on the number and location of the starting cubes, that make it worthwhile to give the algorithm the opportunity to create a new cube when the current update is deemed not 'good' enough. An update is considered to be 'good' when the global minimum of step 4 of Algorithm 5 is smaller than a user-specified threshold (see Algorithm 6). By setting the threshold to a very large value, all updates will be considered good and the results of the updated step 4 will be similar to the original.

It is worthwhile to notice that the ITER algorithm provides constant outputs for each rule, and not a linear combination of the input variables. ITER can also be adjusted for classification problems instead of regression ones.

### 3.3.3 Comparison between the described algorithms

The four different algorithms presented in this thesis enable knowledge extraction from artificial neural networks in various contexts.

**Algorithm 6** Modified fourth step of the ITER algorithm.

4. Find the global minimum over all cubes of these differences.
   If this global minimum is smaller than the threshold then combine the temporary cube for which the difference was minimal with its original cube. Update the mean prediction for this cube and remove all other temporary cubes.
   If this global minimum is larger than the threshold then create a new cube on the position of the temporary cube for which the difference was the global minimum. The size of each side of the hypercube equals $MinimumUpdate_j$ (smaller if this results in overlapping cubes).

If the task at hand is classification, only *Rule-extraction-as-learning* or TREPAN are eligible to extract knowledge from the network. They are both limited to categorical features due to their inability to handle real-valued features. The main differences between them are that *Rule-extraction-as-learning* is an eclectic method that produces a list of propositional rules, simple to implement but with small human readability when applied to problems with a large number of output classes or input features. Conversely, TREPAN is a pedagogical algorithm that extracts decision trees, with higher accuracy, fidelity and readability with respect to *Rule-extraction-as-learning*.

For networks addressing regression tasks, only REFANN or ITER are applicable in order to perform knowledge extraction. The main difference resides in the output of the produced explanator. With REFANN a regression interpretable model is constructed; the model is able to provide to the human a function that represents the approximated relationship between input and output data. On the other hand, ITER provides fixed output values for similar input samples, changing the regressive nature of the problem in a pseudo-classification, where all the samples belonging to a certain region of the feature space (which size is defined by the algorithm parameters) are associated with the same result (as it was a class label). In any case, ITER has to be preferred when the limits of REFANN make it impracticable to be chosen. One limit is represented by a large number of hidden neurons in the network architecture, since REFANN imposes to substitute the non-linear activation function of each neuron with an approximating linear function. Another limit is the presence of more than one hidden layer, because the algorithm is expected to be applied uniquely to single hidden layer networks.

# Chapter 4

# The LISA Pathfinder Mission

Galactic cosmic rays and solar energetic particles are nowadays monitored with an increasing fleet of spacecraft orbiting at different distances from the Sun and different inclinations about the ecliptic. In this thesis work the LISA Pathfinder mission is considered as case study.

## 4.1 The LISA Pathfinder mission

LISA Pathfinder was an ESA space mission aiming to demonstrate that the present technology for gravitational wave detection in space with interferometers is mature. In February 2011 the discovery of gravitational waves generated by stellar black holes collapse with the LIGO Earth interferometers opened a new window on these and other astrophysical sources of the gravitational Universe, studied up to the present time with electromagnetic waves only.

The exceptional results of the LISA Pathfinder mission, which demonstrated the possibility to place two free falling masses in space with a residual acceleration smaller than a millionth of billionth of the gravitational one, will allow to possibly anticipate the scientific mission LISA [93] to the beginning of the '30s, even though the nominal launch date is 2034. The main LISA experiment goal is to reveal the coalescence of supermassive black hole, which mass is up to $10^8$ times that of Sun (this latter equal to $2 \cdot 10^{30}$ kg).

A particle detector was placed on board LISA Pathfinder mission to monitor the flux of galactic cosmic-rays and solar particles with enough energy to traverse the spacecraft and charge the test masses (two platinum and gold cubes), generating spurious forces that, under particular conditions of the interplanetary medium, could simulate the gravitational wave passage. This particle detectors allowed to determine the integral flux of galactic cosmic-rays with a statistical uncertainty of 1% during one-hour intervals.

### 4.1.1 Characteristics and orbit

The LISA Pathfinder spacecraft was launched from the Kourou base in French Guyana on December 3rd, 2015 on board a Vega rocket. LISA Pathfinder reached its final orbit around the Lagrangian point L1 at 1.5 million km from Earth in the Earth-Sun direction on January 2016. July 18th, 2017 was the last day of data taking. The LISA Pathfinder orbit was inclined of about 45 degrees with respect to the ecliptic plane and the spacecraft spent about six months to complete it. Orbit minor and major axes were approximately 0.5 and 0.8 millions of km, respectively. The satellite rotated around its own axis with a six month period. The LISA Pathfinder orbit is reported in Figure 4.1. In the same figure appear also the orbits of the ACE and Wind satellites[1]. In particular, solar wind speed and interplanetary magnetic field intensity measurements gathered with Wind experiment will be used in this work for comparison with the galactic cosmic-ray variations.

The LISA Pathfinder mission was aimed to place two test masses in free fall with a residual acceleration of femto-g for the future experiments meant for gravitational wave detection in space. The gravitational wave detection in space with the interferometric method is based on the principle for which two distant free falling observers that are linked by laser beams, experience a time-varying difference in the beam frequency at the passage of a gravitational wave. Observers are represented by test masses. The LISA Pathfinder mission carried two test masses, namely two free-floating cubes with a side of 4.6 cm, of about 2 kg each, put at 38 cm of distance, composed by gold (70%) and platinum (30%) that played the role of interferometer mirrors. Despite the small distance that prevented from carrying out any scientific measurement, noise sources remained the same with respect to the final mission LISA. In Figure 4.2 a schema of the equipment on board LISA PAthfinder spacecraft is reported.

Protons and ions of galactic origin with energies larger than 100 MeV $n^{-1}$ penetrated and charged the test masses. This charging process induced spurious noise force on both test masses [94]. This process was properly studied before the mission launch on the basis of Monte Carlo simulations [95, 96, 97]. Periodic test masses discharging with ultraviolet light beams was carried out for noise control after the spacecraft launch [98].

### 4.1.2 The particle detector

A particle detector, shown in Figure 4.3, was placed on board LISA Pathfinder to monitor the overall incident galactic cosmic-ray and solar particle fluxes [99]. The particle detector was mounted behind the solar panels with its viewing axis

[1]Data from Wind and ACE experiments were obtained from the NASA-CDAWeb website (`https://cdaweb.gsfc.nasa.gov/index.html`).

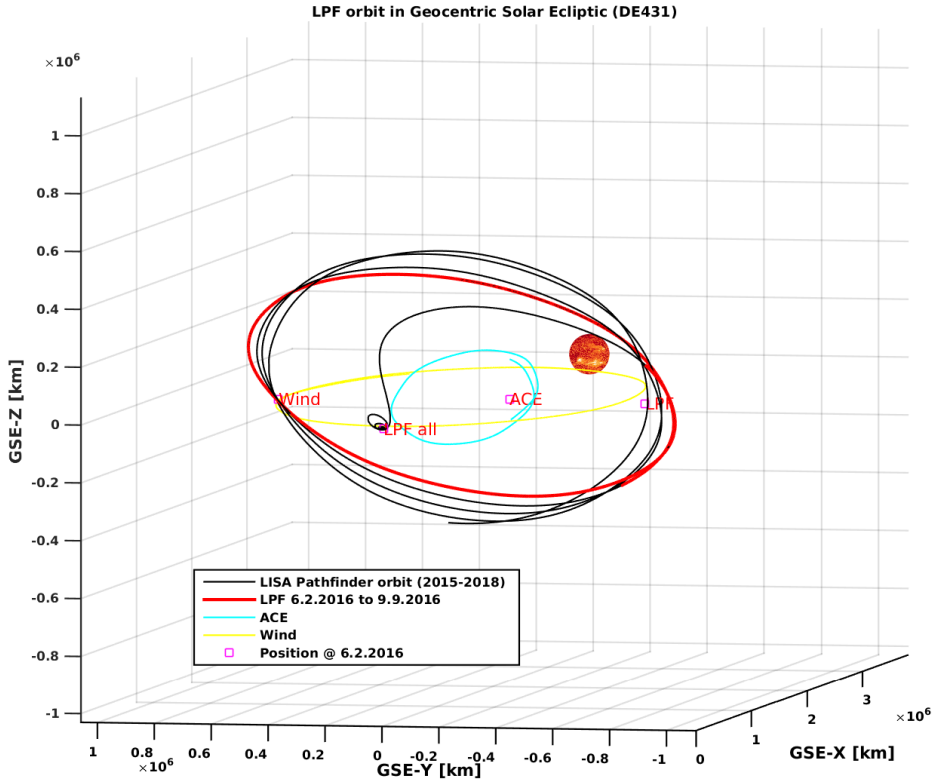



Figure 4.1: Orbit of the LISA Pathfinder satellite (black and red lines). The orbits of other interplanetary missions devoted to the monitoring of the interplanetary medium, ACE (cyan line) and Wind (yellow line), are also shown for comparison. Magenta squares point the effective spacecrafts position on February 6th, 2016, when the particle detectors where turned on. Courtesy of Andrea Cesarini.

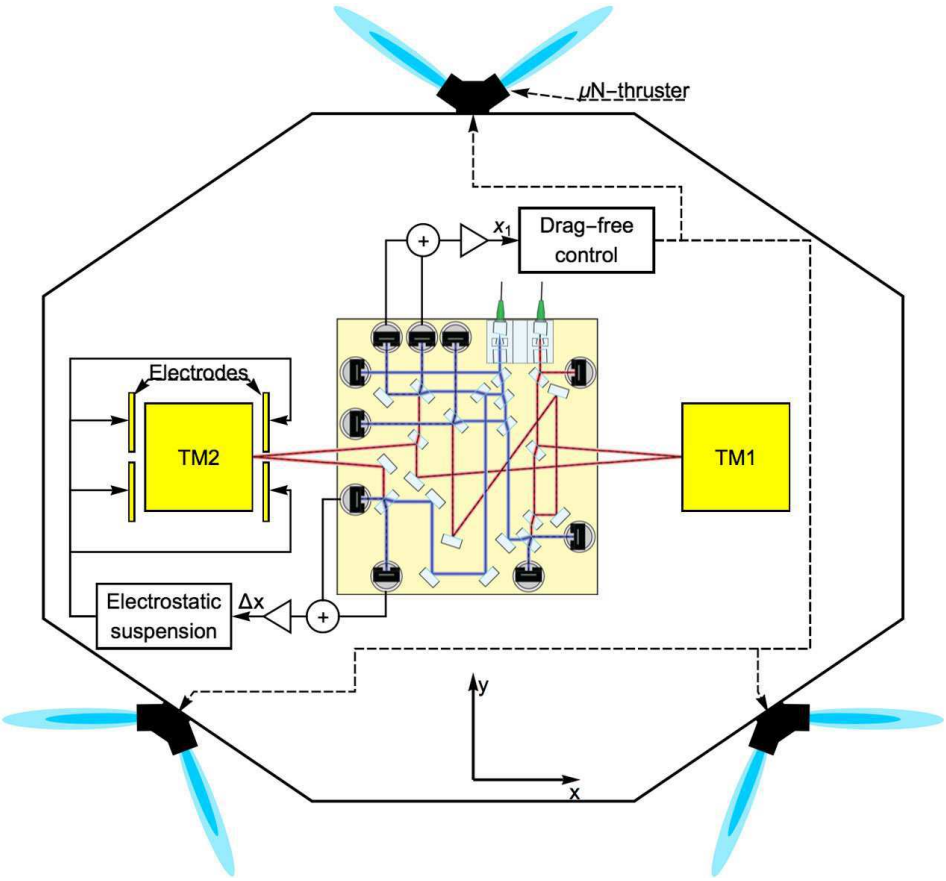



Figure 4.2: Schematic representation of the instrumentation on board the LISA Pathfinder spacecraft. The figure shows the two test masses (TM1 and TM2), the light beams and the optical bench used to measure $\Delta$x and $x_1$, namely the movement between the two masses and that of TM1 with respect to the spacecraft wall. The measure of $\Delta$x controls the electrostatic suspension of TM2, that applies electrical forces needed to reposition it by means of electrodes shown in the figure. Other electrodes surrounding the test masses are not indicated. The measure of $x_1$ guides the control cycle that uses the propellers to generate forces of the order of micronewton on the spacecraft. The image shows the x and y axes of the LISA Pathfinder reference system. The z axis is the normal to the figure. Courtesy of the LISA Pathfinder Collaboration.

along the Sun-Earth direction. The detector consisted of two $\sim$ 300 $\mu$m thick silicon wafers of 1.40 $\times$ 1.05 cm$^2$ area, separated by 2 cm and placed in a telescopic arrangement. For particles with energies $>$ 100 MeV n$^{-1}$ and an isotropic incidence on each silicon layer, the instrument geometrical factor was found to be energy independent and equal to 9 cm$^2$ sr. In coincidence mode (particles traversing both silicon wafers), the geometrical factor was about one tenth of this value. The silicon wafers were placed inside a shielding copper box of 6.4 mm thickness meant to stop particles with energies smaller than 70 MeV n$^{-1}$. The particle detector allowed for the counting of protons and helium nuclei traversing each silicon layer and for the measurement of ionisation energy losses of particles in coincidence mode. The maximum allowed detector counting rate was 6500 counts s$^{-1}$ on both silicon wafers, corresponding to an event integrated proton fluence of $10^8$ protons cm$^{-2}$ at energies $>$ 100 MeV. In coincidence mode up to 5000 energy deposits per second could be stored on the on board computer. No solar energetic particle events with associated proton fluences above a few tens of MeV n$^{-1}$ overcoming that of protons of galactic origin were observed during the LISA Pathfinder mission operation. As a result, a continuous monitoring of long and short-term variations of cosmic-rays of galactic origin was allowed.

The Nymmik model [100, 101] allows for the estimate of the rate of occurrence of solar energetic particle events with fluences larger than the saturation limit. For instance, the expected occurrence of solar energetic particle events with fluence $> 10^8$ protons cm$^{-2}$ at energies $>$ 30 MeV is less than 1 per year [102]. In case of particle detector saturation, the evolution of solar energetic particle events and the particle spatial distribution could be inferred from particle energy differential flux measurements carried out by other experiments and the use of models for particle propagation in the interplanetary medium [103]. In case of solar energetic particle event occurrences, observations gathered by other experiments should have been properly normalised to the counting rates observed on the LISA Pathfinder particle detector at the onset and during the final phases of each event.

## 4.2 Galactic cosmic-rays

### 4.2.1 Long-term variations of galactic cosmic-rays

The overall galactic cosmic-ray flux in the inner heliosphere is approximately of about 1000 particles m$^{-2}$ s$^{-1}$ [104]. Particles of galactic origin show an isotropic spatial distribution and consist approximately of 90% protons, 8% helium nuclei, 1% heavy nuclei and 1% electrons [105]. The galactic cosmic-ray energy spectrum appear modulated following the 11-year solar cycle and the 22-year global solar magnetic field polarity.

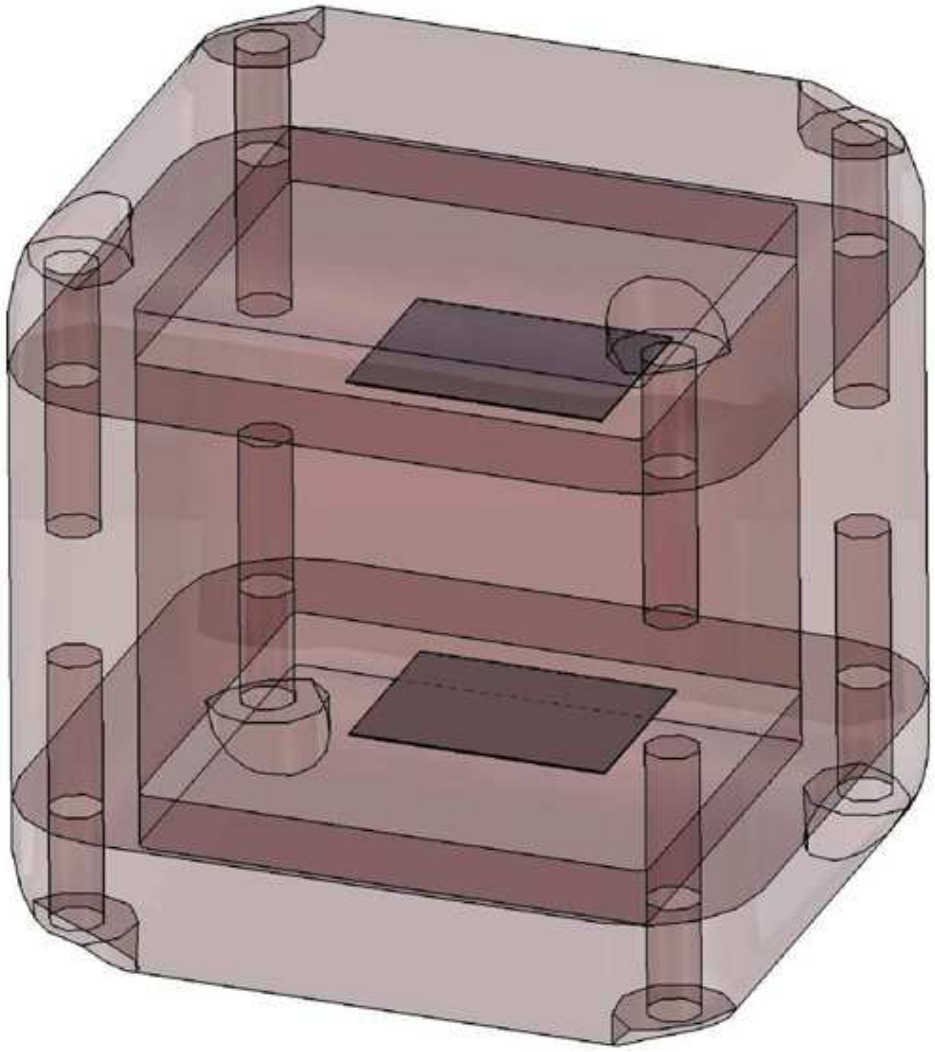


Figure 4.3: Schematic representation of the particle detectors on board LISA Pathfinder. Silicon layers, representing the sensitive part of the instrument, appear inside a copper shielding box.

It is worthwhile to recall that periods of positive polarity for the Sun are those during which the lines of force of the solar magnetic field exit from the North Pole; negative in the opposite case. The polarity change occurs during the solar maximum.

In [106] it was shown that during positive polarity epochs the energy spectra, $J(r, E, t)$, of cosmic-rays at a distance $r$ from the Sun and at a time $t$ are well represented by the symmetric model in the *force field approximation* by Gleeson and Axford (G&A) [107] assuming time-independent interstellar intensities $J(\infty, E + \Phi)$ and an energy loss parameter $\Phi$:

$$\frac{J(r, E, t)}{E^2 - E_0^2} = \frac{J(\infty, E + \Phi)}{(E + \Phi)^2 - E_0^2}, \tag{4.1}$$

where $E$ and $E_0$ represent the particle total energy and the rest mass, respectively. For $Z = 1$ particles with rigidity (particle momentum per unit charge) larger than 100 MV, the solar modulation is completely defined by the *solar modulation parameter*, $\phi$, that, at these energies, is equal to $\Phi$ [108].

The number of observed spots on the Sun photosphere is the most widely used proxy for the solar modulation. The photospheric sunspots are regions of low temperature and high magnetic field. The galactic cosmic-ray flux in the inner heliosphere is maximum (minimum) when the solar activity is minimum (maximum). The maximum proton flux variation between solar minimum and solar maximum is about one order of magnitude at 100 MeV $n^{-1}$ near Earth. The solar modulation does not present any evident effect on the galactic cosmic-ray energy flux above 10 GeV $n^{-1}$. The galactic cosmic-ray flux, consisting for 99% of the total of positive particles, results depressed at most by 40% at 100 MeV $n^{-1}$ during negative polarity periods. The negative polarity of the global solar magnetic field does not affect the galactic cosmic-ray flux above 4 GeV $n^{-1}$.

The sunspot number and solar modulation parameter values observed during the LISA Pathfinder mission are reported in Table 4.1. The integral flux of protons and helium nuclei increased by more than 20% during the mission, due to a decreasing solar activity. The solar modulation parameter is inferred from `http://cosmicrays.oulu.fi/phi/Phi_mon.txt` [109]. The galactic cosmic-ray single counts per sampling time of 15 s, averaged over each Bartels rotation ($GCR_{15s}$), were calculated during the LISA Pathfinder mission elapsed time. A linear correlation was found between the solar modulation parameter $\phi$ and $GCR_{15s}$:

$$GCR_{15s} = 0.23272\ \phi(\mathrm{MV}) + 230.73 \tag{4.2}$$

as it is shown in Figure 4.4. This observation suggests that the LISA Pathfinder particle detector did not present any detectable loss of efficiency during the first year of the mission lifetime.

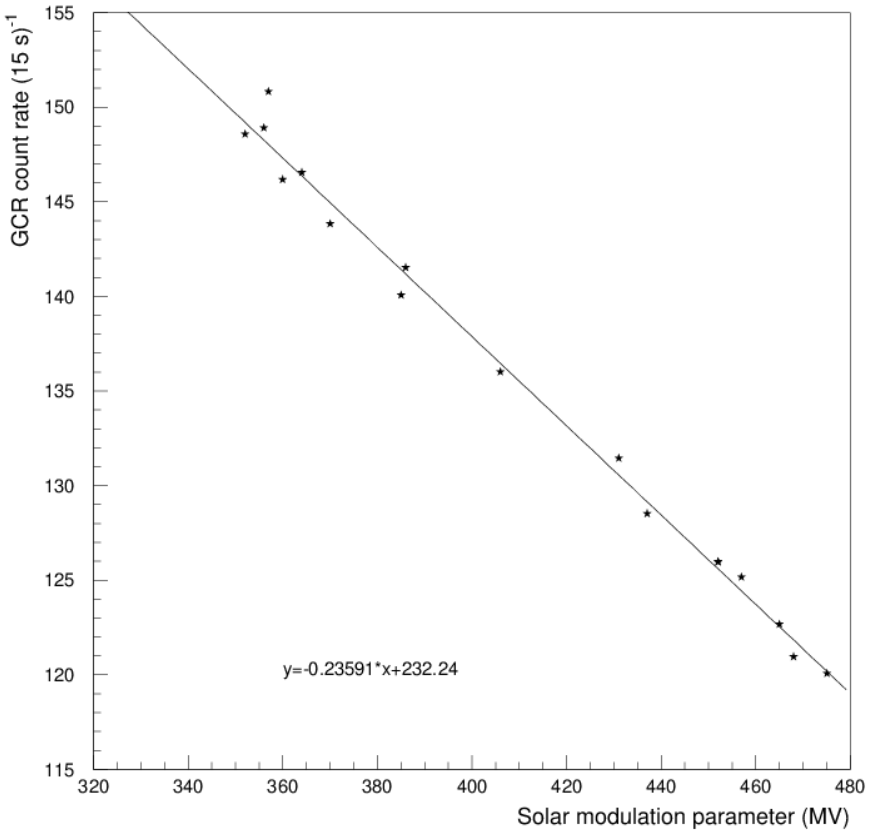



Figure 4.4: Solar modulation parameter and LISA Pathfinder particle detector galactic cosmic-ray single count rate in 15 s sampling time averaged over each Bartels rotation during the LISA Pathfinder mission. High (low) values of the solar modulation parameter correspond to mission beginning (end). The solar modulation parameter is reported in `http://cosmicrays.oulu.fi/phi/Phi_mon.txt` [109]. Courtesy of the LISA Pathfinder Collaboration.

Table 4.1: Observed sunspot number and solar modulation parameter ($\phi$) during the LISA Pathfinder mission. Data were reported in the website `http://cosmicrays.oulu.fi/phi/Phi_mon.txt` [109].

| | Sunspot number | $\phi$ MV |
|---|---|---|
| December 2015 | 58.0 | 561 |
| January 2016 | 57.0 | 500 |
| February 2016 | 56.4 | 468 |
| March 2016 | 54.1 | 475 |
| April 2016 | 37.9 | 468 |
| May 2016 | 51.5 | 464 |
| June 2016 | 20.5 | 447 |
| July 2016 | 32.4 | 464 |
| August 2016 | 50.2 | 438 |
| September 2016 | 44.6 | 436 |
| October 2016 | 33.4 | 407 |
| November 2016 | 21.4 | 385 |
| December 2016 | 18.5 | 386 |
| January 2017 | 26.1 | 366 |
| February 2017 | 26.4 | 357 |
| March 2017 | 17.7 | 348 |
| April 2017 | 32.6 | 367 |
| May 2017 | 18.8 | 359 |
| June 2017 | 19.4 | 350 |
| Mission end | | 383 |

By applying the G&A model to the interstellar proton and helium nucleus energy spectra inferred from a series of balloon flights of the BESS and BESS-POLAR I and II experiments [110, 111, 112], the proton and helium nucleus energy differential fluxes at the beginning (December 2015 - January 2016; $\phi = 550$ MV) and at the nominal end of the LISA Pathfinder mission were obtained and reported in Figure 4.5.

In order to test the reliability of the approach described above, the proton energy spectrum predictions for June 2015 were compared to the preliminary measurements carried out by the PAMELA experiment (private communication) during the same period. A very good agreement was found. For the comparison it was chosen the June 2015 period since the published PAMELA data until 2013 were gathered during a negative polarity period of the solar magnetic field. Predictions for the proton flux during the Bartels rotation 2496 (from July 7th, 2016 through August 16th, 2016) were compared to the AMS-02 experiment analogous data published in 2017 and an agreement of 10% was observed for the integral fluxes.

The energy spectra interpolation function is reported below [105]:

$$F(E) = A\ (E + b)^{-\alpha}\ E^{\beta} \qquad \text{particles } (\text{m}^2\ \text{sr s GeV n}^{-1})^{-1}, \tag{4.3}$$

where E is the particle kinetic energy per nucleon. The parameters $A$, $b$, $\alpha$ and $\beta$ for the LISA Pathfinder mission beginning and end appear in Table 4.2. About these predictions of the galactic cosmic-ray energy spectrum at steady state (without considering galactic cosmic-ray short-term variations), it is worthwhile to recall that cosmic-rays are also modulated on much lower time-scales during the passage of magnetic structures of solar or interplanetary origin, that sometimes produce geomagnetic disturbances.

### 4.2.2 Short-term variations of galactic cosmic-rays

The Sun is a sphere of plasma and gas rotating differentially depending on the heliolatitude. Equatorial and near-equatorial regions rotate with a period of about 25-26 days (sidereal rotation period), while near the poles the period is about 36 days. The solar rotation periodicity appears as 27-28 days (27.28 days on average) to an observer on Earth (synodic rotation period) due to the orbital motion of our planet and to the solar wind characteristics observed from Earth, representative of the conditions of the Sun near-equatorial region in the heliosphere.

All galactic cosmic-ray flux variations characterised by a duration shorter than the solar rotation period and associated with the passage of magnetic structures of solar or interplanetary origin are called short-term variations. Nominal quasi-periodicities of 27, 13.5 and 9 days, related to the Sun rotation period and higher

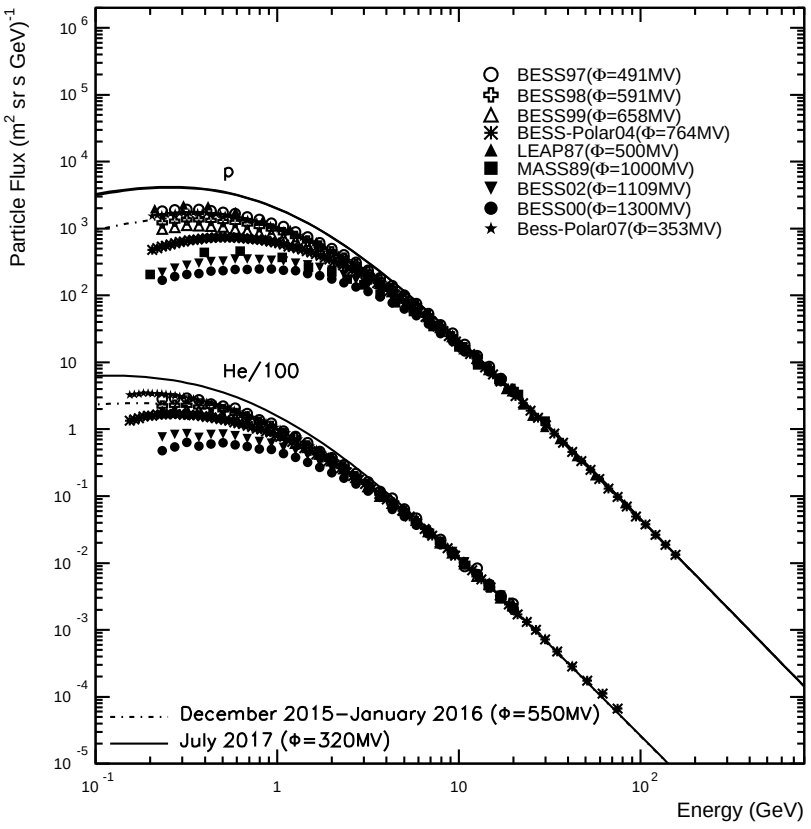



Figure 4.5: Galactic cosmic-ray proton and helium energy spectra measurements (data points; [110]). Estimated energy spectra at the beginning of the LISA Pathfinder mission (December 2015 - January 2016) and predictions at the end of the same (July 2017) are also indicated as dot-dashed and continuous lines, respectively. The helium flux appears properly scaled in order not to superpose lines. Courtesy of the LISA Pathfinder Collaboration.

Table 4.2: Parameterisations of proton and helium energy spectra at the beginning and the end of the LISA Pathfinder mission.

| | $A$ | $b$ | $\alpha$ | $\beta$ |
|---|---|---|---|---|
| p (Dec. 2015 - Jan. 2016) | 18000 | 1.19 | 3.66 | 0.87 |
| p (May 2017 - minimum) | 18000 | 1.09 | 3.66 | 0.87 |
| p (May 2017 - average) | 18000 | 1.03 | 3.66 | 0.87 |
| p (May 2017 - maximum) | 18000 | 0.97 | 3.66 | 0.87 |
| He (Dec. 2015 - Jan. 2016) | 850 | 0.96 | 3.23 | 0.48 |
| He (May 2017 - minimum) | 850 | 0.90 | 3.23 | 0.48 |
| He (May 2017 - average) | 850 | 0.84 | 3.23 | 0.48 |
| He (May 2017 - maximum) | 850 | 0.78 | 3.23 | 0.48 |

harmonics, are observed in the cosmic-ray flux, in the solar wind plasma and magnetic field, and in the geomagnetic activity indices [113, 114]. Oscillations in the cosmic-ray flux were formerly studied in space above a few tens of MeV with the Helios 1, Helios 2 and IMP-8 experiments [115]. These observations indicated that the effects of corotating interaction regions, generated when high-speed solar wind streams emanating from coronal holes overtake the leading slow solar wind associated with the closed streamer belt region, are at the origin of $\sim$ 9-day galactic cosmic-ray flux modulations [116]. Temmer, Vršnak and Veronig [117] found that a $\sim$ 9-day periodicity is also shown by the coronal holes area. A correlation of the galactic cosmic-ray short-term variations with the $BV$ product of the interplanetary magnetic field intensity ($B$) and the solar wind speed ($V$) was investigated by Sabbah [118]. This correlation takes into account both cosmic-ray diffusion from interplanetary magnetic field and convection in the solar wind. From the point of view of geomagnetic indices, a good correlation of $A_p$ and $K_p$ with both $BV$ and $BV^2$ was found by Sabbah [119]. $A_p$ and $K_p$ are magnetic indices defined as follows: $A_p$ index is related to $K_p$ index, that in turn represents the values global mean of the geomagnetic activity produced by solar particle radiation and recorded in thirteen reference stations. The quasi 9-day periodicity observed in the thermospheric energy budget was also associated with the recurrence of high-speed solar wind streams [120].

Clues on the energy-dependence of 27-day galactic cosmic-ray flux variations are reported in [97]. In the past, the majority of work on this topic was carried out with observations gathered on Earth with neutron monitors [121]. These devices have the merit to have provided a continuous monitoring of the overall galactic cosmic-ray flux trend in the last sixty years. However, the use of models is manda-

tory to infer from ground measurements the galactic cosmic-ray spectra at the top of the atmosphere [122]. Interesting attempts to investigate the energy-dependence of short-term depressions of cosmic-ray fluxes through direct measurements with magnetic spectrometers, down to low energies, were carried out by the balloon-borne experiment BESS-POLAR 1 [111] and the satellite experiment PAMELA [123]. BESS-POLAR I flew from Williams Field near Mc Murdo Station from December 13th through December 21st, 2004. At the beginning of the flight, this balloon-borne experiment observed a recovering proton flux from a previous decrease. The recovery intensity appeared to be of 8-9% below 0.86 GeV and of 3% above 6 GeV. Authors claimed that this occurrence was due to the transit of a corotating interaction region interface or a magnetic cloud or a combination of the two. This experiment detected a new galactic cosmic-ray proton flux depression after the passage of a high-speed stream on December 17th. PAMELA carried out the first measurement of proton and helium nucleus differential fluxes in space during a Forbush decrease on December 14th, 2006 (16.50 UT - 22.35 UT) after two solar energetic particle events dated December 13th and December 14th, 2006. Unfortunately, magnetic spectrometer space-borne experiments have small geometrical factors and differential flux data must be integrated over periods longer than the typical one-hour data binning required to sample galactic cosmic-ray short-term variations. Moreover, observations gathered with balloon-borne experiments may be affected by the geomagnetic field playing some role in modifying the galactic cosmic-ray fluxes at low energies. Data gathered with LISA Pathfinder and other experiments in space revealed the energy dependence of the galactic cosmic-ray short-term variations as shown in [].

### 4.2.3 Forbush decreases

Forbush decreases, discovered by Forbush in 1937 [], are the most intense non-recurrent variations. Forbush decreases are characterised by a drop of the galactic cosmic-ray intensity observed during typical periods of one day, while the recovery phases last approximately two or three days.

An example of Forbush decrease is shown in Figure 4.6 [124]. Non-recurrent Forbush decreases are associated with the passage of interplanetary counterparts of coronal mass ejections. In principle, the evolution of the galactic cosmic-ray drop is modulated by, before the passage of shock and ejecta, then by the magnetic cloud. In this case the Forbush decrease is called *classical*. Conversely, in the case the interplanetary counterparts of the coronal mass ejection is not accompanied by shocks or magnetic clouds, the Forbush decrease presents a different dynamics [125]. Non-recurrent Forbush decreases generate galactic cosmic-ray flux depressions of about 5-10% at 10 GV of rigidity.

Recurrent Forbush decreases with intensities $< 2\%$ at 10 GV of rigidity are

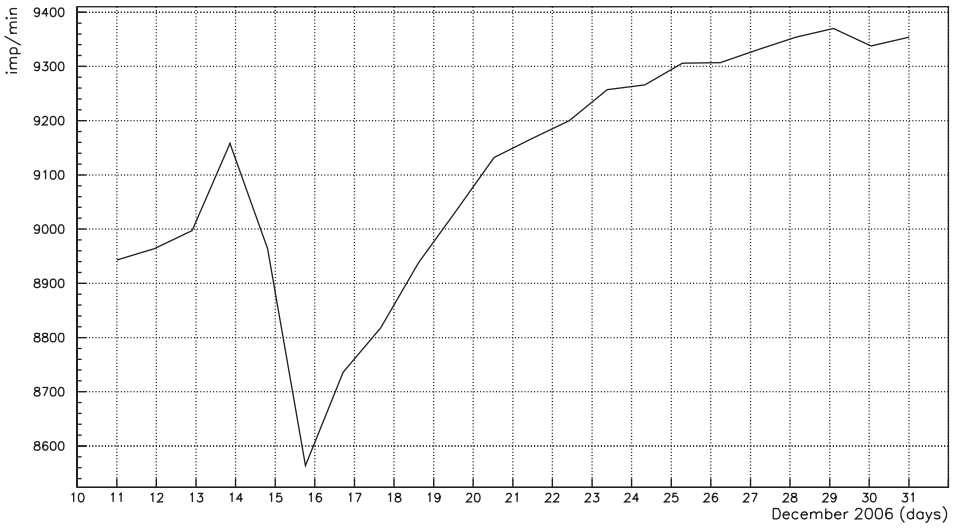



Figure 4.6: Moscow neutron monitor counting rate in December 2006. A non-recurrent Forbush decrease was observed on December 15th. The counting rate of the neutron monitor (y axis) is expressed in impulses per minute.

associated with corotating interaction regions in the interplanetary medium. This last observation is carried out in space with proton differential flux measurements (as on board LISA Pathfinder) and on Earth with neutron monitors. In general, it is observed that the short-term galactic cosmic-ray variations in space show a more marked energy dependence at low energies not accessible to neutron monitors.

During the LISA Pathfinder mission have been observed forty-five recurrent Forbush decreases and only three non-recurrent Forbush decreases.

## 4.3 Cosmic-ray flux short-term variations observed with LISA Pathfinder

The LISA Pathfinder particle detector allowed to study galactic cosmic-ray short-term flux variations during Bartels rotations 2490-2509 (from February 18th, 2016 through July 18th, 2017). A Bartels rotation lasts 27 days and is defined as a complete apparent Sun rotation viewed from Earth. The first day of the rotation 1 was arbitrarily fixed on February 8th, 1832.

In order to focus on recurrent periodicities consistent with the Sun rotation period and higher harmonics, galactic cosmic-ray percent variations were compared to interplanetary magnetic field and solar wind plasma parameters for each Bartels rotation and to neutron monitor measurements. Galactic cosmic-ray counting rates of neutron monitors vary proportionally to the cosmic-ray flux at energies larger

Table 4.3: Neutron monitor station characteristics.

| Station | Vertical cut-off rigidity GV | Effective energy GeV |
|---|---|---|
| Thule | 0.3 | 5.5 |
| Terre Adelie | 0.0 | 5.5 |
| Mc Murdo | 0.3 | 5.5 |
| Oulu | 0.8 | 6 |
| Rome | 6.3 | 15 |
| Mexico | 8.2 | 29 |

than the *effective energy* ranging between 10-11 GeV and more than 20 GeV for near-polar and equatorial stations, respectively [126]. This energies are set by the vertical geomagnetic cut-off and the shielding action of the atmosphere. In other words, neutron monitors allow for a direct measurement of the galactic cosmic-ray flux at energies larger than the effective energy of each station. Vertical cut-off rigidities and effective energies for all neutron monitor stations considered in this work are reported in Table 4.3.

The effects of systematic errors of galactic cosmic-ray observations on board LISA Pathfinder, due to possible fluctuations of the particle detector efficiency over the mission lifetime and solar modulation intensity change, were reduced by considering the percent variations of the cosmic-ray flux with respect to the average value during each Bartels rotation. A similar approach was considered in [127] for the ACE experiment. As an example, in Figure 4.7 the cosmic-ray percent variation (first panel) during the Bartels rotation 2491 (from March 4th through March 31st, 2016) are compared to the solar wind plasma velocity gathered in L1 by the ACE spacecraft (second panel), to the interplanetary magnetic field radial component (third panel) and intensity (fourth panel). In the third panel the heliospheric current sheet crossing is also indicated. The fifth panel shows those periods of time during which the solar wind velocity remains above 400 km $s^{-1}$ and the interplanetary magnetic field intensity stays above 10 nT. It is possible to notice that these values of the interplanetary parameters are correlated with the cosmic-ray depressions.

In Fig. 4.8 the cosmic-ray percent variations are compared to near-polar neutron monitor contemporary observations hourly binned as the LISA Pathfinder data. From the beginning of the data taking phase until April 2016, LISA Pathfinder remained below the ecliptic plane, crossing it only once every three months. Due to the high effective energy of the neutron monitors, there is no evidence of a

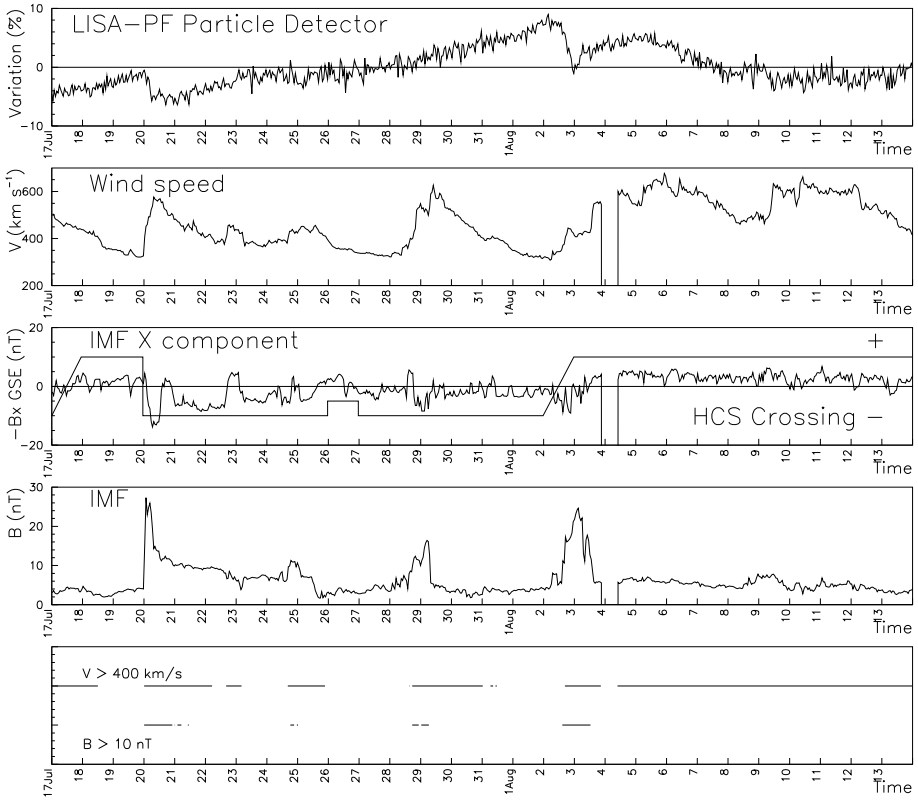



Figure 4.7: Comparison of LISA Pathfinder particle detector counting rate percent variations (first panel) with ACE measurements of the solar wind speed (second panel) and interplanetary magnetic field radial component (third panel) and intensity (fourth panel), during the Bartels rotation 2496 (July 17th, 2016 - August 13th, 2016). In the fifth panel appear indicated those periods of time during which the solar wind speed and the magnetic field intensity remain above 400 km $s^{-1}$ and 10 nT, respectively. Courtesy of the LISA Pathfinder Collaboration.

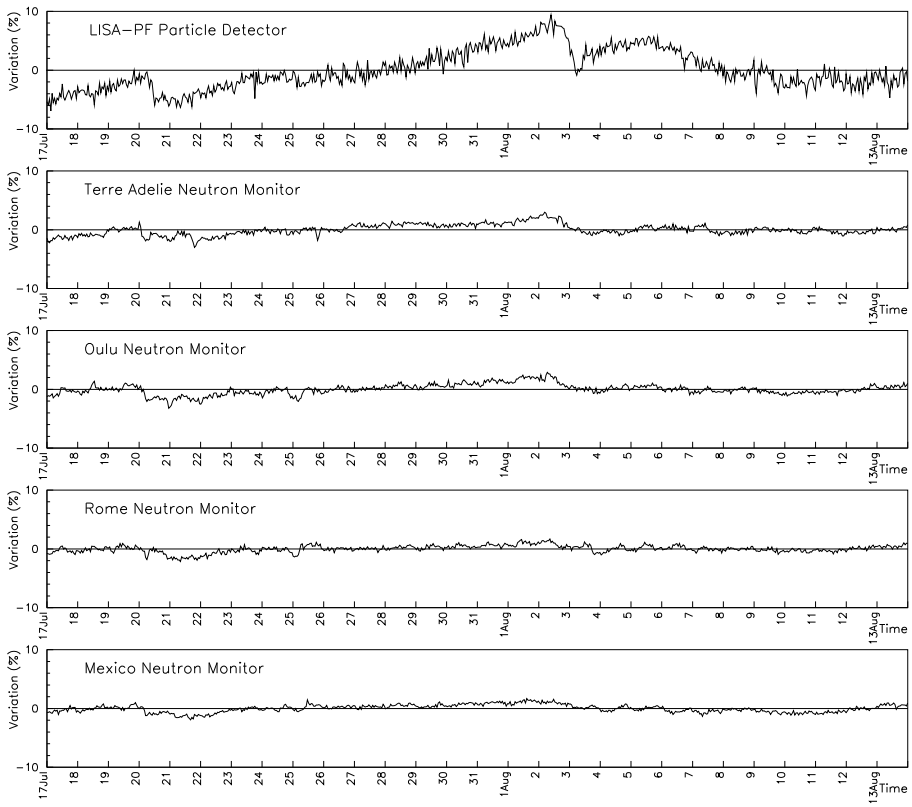



Figure 4.8: Comparison of LISA Pathfinder particle detector counting rate percent variations (first panel) with contemporary, analogous measurements of neutron monitors placed at different vertical geomagnetic cutoffs (other panels) during the Bartels rotation 2496 (July 17th, 2016 - August 13th, 2016). Courtesy of the LISA Pathfinder Collaboration.

better agreement of the trend of the LISA Pathfinder measurements with those of either North or South neutron monitor stations. It can be observed that the measurements carried out on board LISA Pathfinder, that include low energy particles, are more intense in comparison to those of high-latitude neutron monitors (that is at high geomagnetic cut-off).

From the point of view of the time profiles of individual depressions, those presenting similar durations for decrease and recovery phases are called *symmetric*. The symmetric variations are V or U shaped. All the other depressions are called *asymmetric* [128]. The most intense asymmetric depressions are the non-recurrent Forbush decreases. All recurrent variations under study appear asymmetric except six that show a U-shape (five of six) or a V-shape (one of six). The period during which the particle detector counting rate remains at minimum values between

Table 4.4: Average characteristics of galactic cosmic-ray recurrent variations observed with LISA Pathfinder.

| | Days | % |
|---|---|---|
| Decrease | 2.8 ± 2.0 | |
| Plateau | 1.3 ± 1.2 | |
| Recovery | 5.1 ± 3.8 | |
| Total duration | 9.2 ± 5.0 | |
| Intensity | | 5.1 ± 2.5 |

decrease and recovery phases is called *plateau*. A plateau is observed during both U-shaped symmetric and asymmetric depressions. Average duration of decrease, plateau and recovery periods for the $\sim$ 9-day depressions observed with LISA Pathfinder are reported in Table 4.4.

It can be noticed that the galactic cosmic-ray flux appears modulated when the interplanetary magnetic field intensity is larger than 10 nT and/or the solar wind speed remains above the threshold velocity of 400 km $s^{-1}$ (bottom panel of Figure 4.7). This scenario basically corresponds to the transit of fast-slow wind interaction regions. In the majority of cases the cosmic-ray flux begins to recover when the solar wind speed drops below 400 km $s^{-1}$. The correlation of galactic cosmic-ray short-term variations with the $BV$ parameter, carried out in several works, privileges the role of the magnetic field trend and penalises that of the solar wind speed due to the mutual variations of these two parameters, being much more relevant those of the magnetic field. The recurrent galactic cosmic-ray depressions observed with LISA Pathfinder, for instance, are associated with solar wind speed changes smaller than 30%, while the magnetic field is observed to increase up to a factor of 5. Therefore, while the study of galactic cosmic-ray short-term depressions associated with a drift effect ($BV$) is effective, it is inferred that a separate analysis of the role played by $B$ and $V$ increases may help in better understanding the dynamics of individual depressions. This approach allows also to infer the role of interplanetary magnetic structures in influencing galactic cosmic-ray variations.

# Chapter 5

# Predicting LISA Pathfinder Data

This thesis work focuses on the development of a model aiming to predict the galactic cosmic-ray flux short-term variations observed on board the LISA Pathfinder mission. The prediction has to be comprehensible for human users, i.e. the output of the model is a prediction with its associated explanation. In this chapter the workflow leading to the implementation of an explainable model based on a neural network is reported. The explainable model predictions have an average error smaller than the statistical uncertainty of the LISA Pathfinder mission and provide a linear relationship between the input and output variables.

The output of the developed system is the prediction of the galactic cosmic-ray flux variations at a certain instant. Being these target variations mainly modulated by the interplanetary magnetic field intensity and the solar wind speed, the observation of these two variables are chosen as input data. In addition, a single value of the cosmic-ray flux variation is taken in input for flux variation normalisation. All these input variables are combined into a linear function provided as output by the implemented explainable model.

## 5.1 Model design

The work presented here is based on a neural network model. A knowledge extraction algorithm is applied to the neural network in order to obtain a new explainable model. The workflow allowing for the realisation of these models is reported in Figure 5.1.

As shown in the left portion of the figure, the process consists of four macro-steps. The first represents the creation of a data set used for the neural network model training, tuning and testing. The second and third macro-steps indicate the neural network implementation and optimisation, to find the model hyper-parameters and data set options that minimise the prediction error. Finally, a

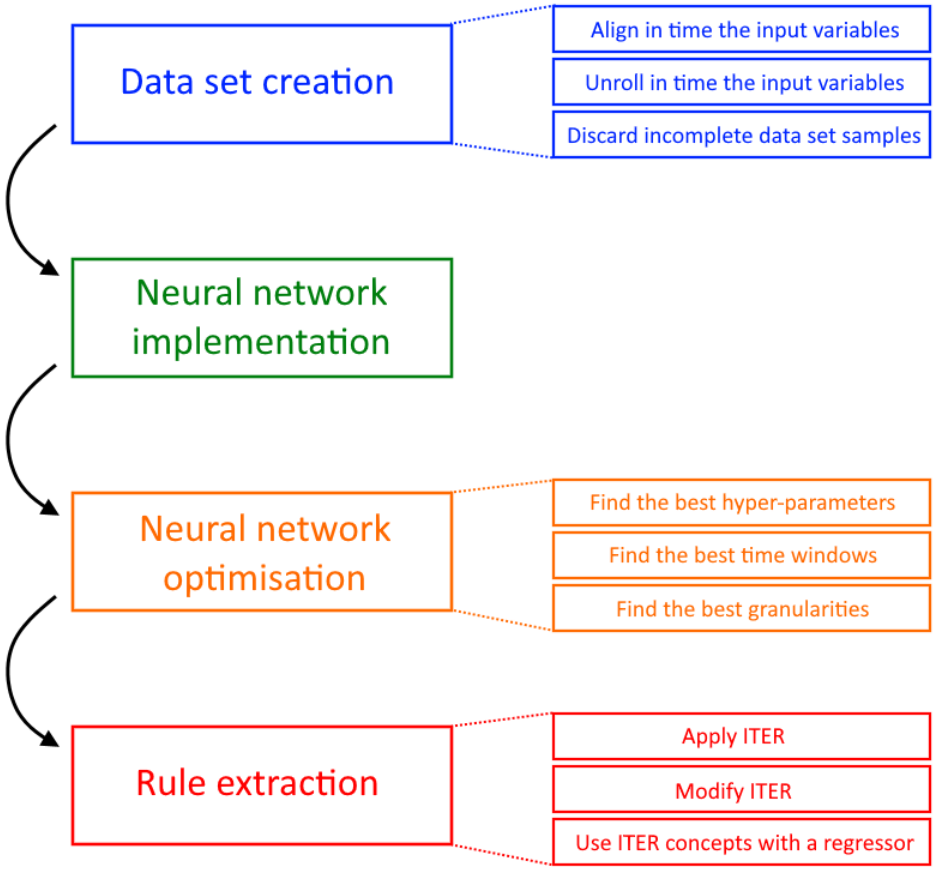



Figure 5.1: Schematic representation of the workflow allowing for the realisation of the models described in this chapter. On the left are reported the macro-steps. The single sub-steps are reported on the right.

rule extraction algorithm is applied to the optimised neural network.

These macro-steps are split into sub-steps, shown in the right portion of the figure. The data set creation consists of three phases: firstly the input variables are time aligned, then each variable (in the form of a time series) is unrolled to constitute the data set sample. Finally, incomplete samples due to missing measurements are discarded. The output data set is then used to compose the training, validation and test sets.

The optimisation of the neural network model consists of three phases: the research of the best hyper-parameter values and network architecture that minimise the prediction error; the study of time windows of different sizes for observing the input variables and finally the use of different granularities for these variable observations. The time windows refer to how many days of past input variable observations are necessary for the prediction; the granularity is the number of

observations that belong to each time window. The aforementioned second and third steps are finalised to obtain an optimised model with the same prediction error of the model selected during the first step but using a smaller data set (by diminishing the size of the time windows and increasing the granularity). A small data set allows to simplify the knowledge extraction process.

To perform the knowledge extraction from the neural network model three steps are performed. Firstly, the ITER algorithm is applied to the neural network. In case the ITER algorithm application would not return a positive result because of the complexity of the network, an alteration of the original ITER algorithm would be implemented and applied to the same neural network. Again, in the case this would not be sufficient, a new model using a linear regressor and with a logic resuming the main ITER concepts is finally applied to the neural network in order to extract linear functions that explain the relationship between the input solar wind speed and interplanetary magnetic field intensity with the output cosmic-ray flux variation.

The aforementioned macro-steps are executed with a modular approach with three different programs described in Section 5.2: the first creates the data set. The output data set represents the input file for the second program, developed to create the predicting neural network and finally, the third program is used for knowledge extraction from the neural network.

## 5.2 Implementation

### 5.2.1 Data set creation

Since the neural network has to predict the galactic cosmic-ray flux variations with a good accuracy, the LISA Pathfinder data have been hourly binned, in order to limit the statistical error to 1% on each bin. The data were gathered between February 18th, 2016 and July 3rd, 2017. Analogously, during the same period of time, solar wind speed and interplanetary magnetic field intensity observations gathered with Wind have been also hourly binned. The three data time series were then time aligned, associating the corresponding timestamp to each set of observations. The interplanetary magnetic field intensity is expressed in nT and the solar wind speed in km $s^{-1}$. For an accurate prediction of the output data, a time window must be set to study the preceding trend of the mentioned variables with respect to present time. The optimum size of this window is set on the basis of the output of the second program, as described below. Being conscious that both solar wind and interplanetary magnetic field modulations have an average duration of a few days (in particular, four/five for high-speed stream passage and two/three for interplanetary magnetic field increases associated to corotating interaction regions

and the transit of interplanetary counterparts of coronal mass ejections), in order to verify that the program will find an analogous solution, a time window of twelve days is initially set. During the training phase of the neural network model it is possible to dynamically select time sub-windows of these twelve days, for example keeping the first four past days and removing the remaining. For each time instant set, the twelve-day data window of the input variables is considered, along with the input solar wind speed and interplanetary magnetic field intensity at the same instant. The cosmic-ray variation nine days before the selected time instant is also considered in input. The nine days elapsed time is set on the basis of the observed average duration of the galactic cosmic-ray flux recurrent variations. The cosmic-ray variation at the time instant constitutes the output variable. The total number of the considered observations is $(12 \cdot 24 + 1) \cdot 2 + 1 = 579$. To limit the role of the statistical uncertainty of the cosmic-ray variation input data, instead of the value of the variation observed nine days before of the selected time instant, data were averaged over four hours; two time bins before selected time and two after. In Figure 5.2 an example of a data set sample during the Bartels rotation 2505, from March 17th, 2017 through April 13th, 2017, is reported. The plot represents in the top panel the galactic cosmic-ray flux percent variation, in the middle one the solar wind speed and in the bottom panel the interplanetary magnetic field intensity. The figure highlights the input features and the output value for the sample corresponding to the April 4th, 2017 at 22:00 UT. In the top panel the output value of galactic cosmic-ray flux variation (red dot on the right) and the input value used for flux normalisation (blue dot on the left) are highlighted. In the middle and bottom panels blue dots indicate the input features representing the solar wind speed and the magnetic field intensity, respectively (namely the observations on April 4th, 2017 at 22:00 UT plus one observation per hour during the preceding twelve days).

All samples for which some features are missing due to unavailability of space mission data are discarded. The data set is split into two separate parts, one that is used for the training of the neural network and one that is adopted for the final testing. The test set is never used during the training or the hyper-parameter tuning and it consists of the 25% of the original data set. This 25% of data is randomly extracted from the original data set. The training set and test set are used as input data for the other programs developed for this analysis. The total number of samples is 11 319.

### 5.2.2 Neural network implementation and optimisation

This section describes the neural network created to predict the galactic cosmic-ray flux short-term variations observed on board LISA Pathfinder. In this program the best hyper-parameter values for the network training are chosen and the final

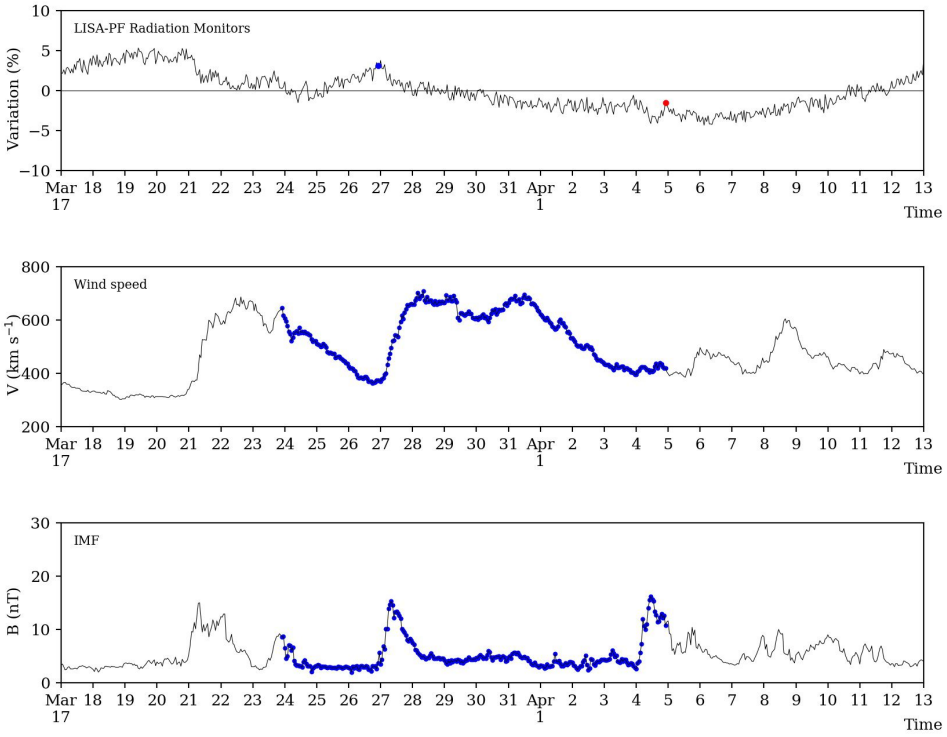



Figure 5.2: Galactic cosmic-ray flux percent variations (top panel), solar wind speed (middle panel) and interplanetary magnetic field intensity (bottom panel) measured during the Bartels rotation 2505. With reference to the data set sample of April 4th, 2017 at 22:00 UT, composed of cosmic-ray flux variations and solar wind speed and interplanetary magnetic field intensity observations, all the input features are highlighted with blue dots, while the output value is reported in the top panel with a red dot. The predicted galactic cosmic-ray percent variation dated April 4th, 2017 at 22:00 UT (red dot) was obtained on the basis of the cosmic-ray flux variation value observed nine days before (blue dot in the top panel) and hourly observations of the solar wind speed and interplanetary magnetic field intensity (blue dots in the middle and bottom panels, respectively) gathered during the preceding twelve days.

prediction accuracy with respect to the test set data is evaluated. The program also allows for the graphical display of the results, in order to give the user the opportunity to check at a glance the network performance.

**Hyper-parameter tuning.** The training of a neural network requires the choice of the best values for each hyper-parameter. This goal cannot be easily reached, since the combination of all possible values of each hyper-parameter makes an exhaustive research in the hyper-parameter value space impracticable. The approach adopted in this work to find acceptable values of the hyper-parameter is the following: some hyper-parameters are fixed, such as the activation function of the hidden and output layers; the others have been explored with a grid search to highlight possible promising regions to explore more finely.

For the very preliminary estimation of these hyper-parameters, the batch size is fixed and set equal to 1000 samples. Several tests were carried out to estimate the impact of changing the batch size. No sensitive variations were observed for batch sizes varying from hundreds to a few thousands. The only difference consists in the number of epochs required for the convergence.

After setting the batch size, the maximum number of epochs is also set to 200, in order to allow for possible slow trainings. An early stopping method is used to monitor the trend of the prediction error on the validation set during the training. Early stopping patience value and threshold for possible improvements between consecutive epochs are 5 and 0.05, respectively. The statistical uncertainty of the LISA Pathfinder cosmic-ray data is of 1%. Therefore, the mean squared error (MSE) is preferred with respect to the mean absolute error (MAE) to estimate the predictive performance of the neural network (since the MSE assigns more weight to errors larger than 1% and reduces that of errors smaller than 1%). The MSE is calculated on the validation set for triggering the early stopping. All hidden neurons have an activation function of type hyperbolic tangent, while the identity function is used for the output neuron (see Section 2.2.2).

The test set is never used during the hyper-parameter tuning or the network training. The remaining samples are divided into a fixed validation set (10%), used for monitoring the early stopping, and an effective training set. The accuracy measurements for the best hyper-parameter values are done by applying the $k$-fold cross validation technique to the training set, with $k = 10$ (see Figure 2.1). All tests are done with this same sets for better comparing the results. At the end of the grid search on the hyper-parameter space and the selection of the best hyper-parameters, the neural network is re-trained with these best values and the whole training set. Only the definitive network is tested on the test set for checking its prediction accuracy.

With the constraints defined above the network architecture and the optimal

learning rate are chosen. Different ways to minimise the data set to improve the prediction accuracy or remove redundancy are explored.

**Best network architecture and learning rate selection.** The first issue to handle is to select the optimum network architecture, i.e. to set the number of hidden layers and the number of hidden units. Even though it has been demonstrated that a neural network with a single hidden layer using a sigmoid activation function is a universal approximator, it is a common practice to try adding a hidden layer to attempt to improve the network accuracy. For this reason both architectures with one and two hidden layers are tested. A good practice suggests to chose the number of hidden neurons by adding them incrementally, starting from a little number and adding units iteratively until the prediction error on the validation set stops to decrease or an over-fitting condition (see Section 2.1.3) is identified. For the four-layer architecture it has been decided to limit the second hidden layer to only half of the first hidden layer neurons.

It is worthwhile to recall that the optimal hyper-parameter values are intertwined, so the learning rate critically depends on the chosen architecture and its optimum value is not necessarily the same for all the different possible architectures.

In Table 5.1 the mean squared error calculated on the validation set during the training of a neural network with a single hidden layer is reported. The number of hidden units varies from 75 to 900. Larger values are useless, because the prediction accuracy on the validation set decreases very slowly with respect to the network complexity. The learning rate ranges between 0.1 and 0.00016. All possible combinations of all the parameter values are tested, selecting intersection points in a grid that covers the entire space described. Twelve different values are tested for the number of neurons and five for the learning rate, resulting in sixty different combinations of hyper-parameter values.

The standard deviation reported in this and other tables has been corrected. The same results of Table 5.1 are reported graphically in Figure 5.3. The results highlight that large learning rates or small numbers of hidden units are poorly accurate. Figure 5.4 shows the same tests but boosted in the region of minimisation of the squared error. Neural networks trained with a learning rate equal to 0.004 perform better than the other independently from the number of neurons. Various single hidden layer architectures have a validation MSE of 0.59. Since it is convenient to choose the simplest neural network with highest accuracy, the best result of the grid search is the architecture with 375 hidden units. Since the learning rate seems to be a discriminating factor stronger than the number of hidden units, further tests for checking the MSE for values around 0.004 are executed. With a learning rate of 0.0024 the MSE on the validation set decreases to about 0.56%.

Table 5.1: Mean squared error calculated on neural networks with an only hidden layer with a different number of neurons and for various learning rates. The mean values are calculated by applying the 10-fold cross validation technique at the training data. The standard deviation has been corrected. Entries with MSE smaller than 0.6 are highlighted in bold font.

| Learning rate | Hidden neurons | Mean squared error (%) | Learning rate | Hidden neurons | Mean squared error (%) |
|---|---|---|---|---|---|
| 0.1 | 75 | 4.16 ± 0.46 | 0.0008 | 75 | 1.01 ± 0.04 |
| | 150 | 2.62 ± 0.34 | | 150 | 0.77 ± 0.04 |
| | 225 | 2.06 ± 0.19 | | 225 | 0.72 ± 0.03 |
| | 300 | 1.53 ± 0.08 | | 300 | 0.66 ± 0.01 |
| | 375 | 1.48 ± 0.16 | | 375 | 0.67 ± 0.04 |
| | 450 | 1.37 ± 0.12 | | 450 | 0.63 ± 0.03 |
| | 525 | 1.38 ± 0.11 | | 525 | 0.61 ± 0.03 |
| | 600 | 2.97 ± 3.01 | | 600 | 0.63 ± 0.03 |
| | 675 | 1.39 ± 0.13 | | 675 | 0.66 ± 0.02 |
| | 750 | 3.60 ± 4.37 | | 750 | 0.64 ± 0.04 |
| | 825 | 1.46 ± 0.07 | | 825 | 0.62 ± 0.03 |
| | 900 | 1.51 ± 0.11 | | 900 | 0.62 ± 0.02 |
| 0.02 | 75 | 1.80 ± 0.32 | 0.00016 | 75 | 3.28 ± 0.09 |
| | 150 | 1.23 ± 0.11 | | 150 | 1.28 ± 0.07 |
| | 225 | 1.06 ± 0.08 | | 225 | 1.04 ± 0.03 |
| | 300 | 0.90 ± 0.05 | | 300 | 0.94 ± 0.05 |
| | 375 | 0.84 ± 0.05 | | 375 | 0.92 ± 0.02 |
| | 450 | 0.83 ± 0.05 | | 450 | 0.87 ± 0.04 |
| | 525 | 0.79 ± 0.05 | | 525 | 0.85 ± 0.03 |
| | 600 | 0.78 ± 0.06 | | 600 | 0.87 ± 0.07 |
| | 675 | 0.77 ± 0.02 | | 675 | 0.84 ± 0.04 |
| | 750 | 0.81 ± 0.07 | | 750 | 0.83 ± 0.02 |
| | 825 | 0.75 ± 0.03 | | 825 | 0.80 ± 0.01 |
| | 900 | 0.80 ± 0.05 | | 900 | 0.81 ± 0.04 |
| 0.004 | 75 | 0.92 ± 0.05 | | | |
| | 150 | 0.68 ± 0.04 | | | |
| | 225 | 0.63 ± 0.03 | | | |
| | 300 | 0.62 ± 0.02 | | | |
| | **375** | **0.59 ± 0.02** | | | |
| | 450 | 0.60 ± 0.03 | | | |
| | 525 | 0.60 ± 0.04 | | | |
| | **600** | **0.59 ± 0.02** | | | |
| | 675 | 0.60 ± 0.03 | | | |
| | 750 | 0.61 ± 0.04 | | | |
| | **825** | **0.59 ± 0.02** | | | |
| | **900** | **0.59 ± 0.02** | | | |

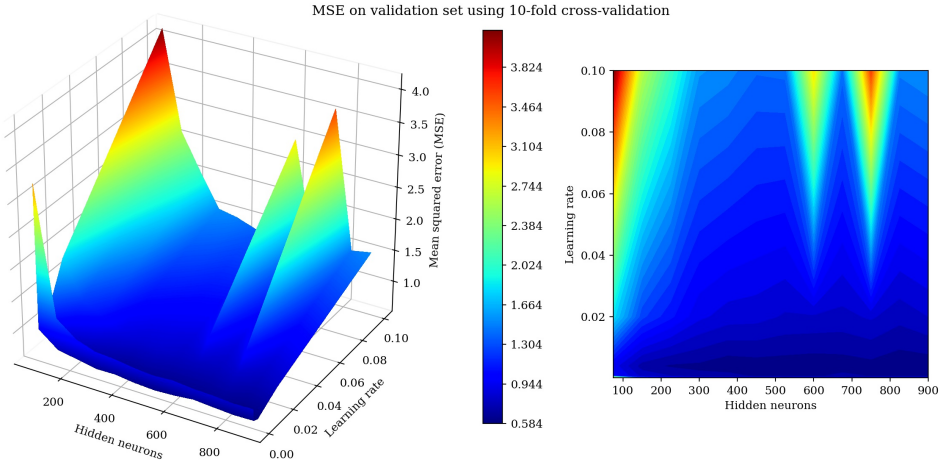



Figure 5.3: Mean squared error calculated on neural networks with one only hidden layer with a different number of neurons and for various learning rates. The mean values are calculated by applying the 10-fold cross validation technique at the training data.

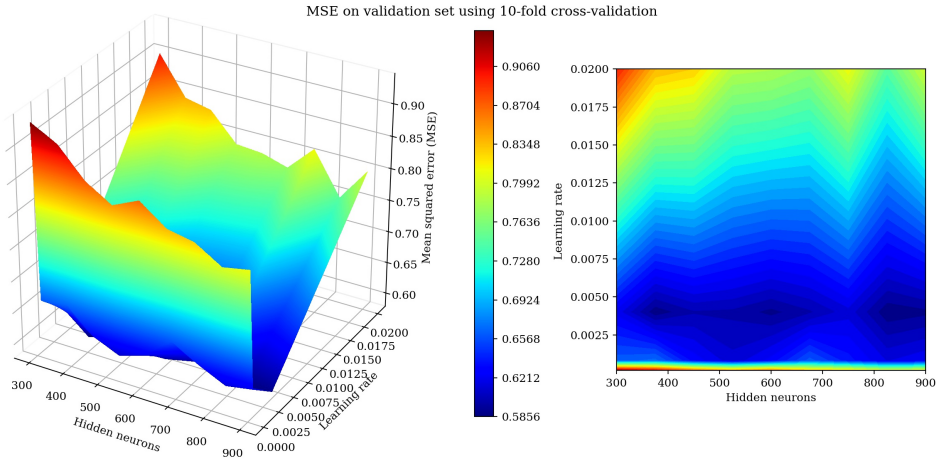



Figure 5.4: Refined region of minimum MSE following from Figure 5.3.

The analogous for the neural network with two hidden layers (starting from 50 and 25 neurons per layer up to 600 and 300) is shown in Table 5.2 and Figure 5.5. As seen above, configurations with large learning rates or small number of neurons are bad choices. The region of MSE minimisation is shown in Figure 5.6. Also for this architecture, 0.004 is the best learning rate. As for the number of hidden units, the best choice with this learning rate value is equal to 350 neurons in the first hidden layer and 175 in the second. With this configuration the validation MSE is about 0.6%. Further tests on learning rate values around 0.004 are executed. The validation MSE decreases to about 0.57% with a learning rate equal to 0.003.

For these experiments the input features corresponding to twelve days time windows have been used. The single hidden layer architecture has a performance comparable to that with two hidden layers. Since when there are many solutions with similar accuracy it is always suggested to adopt the simplest, a study on the complexity appears worthwhile. The complexity of a neural network is proportional to the number of its trainable parameters. For a multi-layer perceptron the number of trainable parameters $N_p$ is:

$$N_p = N_w + N_b, \tag{5.1}$$

where $N_w$ and $N_b$ are the number of connection weights and neuron biases, respectively. For networks with a single hidden layer with $i$ input units, $h$ hidden units and $o$ output units they are defined as:

$$N_w = i \cdot h + h \cdot o \tag{5.2}$$
$$N_b = h + o. \tag{5.3}$$

Since the optimal studied architecture has 579 input neurons, 375 hidden neurons and 1 output neuron:

$$N_w = 579 \cdot 375 + 375 \cdot 1 = 217\,500 \tag{5.4}$$
$$N_b = 375 + 1 = 376 \tag{5.5}$$
$$N_p = 217\,500 + 376 = 217\,876. \tag{5.6}$$

The number of parameters of neural networks with two hidden layers with $h_1$ and $h_2$ hidden units, respectively, is calculated as follows:

$$N_w = i \cdot h_1 + h_1 \cdot h_2 + h_2 \cdot o \tag{5.7}$$
$$N_b = h_1 + h_2 + o. \tag{5.8}$$

Table 5.2: The same of Table 5.1 for neural networks with two hidden layers. Entries with the smallest MSE are highlighted in bold font.

| Learning rate | Hidden neurons | Mean squared error (%) | Learning rate | Hidden neurons | Mean squared error (%) |
|---|---|---|---|---|---|
| 0.1 | (50, 25) | 3.34 ± 0.39 | 0.0008 | (50, 25) | 1.11 ± 0.09 |
| | (100, 50) | 3.40 ± 0.45 | | (100, 50) | 0.78 ± 0.03 |
| | (150, 75) | 3.38 ± 0.50 | | (150, 75) | 0.71 ± 0.04 |
| | (200, 100) | 3.51 ± 0.79 | | (200, 100) | 0.68 ± 0.03 |
| | (250, 125) | 4.22 ± 1.08 | | (250, 125) | 0.65 ± 0.03 |
| | (300, 150) | 3.84 ± 0.62 | | (300, 150) | 0.67 ± 0.03 |
| | (350, 175) | 2.75 ± 0.51 | | (350, 175) | 0.67 ± 0.02 |
| | (400, 200) | 3.38 ± 0.51 | | (400, 200) | 0.64 ± 0.03 |
| | (450, 225) | 3.35 ± 0.73 | | (450, 225) | 0.64 ± 0.05 |
| | (500, 250) | 3.56 ± 0.60 | | (500, 250) | 0.63 ± 0.03 |
| | (550, 275) | 5.07 ± 3.92 | | (550, 275) | 0.61 ± 0.03 |
| | (600, 300) | 3.58 ± 0.59 | | (600, 300) | 0.67 ± 0.06 |
| 0.02 | (50, 25) | 1.08 ± 0.06 | 0.00016 | (50, 25) | 2.30 ± 0.10 |
| | (100, 50) | 0.87 ± 0.07 | | (100, 50) | 1.10 ± 0.07 |
| | (150, 75) | 0.81 ± 0.03 | | (150, 75) | 0.90 ± 0.09 |
| | (200, 100) | 0.82 ± 0.06 | | (200, 100) | 0.81 ± 0.04 |
| | (250, 125) | 0.86 ± 0.05 | | (250, 125) | 0.75 ± 0.05 |
| | (300, 150) | 0.82 ± 0.07 | | (300, 150) | 0.72 ± 0.05 |
| | (350, 175) | 0.84 ± 0.05 | | (350, 175) | 0.75 ± 0.08 |
| | (400, 200) | 0.82 ± 0.05 | | (400, 200) | 0.74 ± 0.07 |
| | (450, 225) | 0.88 ± 0.08 | | (450, 225) | 0.78 ± 0.05 |
| | (500, 250) | 0.92 ± 0.04 | | (500, 250) | 0.76 ± 0.05 |
| | (550, 275) | 0.94 ± 0.10 | | (550, 275) | 0.74 ± 0.04 |
| | (600, 300) | 0.89 ± 0.04 | | (600, 300) | 0.75 ± 0.05 |
| 0.004 | (50, 25) | 0.96 ± 0.04 | | | |
| | (100, 50) | 0.75 ± 0.05 | | | |
| | (150, 75) | 0.68 ± 0.04 | | | |
| | (200, 100) | 0.65 ± 0.06 | | | |
| | (250, 125) | 0.62 ± 0.05 | | | |
| | (300, 150) | 0.62 ± 0.05 | | | |
| | **(350, 175)** | **0.60 ± 0.01** | | | |
| | (400, 200) | 0.63 ± 0.03 | | | |
| | (450, 225) | 0.62 ± 0.04 | | | |
| | (500, 250) | 0.61 ± 0.03 | | | |
| | **(550, 275)** | **0.60 ± 0.02** | | | |
| | **(600, 300)** | **0.60 ± 0.02** | | | |

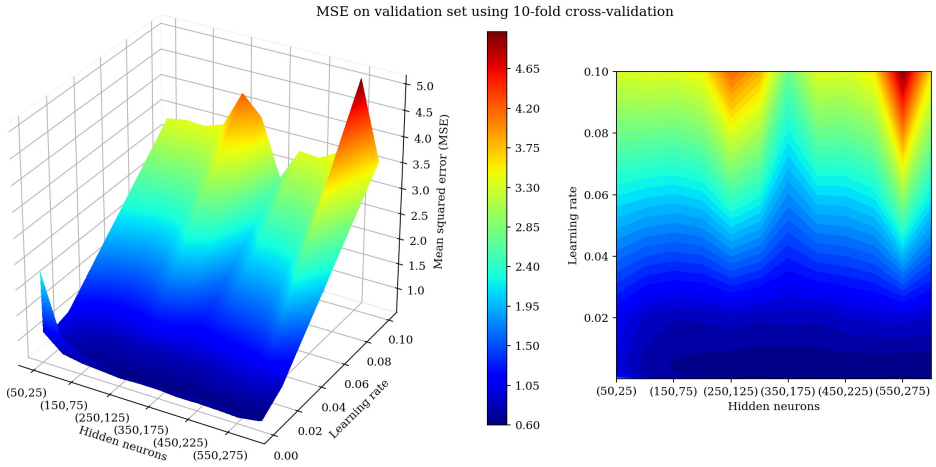



Figure 5.5: The same of Figure 5.3 for neural networks with two hidden layers.

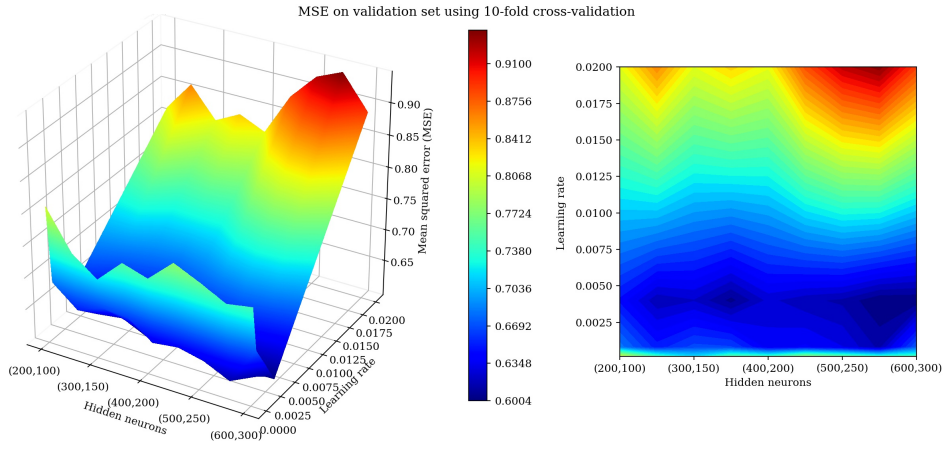



Figure 5.6: Refined region of minimum MSE following from Figure 5.5.

Table 5.3: Complexity and mean squared error measured on validation set for the neural networks trained with the best hyper-parameters found with the grid search.

| Hidden layers | Hidden neurons | Learning rate | Mean squared error (%) | Parameters | Weights | Biases |
|---|---|---|---|---|---|---|
| 1 | 375 | 0.004 | 0.59 ± 0.02 | 217 876 | 217 500 | 376 |
| 1 | 600 | 0.004 | 0.59 ± 0.02 | 348 601 | 348 000 | 601 |
| 1 | 825 | 0.004 | 0.59 ± 0.02 | 479 326 | 478 500 | 826 |
| 1 | 900 | 0.004 | 0.59 ± 0.02 | 522 901 | 522 000 | 901 |
| 2 | (350, 175) | 0.004 | 0.60 ± 0.01 | 264 601 | 264 075 | 526 |
| 2 | (550, 275) | 0.004 | 0.60 ± 0.02 | 470 801 | 469 975 | 826 |
| 2 | (600, 300) | 0.004 | 0.60 ± 0.02 | 528 601 | 527 700 | 901 |
| 2 | (500, 250) | 0.004 | 0.61 ± 0.03 | 415 501 | 414 750 | 751 |
| 2 | (550, 275) | 0.0008 | 0.61 ± 0.03 | 470 801 | 469 975 | 826 |

Since the best studied architecture with two hidden layers has the same number of input and output neurons than that with just one hidden layer and 350 and 175 hidden units in the two hidden layers, it follows:

$$N_w = 579 \cdot 350 + 350 \cdot 175 + 175 \cdot 1 = 264\,075 \tag{5.9}$$

$$N_b = 350 + 175 + 1 = 526 \tag{5.10}$$

$$N_p = 264\,075 + 526 = 264\,601. \tag{5.11}$$

The network with two hidden layers has about 45 000 trainable parameters more than the other. Since the accuracy of the networks with one and two hidden layers are almost identical, it appears a useless complication to choose the second solution. In Table 5.3 are reported the complexity of some of the best architectures considered, together with the hyper-parameter values and the calculated MSEs.

**Input feature selection.** All tests described above are done by using a data set with twelve days of hourly observations for both interplanetary magnetic field intensity $B$ and solar wind speed $V$. It is reasonable to think that some time sub-windows can be more relevant than others. In particular, data referring to instants just before the time considered may affect the result much more than data observed days in advance.

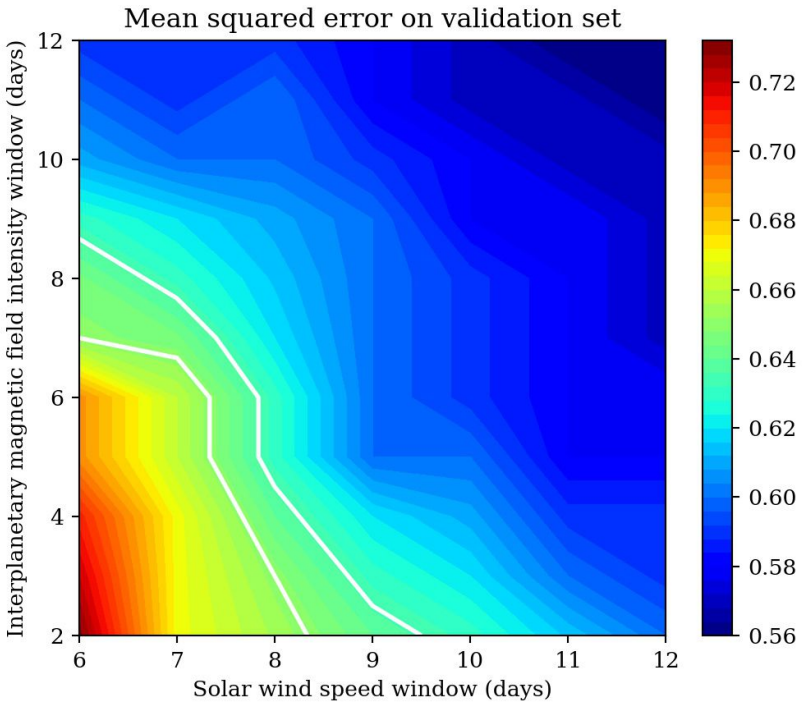



Figure 5.7: Mean squared error calculated on neural networks with different time windows for the input variables (solar wind speed and interplanetary magnetic field). The MSE of each combination is calculated on the validation set with a 10-fold cross validation with different learning rate values and hidden layers and units. The best results for each combination are reported in the figure. The region between the white lines corresponds to the MSE assuming values of approximately 0.65%.

In order to find the minimal time window for both $V$ and $B$ variables, several tests with different cuts of the maximal twelve-day window are executed. These tests were carried out by considering the results showed in the previous paragraph in order to set the grid search on neuron number and learning rate value and using the same 10-fold cross-validation to check if smaller input sets allow for better accuracy in case different hyper-parameter values are chosen. As a successive step, further tests are executed to check the possibility of sampling the input variables on daily cadence, for instance, rather than hourly.

Since the best MSE found with the twelve-day windows is about 0.56% and 0.57% for the one and two hidden layers architecture, respectively, and since the statistical uncertainty of the LISA Pathfinder hourly binned data is 1%, it is

reasonable to accept a smaller input data set that determines neural network model predictions with mean squared error smaller than 0.65%. Sub-windows ranging from six to twelve days for the solar wind speed $V$ and from two to twelve days for the interplanetary magnetic field intensity $B$ are investigated. Each combination of $B$ and $V$ sub-windows is tested with 10-fold cross-validation in order to find the best architecture and learning rate value for the specific combination. The optimum regions of these hyper-parameters are chosen on the basis of the previous results. In Figure 5.7 the best results obtained for each combination of $B$ and $V$ sub-windows are reported. By reducing the number of input variables, the architecture with two hidden layers always performs better than the single one. For example, with seven days of $V$ observations and two for $B$, the architecture with two hidden layers has an MSE of 0.67%, while that with one hidden layer has an MSE of 0.77%. A learning rate equal to 0.005 is on average the best, although for combinations with less input variables a larger value, such as 0.007 or 0.01, seems better. In the region of Figure 5.7 highlighted between the white lines the MSE is nearly constant (about 0.65%). Among all the possibilities contained in this region, nine days of observations for $V$ and two for $B$ is chosen, since it is the one with less input variables with the same output error, but also represents a correlation with the typical duration of solar wind speed and interplanetary magnetic field increases at the passage of high-speed streams that modulate the recurrent variations of the galactic cosmic-ray flux. With these time windows, using a learning rate equal to 0.005 and two hidden layers, with 550 and 275 neurons, respectively, the MSE on the validation set is 0.64% $\pm$ 0.04%. In Figure 5.2 the Bartels rotation 2505 is reported, as in Figure 5.8, but with the optimal value found for the $B$ and $V$ time windows.

The number of input features can be further decreased by sampling differently the time window of each variable. For example, it may be sufficient to get one sample every $n$ hours instead of hourly observations. In order to keep as much as possible the information contained in the data set, each considered observation is substituted with the mean value of the contiguous discarded observations. For instance, by keeping one value every twenty-four hours, each resulting observation is the mean value measured during a 24-hour period. In this work, different values for the parameter $n$, ranging up to 36 hours for $V$ and 24 hours for $B$, are tested with nine days of observations for $V$ and two for $B$ and different learning rate values and hidden unit number. In this case the architecture with only one hidden layer has a worse performance (for instance, with $n = 24$ it has a MSE above 1.1, with respect to the 0.7 of the architecture with two hidden layers). Passing from twelve days of hourly observations for both variables to two days of daily observations for $B$ and nine days of observations for $V$, with one observation every twenty-four hours for $B$ and thirty-six hours for $V$, allows to drop the number of input

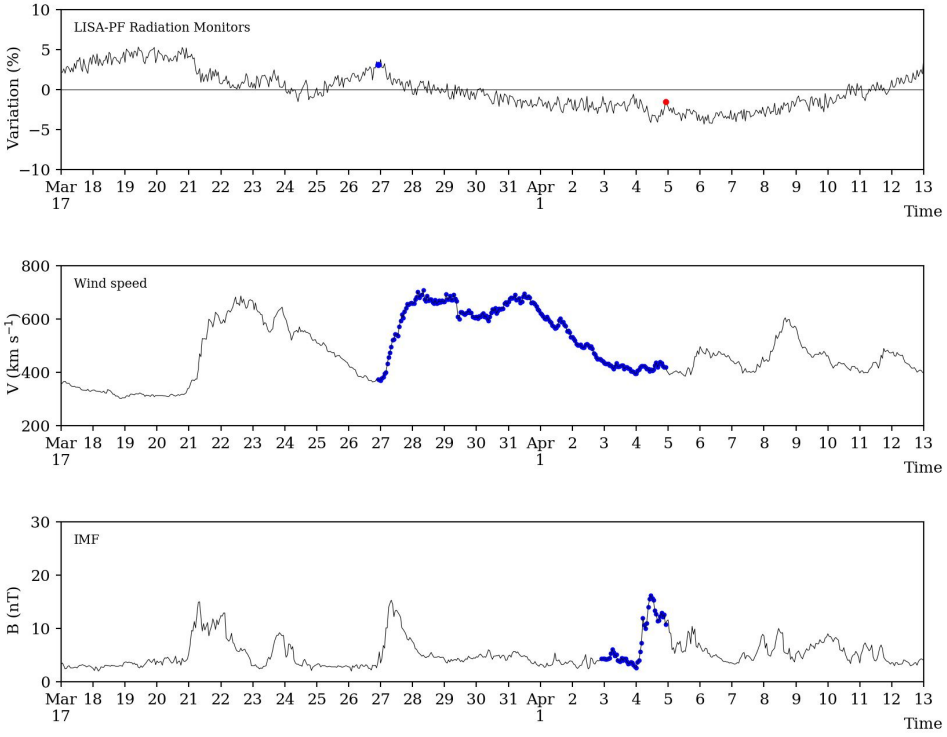



Figure 5.8: Same as Figure 5.2. In this plot the blue dots represent the optimal time windows estimated for the solar wind speed and interplanetary magnetic field intensity past observations included in the data set (nine days for the solar wind and two days for the magnetic field).

features from 579 to 11 (3 variables for $B$, 7 for $V$ and 1 for galactic cosmic-ray flux variation normalisation). The resulting MSEs found with the aforementioned 10-fold cross-validation are reported in Table 5.4. The steps used for $V$ and $B$ appear for each neural network, together with the number of hidden units and the learning rate that minimise the MSE are reported. All tests are executed by averaging $n$ observations and the mean values are the new input data. Without averaging the data, statistical fluctuations show an undesired impact on the prediction accuracy.

The same Bartels rotation shown in Figure 5.8 is also reported in Figure 5.9 and Figure 5.10, but this time with different data sampling rates for $B$ and $V$. The data sample shown in Figure 5.9 is obtained only by keeping one observation every twenty-four for $B$ and every thirty-six for $V$, while in Figure 5.10 sequences

Table 5.4: Mean squared error calculated on the validation set for neural networks with different architectures and learning rates varying the sampling rate of the input variables. In input are taken nine and two days of solar wind speed $V$ and interplanetary magnetic field intensity $B$ observations, respectively.

| $V$ sampling rate (hours) | $B$ sampling rate (hours) | Hidden units | Learning rate | Mean squared error (%) |
|---|---|---|---|---|
| 1 | 1 | 600 | 0.006 | 0.62 ± 0.06 |
| | | (550, 275) | 0.005 | 0.64 ± 0.04 |
| 3 | 3 | 525 | 0.015 | 0.64 ± 0.07 |
| | | (300, 150) | 0.01 | 0.62 ± 0.05 |
| 6 | 6 | 375 | 0.015 | 0.69 ± 0.04 |
| | | (450, 225) | 0.005 | 0.62 ± 0.02 |
| 12 | 12 | 750 | 0.015 | 0.82 ± 0.05 |
| | | (300, 150) | 0.007 | 0.63 ± 0.04 |
| 24 | 24 | 525 | 0.015 | 1.11 ± 0.11 |
| | | (450, 225) | 0.007 | 0.69 ± 0.02 |
| 36 | 24 | (300, 150) | 0.01 | 0.73 ± 0.06 |

of twenty-four and thirty-six observations are substituted with their mean value.

The mean squared errors of the best networks found during the optimisations phase of the work described in this paragraph are shown in Table 5.5. All the information regarding the input variables, the hyper-parameter values and the complexity expressed as number of trainable parameters are reported for each network. Since, as recalled above, the LISA Pathfinder data statistical error is about 1%, the network with input variables $B$ and $V$ observed every twenty-four and thirty-six hours, respectively, appears the best choice with a MSE of 0.73, because it has only eleven input variables.

**Plausibility of chosen variable time windows** The artificial neural network accepts as input data past observations of the solar wind speed and interplanetary magnetic field intensity in terms of time windows of nine days for the first and two days for the second. Different sizes of the time windows have been considered, specifically from six to twelve days for the wind speed and from two to twelve days for the interplanetary magnetic field. These ranges have been considered appropriate for the following reasons. Galactic cosmic-ray short-term variations present

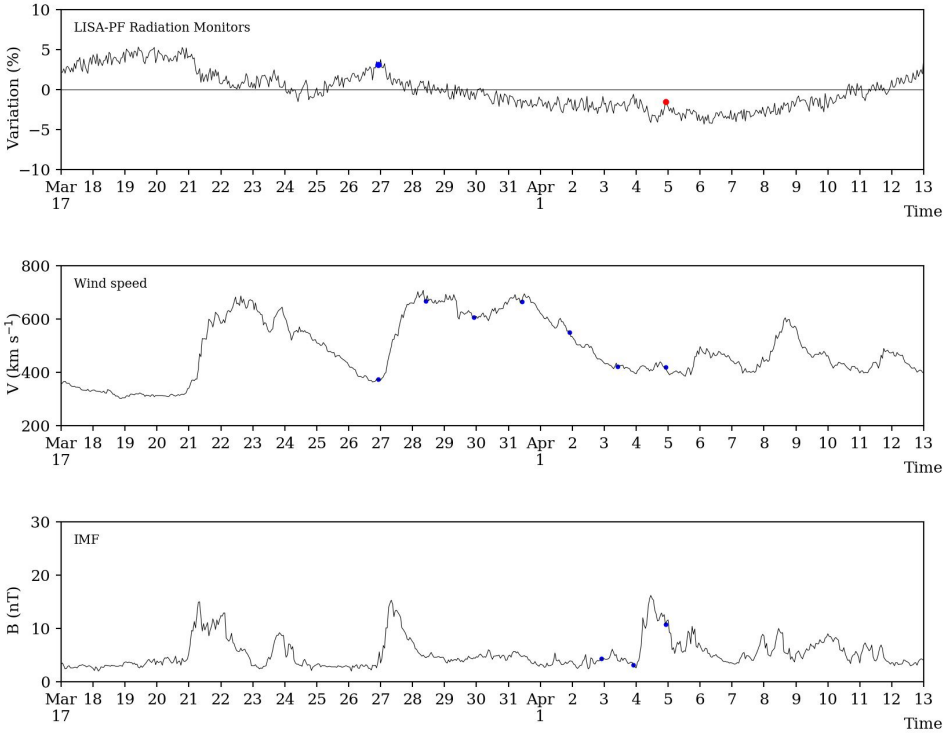



Figure 5.9: Same as Figure 5.8. In this plot the blue dots represent the optimal samplings estimated for the solar wind speed and interplanetary magnetic field intensity past observations included in the data set (thirty-six hours for the solar wind and twenty-four hours for the magnetic field).

quasi-periodicities associated with the solar rotation ($\sim$ 27 days) and higher harmonics. The interplanetary magnetic field intensity and solar wind speed increases show similar periodicities. The LISA Pathfinder cosmic-ray data were gathered during the declining part of the solar cycle 24 between 2016 and 2017, during which a high number of high-speed solar wind streams from equatorial coronal holes and equatorward extensions of polar coronal holes were observed generating forty-five galactic cosmic-ray flux recurrent variations. Conversely, only three non-recurrent Forbush decreases caused by the passage of three interplanetary counterparts of coronal mass ejections were observed for a total duration of six days during the same time. The recurrent galactic cosmic-ray variations presented an average intensity of 5% and an average duration of nine days. The typical duration of

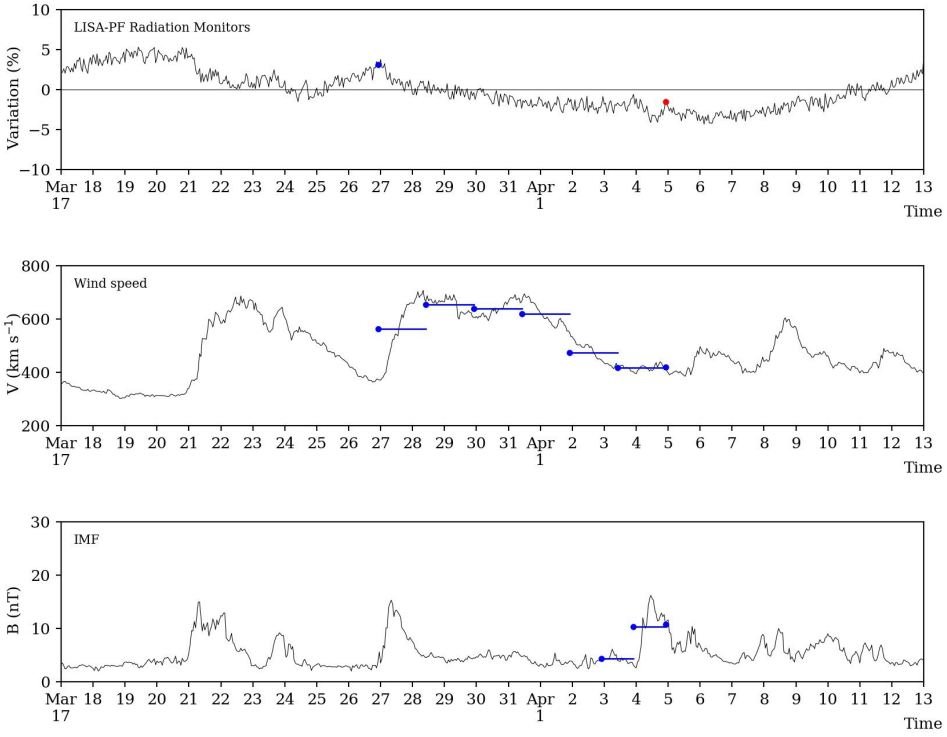



Figure 5.10: Same as Figure 5.9. In this plot each blue dot represents the mean value of the successive observations highlighted with the blue line on the right of the dot.

high-speed solar wind stream passage corresponding to wind velocities larger than 400 km s$^{-1}$ is four days, while for velocities larger than 380 km s$^{-1}$ is five days. The increase of the magnetic field associated with the high-speed stream transit lasts typically two days. The combination of solar wind speed and interplanetary magnetic field at the interaction regions between high-speed solar wind with the preceding slow wind increase causes the galactic cosmic-ray recurrent variations. The model presented here was optimised by starting from the aforementioned observational clues. The time windows for the input variables were initially set to twelve days and then reduced to nine and two days for the wind speed and the magnetic field intensity, respectively. The time window size corresponding to the solar wind speed stream passage results in line with the duration of single high-speed stream transits because, despite several high-speed streams superpose, the

Table 5.5: Complexity and mean squared error measured on validation set for the best optimised neural networks described in the paragraph.

| ($V$, $B$) windows (days) | ($V$, $B$) sampling rate (hours) | Hidden units | Learning rate | Mean squared error (%) | Parameters (weights and biases) |
|---|---|---|---|---|---|
| (12, 12) | (1, 1) | 750 | 0.0024 | 0.56 ± 0.02 | 435 751 |
| (12, 12) | (1, 1) | (500, 250) | 0.003 | 0.57 ± 0.03 | 415 501 |
| (9, 6) | (1, 1) | 450 | 0.0035 | 0.62 ± 0.02 | 164 251 |
| (9, 2) | (1, 1) | (550, 275) | 0.005 | 0.64 ± 0.04 | 299 201 |
| (9, 2) | (3, 3) | 525 | 0.015 | 0.64 ± 0.07 | 48 826 |
| (9, 2) | (24, 24) | (450, 225) | 0.007 | 0.69 ± 0.02 | 108 451 |
| (9, 2) | (36, 24) | (300, 150) | 0.01 | 0.73 ± 0.06 | 48 901 |

solar wind speed goes below 400 km $s^{-1}$ even for short time. Time windows of nine days capture the dynamics of these multiple events. Conversely, since interplanetary magnetic field intensity increases are usually isolated and limited to two-day periods, a time window of size equal to two days is sufficient.

### 5.2.3 Knowledge extraction from the neural network

As shown in Chapter 3, several methods for extracting rules from neural networks solving regression problems exist in the literature. All methods, however, show a complexity growing proportionally to the number of neurons and/or input features. Some algorithms are only applicable to neural networks with one hidden layer. Amongst the algorithms described in Chapter 3, the most adequate for regression problems and networks with more than one hidden layer is ITER (see Algorithm 5), that is chosen for this work. It may be possible to extend REFANN (see Algorithm 3) for extracting rules from neural networks with two hidden layers, but considering also the number of input features of the problem under study and the total number of hidden neurons required to have a good enough prediction, the resulting rules will be so complex to become unintelligible. REFANN provides more regressive outputs than ITER and it may be possible to apply the REFANN algorithm to neural network models with only one hidden layer. However, it has been demonstrated in Section 5.2.2 how two hidden layers perform better with respect to the problem discussed here, so this solution has not been adopted to extract rules. *Rule-extraction-as-learning* and TREPAN (see Algorithm 1 and Algorithm 2, respectively) are not suitable because they are designed for classi-

fication problems, thus they require the transformation of the data set and the neural network accordingly, in order to pass from regression to classification, and for this work is not the case for several reasons. Firstly, the output real-valued variable to predict has to be converted in a categorical variable. This can be done by splitting the range between minimum and maximum of the output variable into $n$ contiguous and non-overlapping sub-ranges, each of them associated to a different class label. Each sub-range $i$ has lower and upper bounds, $a_i$ and $b_i$, respectively. The output value of each data set sample is then substituted with a class label as described in the following. Given the partitioning criterion of the initial output variable range, the output $o_j$ of a given sample $j$ belongs to one and only one sub-range $i$, that for which $a_i <= o_j < b_i$ holds. The higher the $n$ value is, the most the regressive nature of the initial problem is preserved, because a large amount of distinct classes allows the prediction of a wider range of different output values. Conversely, having only few classes causes the implementation of a model able to predict only few distinct output values. Thus, the transformation from regression to classification imply a loss of precision in the output data, that pass from real values to intervals containing those values. A second problem concerns the evaluation of the prediction performance of the model to implement: while it is simple to measure the prediction error between two real values, the same it is not true for two class labels identifying intervals. A further reason to discard *Rule-extraction-as-learning* and TREPAN is their inability to handle real-valued input features (but only categorical features): they would require a transformation of the features analogous to that described above for the output values, to convert real values into intervals associated to class labels. This conversion would provoke more loss of precision and an explosion of the number of input features, resulting in the implementation of a neural network with poor predictive performance and in an extraction algorithm with major complexity. Since a growing complexity corresponds to less readable extracted rules, this feasibility study suggests to adopt neither *Rule-extraction-as-learning* nor TREPAN.

Given all the considerations discussed above, for this thesis work ITER appears to be the best choice. However, with respect to the actual problem, the main drawbacks of applying ITER are two: the output value of the extracted rules is a constant value, different for each rule but equal for all the examples represented by a single rule, so the model produced by the algorithm performs a sort of classification instead of a real regression. The second disadvantage is the number of input features; with eleven features the algorithm create one hyper-cube per rule, where each hyper-cube has eleven dimensions, resulting in elaboration complexity and rules with eleven constraints on the variables. While it is not possible to reduce the number of input features any further, there is a chance to remove the former problem of ITER by implementing a different model that uses the main concepts

of ITER but in synergy with a linear regressor, in order to create hyper-cubes of samples with similar output from which it is feasible extracting rules in the form of linear function of the input features.

**Notation.** Since the goal of extracting regression rules is to find a relationship between input features and output predictions, it is worthwhile to introduce a notation to indicate the input variables in a concise and non-ambiguous way.

Let $X$ be an input variable, $X_0$ is the value of the input variable $X$ at the instant $t = 0$ and $X_j^i$ represents the value of the $X$ variable obtained by averaging the data over the time interval set by $j$ as the number of time intervals of $i$ hours to go backward in time, from $t_1$ through $t_2$ where: $t_1 = -j \cdot i$ and $t_2 = -(j-1) \cdot i$, assuming that the values of $t_1$ and $t_2$ are the number of hours before instant $t = 0$. For instance, $X_1^{24}$ represents the average value of 24 consecutive observations of the $X$ variable from $t_1 = -24$ to $t_2 = -1$, that is the average value of the variable $X$ during the day preceding the current instant; $X_2^{36}$ is the average value between three days and a day and a half before current instant and corresponding $t_1$ and $t_2$ are equal to -72 and -37, respectively.

The best neural network described in Section 5.2.2 to adopt for knowledge extraction is that with nine days of solar wind speed observations averaged every thirty-six hours and two days of interplanetary magnetic field intensity observations every twenty-four hours. The last input variable is the value of the galactic cosmic-ray flux nine days before the instant to predict. The output is the flux value at current time expressed in terms of increment with respect to the flux value nine days before. Adopting the notation described above, let $O$ be the output and $F$ the cosmic-ray flux at time $t = -9 \cdot 24$:

$$O = f(V_0, V_1^{36}, V_2^{36}, V_3^{36}, V_4^{36}, V_5^{36}, V_6^{36}, B_0, B_1^{24}, B_2^{24}, F). \tag{5.12}$$

The goal of the knowledge extraction is to estimate the $f$ function.

**ITER implementation.** The implementation of ITER as shown in Algorithm 5 and Algorithm 6 applied to the case study reported above has a really poor performance. The problems highlighted are several. First of all, in [92] the authors recommend as default update value for each dimension a twentieth of the total size of the dimension. If the dimension number is eleven and each one is partitioned into twenty portions, there are $20^{11}$ sub-cubes contained in the surrounding cube (it is recalled that the latter is the hyper-cube containing all the samples of the problem). Since the feature space is not entirely filled (for example, feature $x$ ranges between $a$ and $b$, while feature $y$ ranges between $c$ and $d$, but there are not examples where $x = a$ and $y = d$), this exaggerated number of sub-portions leads to the repeated hyper-cube expansion during the algorithm toward regions

absolutely not useful for the handled problem. Computationally, this is translated into a very long CPU-intensive work with a useless output. This problem can be overcomed by partitioning the feature space into bigger portions and stopping the hyper-cube expansion when this goes towards regions with no data set samples.

A second problem emerged during the development of the algorithm is related to the (non-)exhaustivity of ITER. The algorithm ends when it is not possible to further expand any hyper-cube; although this stuck configuration is reached in different moments depending on the number, position and size of the initial hyper-cubes and the parameters of the algorithm (such as the value of the single hyper-cube update), ITER ends with a very little prediction capacity (it is able to predict less than 1% of the test set, with accuracy and fidelity variable on the basis of the algorithm parameters). This issue can be addressed by adding extra hyper-cubes in the space left between the existing ones, but since the very first iteration the new cubes added are so little (due to the absence of big empty spaces) that can comprehend only few data set samples. This lead to a huge number of hyper-cubes (i.e. one for each training sample) and, as a consequence, of rules. In addition, since these cubes are so small, the examples of the test set can remain uncovered by the resulting model. In Figure 5.11 are shown the expansions of two hyper-cubes, supposing a problem with two input features. Initially the hyper-cubes are the light blue squares. During the first iteration they expand toward the green squares, then toward the orange ones, the yellow and finally the red. The figure helps to catch the extent of the cube expansion through the algorithm iterations, showing at the same time the dependence of the final configuration from the number, position and size of the starting hyper-cubes. In addition, it is easy to imagine that the growing hyper-cubes tend to leave big blank spaces in the surrounding cube, forcing to add new smaller and smaller cubes.

The problems and considerations presented above remain valid with all reasonable values for the algorithm parameters, even though it is possible to notice several differences by varying the parameter values. Such parameters are the number of initial hyper-cubes (more points produce a more accurate model and more blank spaces between hyper-cubes), the dimension of the initial cubes (the larger the size, the smaller the accuracy and the blank spaces) and the threshold value for the creation of new hyper-cubes instead of expanding the existing ones (higher values tend to produce a more accurate model but with higher number of cubes and therefore of rules).

Considering these criticalities, the idea of altering the original ITER algorithm appears reasonable. For example, it can be convenient to partition the surrounding cube into a predetermined number of sub-cubes of fixed dimension; each cube has for each dimension a minimum and a maximum value and its predicted value is equal to the mean value of the training examples covered by the cube. Since there

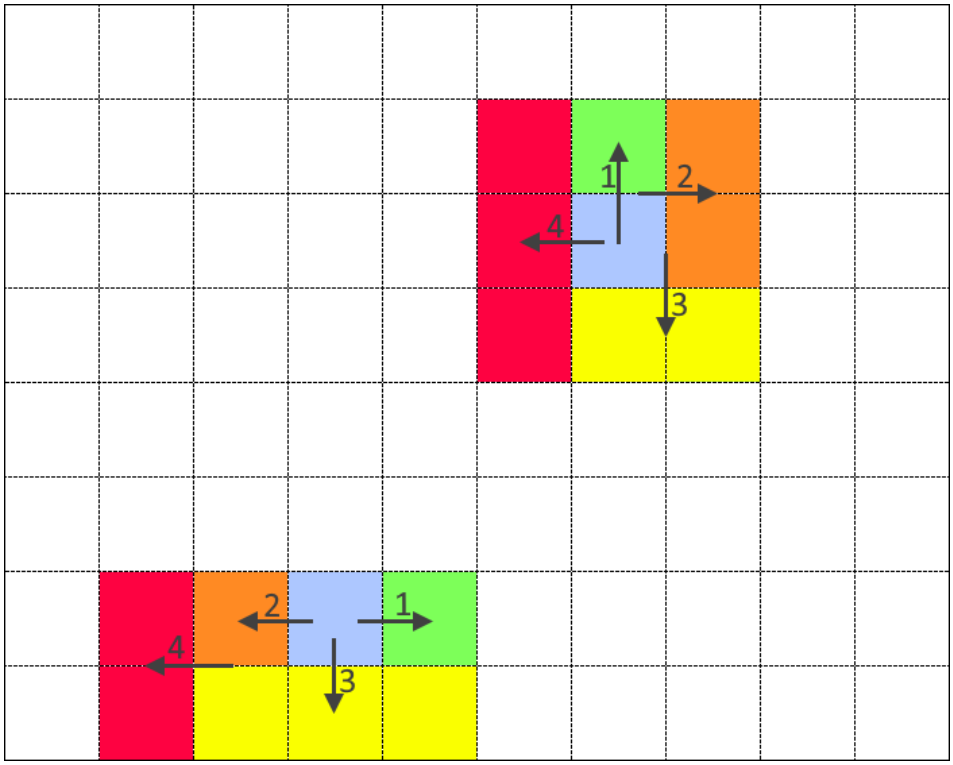



Figure 5.11: Hyper-cube expansion during the iterations of the ITER algorithm.

are eleven dimensions, the number of resulting cubes is $n^{11}$ if $n$ is the number of partitions of each dimension. For instance, $n = 2$ produces over 2 000 cubes, while for $n = 3$ they are more than 175 000. A first optimisation of this algorithm consists in discarding all those useless hyper-cubes that do not cover any sample of the data set. This precaution allows to dramatically reduce the number of effective cubes of the output model. A second optimisation step can be the further split of those hyper-cubes that cover data set examples with high variance in the output value, in order to create sub-cubes more specific and accurate. As a result, the proposed variation of ITER starts splitting the surrounding cube into $n^{11}$ cubic disjoint and contiguous partitions. Each created hyper-cube is discarded if useless; otherwise it is added to the output model or further partitioned into $m^{11}$ sub-cubes if the variance of the examples that covers is higher than an assigned threshold. The value of $m$ can be fixed (such as equal to $n$) or proportional to the sample variance (high variances imply high values of the $m$ parameter). Figure 5.12 shows the proposed variation of ITER. The example assumes a two-dimensional problem. The white square is the surrounding hyper-cube. Each dimension is partitioned into two halves, so at each iteration a square is divided into $2^2$ square splits. During the first iteration the surrounding cube is divided into four light blue squares. Then

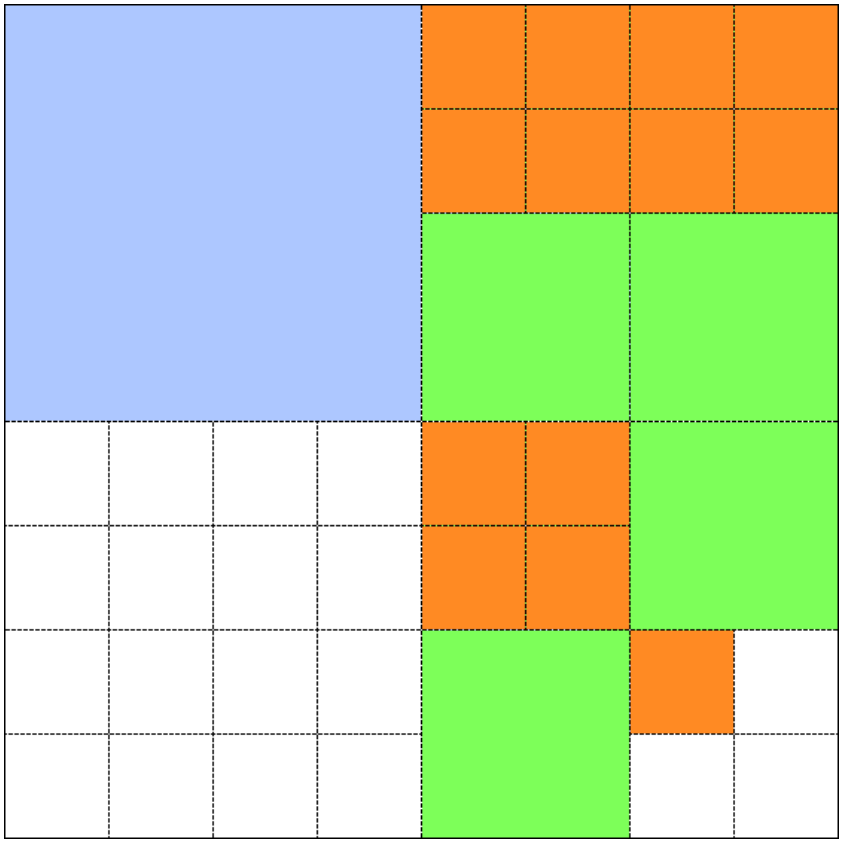


Figure 5.12: Proposed variation of the ITER algorithm.

the usefulness of each square is evaluated. If it is useless, it is discarded (white bottom left square); if it is useful and covers examples with low variance, it is added to the final output model (light blue top right square); otherwise, it is further split into four squares (the green and orange ones in the second and third iterations, respectively). Each new square is recursively evaluated to check its usefulness and variance. For example, by using this technique with two splitting steps ($n = m = 2$) it is possible to achieve the same accuracy than a single step with $n = 4$ but with a very smaller number of resulting hyper-cubes.

Since the surrounding cube is calculated on the training set, it is possible to find in the test set samples that lie outside the cube. In addition, due to discarding useless squares, some test examples may correspond to uncovered regions of the model. These problems are resolved by finding the nearest cube with respect to the example and assigning the example to it. The mean absolute error of the predictions performed by the presented variation of ITER with respect to both the neural network output used to build the understandable model and the real data is reported in Table 5.6. Since the models involved in the comparison present galactic

Table 5.6: Mean absolute error of the predictions made by the variation of ITER with respect to the underlying neural network (NN) and the real data. For each model is reported the number of hyper-cubes.

| First split | Second split | Hyper-cubes | NN MAE (%) | Data MAE (%) |
|---|---|---|---|---|
| 2 | - | 243 | 1.57 | 1.74 |
| 3 | - | 986 | 1.04 | 1.31 |
| 4 | - | 1 984 | 0.51 | 0.81 |
| 2 | 2 | 1 749 | 0.54 | 0.83 |

cosmic-ray flux variation prediction errors smaller than 1% in the majority of cases, in the following the mean absolute error is considered to be more appropriate than the mean squared error for studying the precision of the various models. For each model the number of hyper-cubes is reported.

Each hyper-cube in the output model corresponds to a rule extracted from the underlying neural network. Each rule is formed by a body expressed as a conjunction of clauses on the input variables, such as $(a < x < b) \wedge (c < y < d)$, where $x$ and $y$ are input variables and $a, b, c, d$ are threshold values, and an output value. For instance, let $E$ be a sample of the test set expressed accordingly to the notation described above:

$$\begin{aligned} E &= (V_0, V_1^{36}, V_2^{36}, V_3^{36}, V_4^{36}, V_5^{36}, V_6^{36}, B_0, B_1^{24}, B_2^{24}, F) \\ &= (546.0, 384.0, 366.6, 486.8, 460.1, 360.6, 420.2, 4.9, 12.2, 4.8, 2.92). \end{aligned}$$

The sample $E$ is covered by the hyper-cube $HC$, defined as follows by using the proposed variation of ITER and two splitting steps with parameters $m = n = 2$:

Average increment is about -1.83% if:
- $V_0$ is between 508 and 630 km s$^{-1}$
- $V_1^{36}$ is between 284 and 390 km s$^{-1}$
- $V_2^{36}$ is between 285 and 390 km s$^{-1}$
- $V_3^{36}$ is between 390 and 495 km s$^{-1}$
- $V_4^{36}$ is between 390 and 495 km s$^{-1}$
- $V_5^{36}$ is between 284 and 390 km s$^{-1}$
- $V_6^{36}$ is between 390 and 495 km s$^{-1}$
- $B_0$ is between 0.90 and 7.50 nT
- $B_1^{24}$ is between 9.65 and 13.56 nT
- $B_2^{24}$ is between 1.87 and 5.71 nT

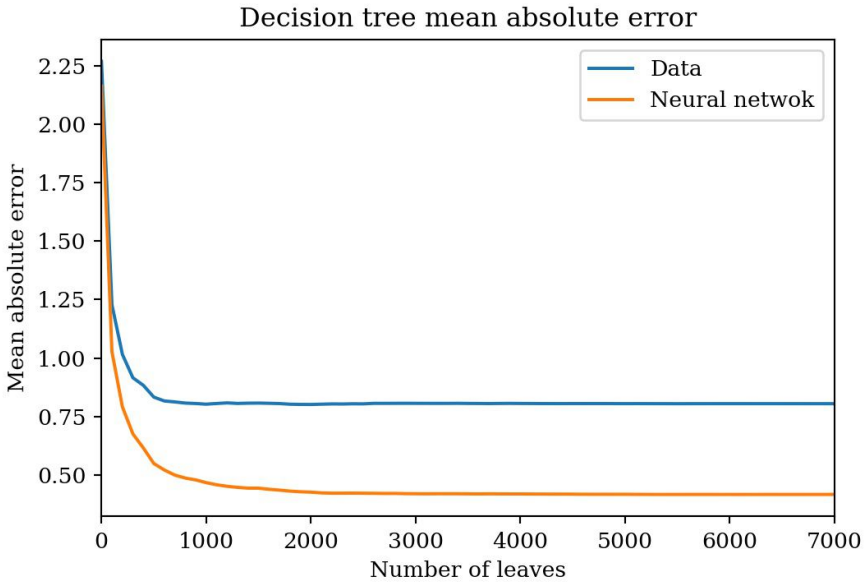



Figure 5.13: Mean absolute error measured on the predictions of a regression tree trained with the output of the presented neural network. The errors are calculated with respect to the network predictions (orange line) and to the real data (blue line) varying the number of leaf nodes.

- $F$ is between 0.48 and 4.31%.

According to the extracted rule, the sample belongs to a region of the feature space where the predicted increment is about $-1.83\%$. The increment relative to the same sample predicted by the underlying neural network is $-1.61\%$, while the real increment is $-1.82\%$.

The example above is obtained by applying the proposed variation of the ITER algorithm. Since in [92] the authors of ITER compare the performance of this algorithm to a classical binary regression tree, the same comparison has been done between the proposed ITER variation and a regression tree, both applied to the case study described in this work. The mean absolute error versus the number of leaves of a regression tree trained with the output of the presented neural network with respect to both the underlying neural network predictions and the real data (orange and blue lines, respectively) are reported in Figure 5.13. Since each leaf is associated to one rule (as each ITER hyper-cube), it is clear that the tree can achieve the same or even better accuracy of the ITER variation with about the 50% of rules.

Although the ITER variation addresses some issues encountered during the

**Algorithm 7** Algorithm for the creation of an explainable model that uses the main ITER concepts and a linear regressor.

1. select an input sample
2. build a hyper-cube centered in the sample
3. generate $n$ random samples belonging to the hyper-cube
4. predict the output value of each generated sample with the artificial neural network
5. calculate the mean output value and assign it to the hyper-cube
6. expand the hyper-cube
   - create temporary adjacent hyper-cubes as described by the ITER algorithm
   - repeat steps 3, 4 and 5 for each temporary adjacent hyper-cube
   - select the most similar temporary adjacent hyper-cube with respect to the main hyper-cube (the one which mean value is the nearest to that of the main hyper-cube)
   - if the difference between the selected hyper-cube mean value and that of the main hyper-cube exceeds the defined threshold, it is not possible to expand further the hyper-cube; go to step 7
   - merge the main hyper-cube with the selected one and recalculate the mean value
   - remove all the temporary hyper-cubes and repeat step 6
7. train a linear regressor with the samples inside the expanded hyper-cube
8. predict the output of the initial sample with the linear regressor
9. use the regressor parameters to obtain a linear function that explains the prediction obtained in step 8

classical implementation and ignoring the disadvantageous comparison with regression trees, there are two main problems in its provided explanation: the model is not globally intelligible by a human, because there are thousands of rules, and the regression problem has been converted into a multi-class classification problem, since each hyper-cube assigns the same output value to each sample that it covers. In other words, if there are $n$ cubes, the new problem is a classification with $n$ classes.

These problems derive from the nature itself of ITER and can be resolved by applying the core concepts of the algorithm in a different way. The steps allowing for the implementation of the new explainable model described in the following are reported in Algorithm 7. Instead of the creation of an explainable model capable of mimic the underlying neural network, it seems to be a better choice to focus on a single sample and try to explain the specific output returned by the neural network for the sample. For instance, when the neural network receives

a sample, it can be seen as a point in the feature space. It seems reasonable to use the ITER concepts to create only a single hyper-cube, centered on this point, and expanding it using the neural network as oracle in order to find the biggest region in the feature space that contains samples with similar output values. The similarity of the output values is calculated as the difference between the mean value inside the hyper-cube and that of the adjacent cubes, and comparing this difference with a threshold. When the hyper-cube corresponding to the promising region is found, it is possible to train a linear regressor in order to find not a single value as output of the rule (as in the ITER algorithm), but a linear equation of the input variables. It is worthwhile to notice that training a global linear regressor on the whole data set does not produce good results. Since local regressors are more accurate, this seems the most reasonable approach to get an explanation for the output values of the neural network. Using the neural network as an oracle means to generate random samples that lie in a certain hyper-cube and get the network predictions for these samples, in order to have a larger number of examples in each region and thus more robust mean values. This process is shown in Figure 5.14, with two-dimensional hyper-cubes and $n = 25$ as parameter for the number of random examples to create inside each temporary cube. Initially a hyper-cube is constructed around the sample to predict (the blue square with the central blue dot). Then $n$ samples are generated inside the hyper-cube (the black dots) and predicted with the neural network. The mean value is reported in the label inside the square. Two temporary hyper-cubes are created for each dimension (the red dotted squares) and then $n$ random samples are generated into each temporary hyper-cube. The mean of each hyper-cube is then calculated. If there is at least one cube with a mean value that differs from the starting cube mean less than the specified threshold, then the starting cube is merged with the temporary cube with the more similar mean (the green dashed square). The random samples of this best temporary cube are kept, while other cubes and samples are discarded. This is repeated until there are no further expansions left.

As output instance, from the sample $E$ previously described it is possible to create the hyper-cube $HC'$ defined as follows:

Average increment is about -1.64% if:
- $V_0$ is between 534 and 558 km s$^{-1}$
- $V_1^{36}$ is between 373 and 394 km s$^{-1}$
- $V_2^{36}$ is between 356 and 377 km s$^{-1}$
- $V_3^{36}$ is between 476 and 497 km s$^{-1}$
- $V_4^{36}$ is between 450 and 471 km s$^{-1}$
- $V_5^{36}$ is between 350 and 371 km s$^{-1}$
- $V_6^{36}$ is between 410 and 431 km s$^{-1}$

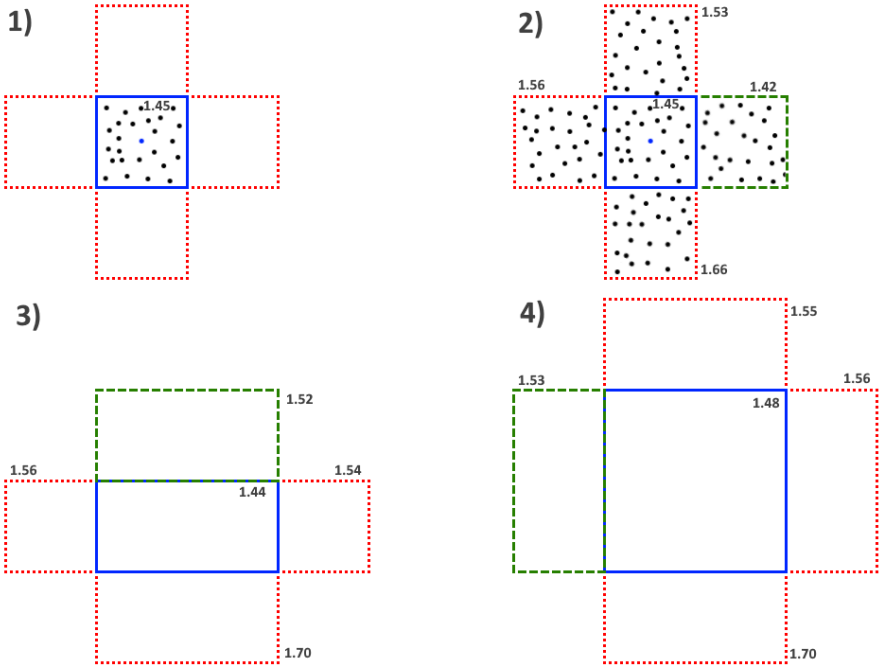



Figure 5.14: Proposed algorithm for the expansion in the feature space starting from a single data set example.

- $B_0$ is between 4.24 and 5.56 nT
- $B_1^{24}$ is between 11.86 and 12.64 nT
- $B_2^{24}$ is between 4.42 and 5.19 nT
- $F$ is between 2.54 and 3.30%.

The $HC'$ hyper-cube has been created by using 20 as parameter for defining the number of equal portions of each input dimension (and thus their size) and 0.1 as threshold for stopping the cube expansion. The mean absolute error measured between the output of this model and both the expected values and those predicted by the underlying neural network with respect to the variation of the splitting parameter value is reported in Table 5.7. It is evident how this parameter is not very impacting on the final predictions, since large values imply less expanding iterations, while small ones requires more iterations, but in both cases the final cube is similar. From $HC'$ it is possible to extract a linear regression rule to predict the output value of the samples belonging to the hyper-cube and, in particular, of $E$:

Table 5.7: Mean absolute error of the predictions of the proposed model with respect to the underlying neural network (NN) and the real data. The data reported is referred to different values of the splitting parameter.

| Splitting parameter value | NN MAE (%) | Data MAE (%) |
|---|---|---|
| 10 | 0.25 | 0.73 |
| 15 | 0.14 | 0.69 |
| 20 | 0.09 | 0.68 |

$$O_E = 0.01V_0 - 0.0048V_1^{36} - 0.026V_2^{36} + 0.0021V_3^{36} + 0.0091V_4^{36} - 0.018V_5^{36} - 0.011V_6^{36} - 0.31B_0 - 0.61B_1^{24} + 0.29B_2^{24} - 1.5F = -1.65$$

It is worthwhile to recall that the real output of $E$ is $-1.82\%$, while the prediction of the neural network is $-1.61\%$. The prediction of the hyper-cube $HC'$ is $-1.64\%$ and, finally, the regression rule gives an output value of $-1.65\%$. By merging the hyper-cube ranges with the regression rule, the final explanation extracted for the provided output of the sample $E$, valid also for other samples covered by $HC'$ is the following:

The increment is $0.01V_0 - 0.0048V_1^{36} - 0.026V_2^{36} + 0.0021V_3^{36} + 0.0091V_4^{36} - 0.018V_5^{36} - 0.011V_6^{36} - 0.31B_0 - 0.61B_1^{24} + 0.29B_2^{24} - 1.5F$

if:

- $V_0$ is between 534 and 558 km $s^{-1}$
- $V_1^{36}$ is between 373 and 394 km $s^{-1}$
- $V_2^{36}$ is between 356 and 377 km $s^{-1}$
- $V_3^{36}$ is between 476 and 497 km $s^{-1}$
- $V_4^{36}$ is between 450 and 471 km $s^{-1}$
- $V_5^{36}$ is between 350 and 371 km $s^{-1}$
- $V_6^{36}$ is between 410 and 431 km $s^{-1}$
- $B_0$ is between 4.24 and 5.56 nT
- $B_1^{24}$ is between 11.86 and 12.64 nT
- $B_2^{24}$ is between 4.42 and 5.19 nT
- $F$ is between 2.54 and 3.30%.

## 5.3 Programming language

All the codes written during this thesis work are in Python programming language. Python is known for being an interpreted, high-level and general-purpose programming language designed to emphasize code readability. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for both small and large-scale projects. Python is dynamically typed and garbage-collected. It can be easily interfaced with other languages and allows the programmer to write application with graphical user interface. Python supports multiple programming paradigms, including structured, object-oriented and functional, and has a large number of standard libraries. Between the existent paradigms, the most used is the imperative, where the program is seen as a sequence of instructions. Each instruction represents an order given to the computer. As for the imperative syntax, often its constructs are expressed as verbs at the imperative mood.

It is possible to download, install and use Python completely free and it is portable; it is sufficient to have the correct interpreter version installed on the device. Python portability derives from being a pseudo-compiled language (the source code is analysed and executed by an interpreter without being compiled). A drawback of this is the lower speed with respect to the compiled languages. Actually Python is faster than purely interpreted languages, because initially the source code undergoes a pre-compiling step in order to translate it into a bytecode, an intermediate language between Python and machine language. The successive code executions have a better performance because the generated bytecode is reused instead of reinterpreting the source code. A more efficient execution can be obtained by using additional modules, such as Numba, providing just-in-time compilers in order to compile in machine language some portions of the source code. Its use is particularly convenient to optimize mathematical and vectorial operations.

Python is particularly suitable for executing scientific computing thanks to the module NumPy; it consents the use of multi-dimensional vectors and vectorised functions that allow to avoid classical loops, inefficient and thus slower. Other modules allow to generate random numbers, data display and drawings (even in three dimensions), downloading files through File Transfer Protocol (FTP) and read/write files in Common Data Format (CDF), used as instance by the National Aeronautics and Space Administration (NASA) and ESA for the dissemination of science data among different communities. Python external libraries also offer powerful tools for designing and optimising neural networks and doing other machine learning experiments, for example the Scikit-learn and Keras libraries.

Another point of strength of using Python are the built-in high-level data structures, such as tuples, lists, sets and dictionaries, that allow the programmer

to save time during the code writing and have a cleaner and clearer code.

The version of Python adopted here is the 3.7.

# Chapter 6

# Results

The predictive models presented in this work aim to reproduce the galactic cosmic-ray flux short-term variations observed on board the LISA Pathfinder mission. The data set samples used as input data by the models are composed of solar wind speed and interplanetary magnetic field observations and one preceding galactic cosmic-ray flux variation measurement for normalisation. The models return as output the percent variations of the cosmic-ray flux with respect to past values. In Section 6.1 a comparison between the different models is reported; in Section 6.2 the neural network model is applied in order to carry out predictions about LISA Pathfinder cosmic-ray missing data.

## 6.1 Comparison between the implemented models

The first implemented model is the artificial neural network trained with the LISA Pathfinder data. The second model is the variation of ITER described in the previous chapter. Finally, the latter is the model obtained by applying the main ITER concepts together with a linear regressor and using the neural network as predicting oracle. The prediction performance evaluation of each model is reported in Table 6.1 in terms of mean absolute error and standard deviation with respect to the data. Errors are measured on the test set initially separated from the whole data set. Results are reported for each Bartels rotation during which the LISA Pathfinder was taking cosmic-ray data (March 2016 - June 2017); the last row contains the results about the entire test set. For each entry the number of samples of both training and test sets are reported.

From Figure 6.1 through Fig. 6.18 the model predictions are shown in comparison with the LISA Pathfinder mission data gathered from March 17th, 2016 through July 2nd, 2017. In all figures the galactic cosmic-ray flux percent variation measurements are indicated by black continuous lines in top panels while the

Table 6.1: Mean absolute error (MAE) and standard deviation of the predictions made with the three models described in the previous chapter (the artificial neural network (ANN), the variation of ITER and the third model using ITER concept and a linear regressor) with respect to the data of the test set. Results are reported for each Bartels rotation. The number of samples of both training and test sets is also indicated.

| Bartels rotation | Training samples | Test samples | ANN MAE (%) | ITER MAE (%) | Regressor MAE (%) |
|---|---|---|---|---|---|
| 2491 | 482 | 166 | 0.89 ± 0.71 | 0.98 ± 0.81 | 0.88 ± 0.71 |
| 2492 | 486 | 162 | 0.87 ± 0.65 | 1.09 ± 0.88 | 0.89 ± 0.65 |
| 2493 | 496 | 152 | 0.71 ± 0.59 | 0.86 ± 0.78 | 0.70 ± 0.58 |
| 2494 | 481 | 167 | 0.77 ± 0.59 | 0.92 ± 0.72 | 0.76 ± 0.59 |
| 2495 | 492 | 156 | 0.77 ± 0.65 | 0.85 ± 0.75 | 0.81 ± 0.69 |
| 2496 | 493 | 155 | 0.84 ± 0.60 | 0.99 ± 0.99 | 0.85 ± 0.63 |
| 2497 | 474 | 174 | 0.67 ± 0.47 | 0.95 ± 0.82 | 0.69 ± 0.48 |
| 2498 | 411 | 147 | 0.64 ± 0.52 | 0.94 ± 0.85 | 0.63 ± 0.51 |
| 2499 | 481 | 167 | 0.75 ± 0.53 | 0.98 ± 0.88 | 0.75 ± 0.53 |
| 2500 | 474 | 174 | 0.64 ± 0.48 | 0.98 ± 0.84 | 0.64 ± 0.48 |
| 2501 | 492 | 156 | 0.66 ± 0.49 | 1.04 ± 1.02 | 0.71 ± 0.51 |
| 2502 | 489 | 159 | 0.67 ± 0.49 | 1.05 ± 1.01 | 0.69 ± 0.51 |
| 2503 | 458 | 151 | 0.78 ± 0.54 | 0.97 ± 0.75 | 0.79 ± 0.53 |
| 2504 | 394 | 107 | 0.67 ± 0.56 | 0.83 ± 0.71 | 0.66 ± 0.57 |
| 2505 | 500 | 148 | 0.60 ± 0.41 | 0.70 ± 0.55 | 0.60 ± 0.39 |
| 2506 | 388 | 120 | 0.71 ± 0.65 | 0.94 ± 0.87 | 0.73 ± 0.68 |
| 2507 | 477 | 171 | 0.68 ± 0.55 | 0.87 ± 0.81 | 0.67 ± 0.54 |
| 2508 | 470 | 178 | 0.67 ± 0.53 | 0.83 ± 0.83 | 0.71 ± 0.54 |
| All | 8 490 | 2 829 | 0.72 ± 0.57 | 0.94 ± 0.84 | 0.73 ± 0.57 |

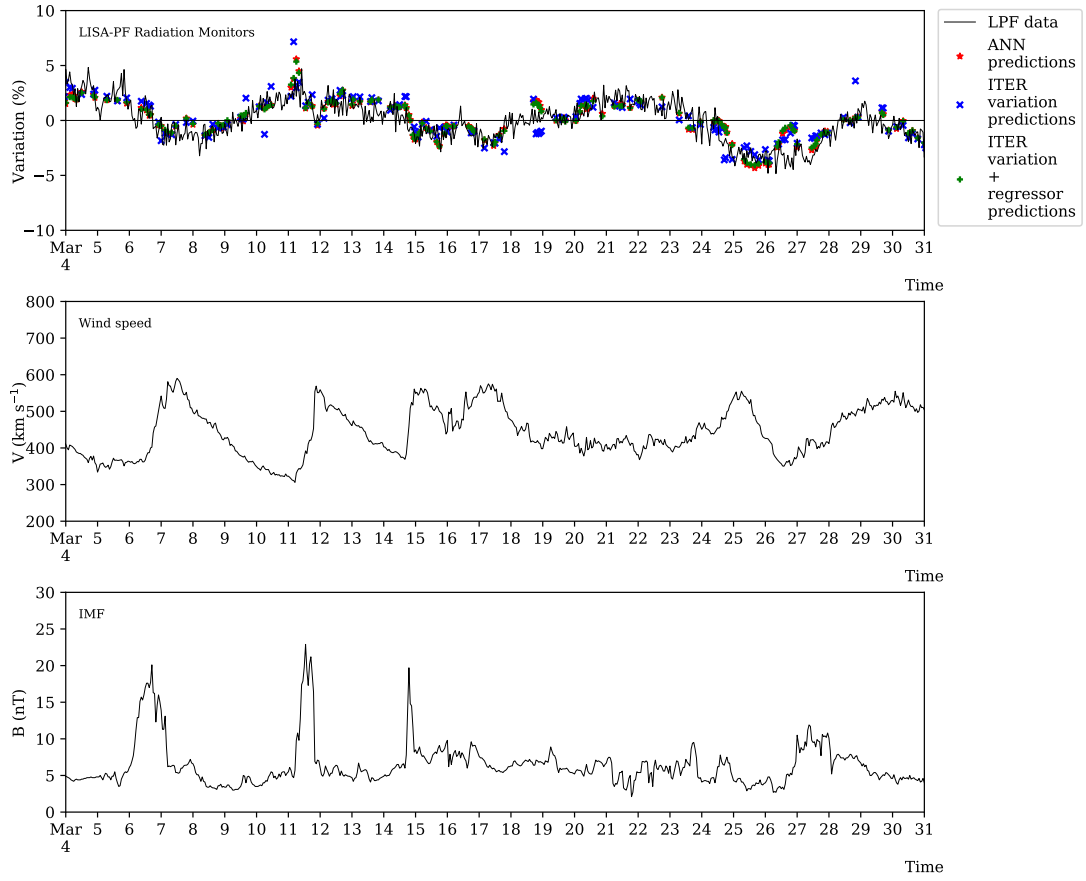



Figure 6.1: Galactic cosmic-ray flux percent variations (black continuous line in top panel), solar wind speed (middle panel) and interplanetary magnetic field intensity (bottom panel) measured during the Bartels rotation 2491, from March 4th, 2016 through March 30th, 2016. In the top panel is also reported the predictions obtained as output of the three described models: red asterisks represent the artificial neural network predictions; blue exes indicate those obtained with the variation of ITER; green crosses show the predictions of the model using ITER concepts with a linear regressor.

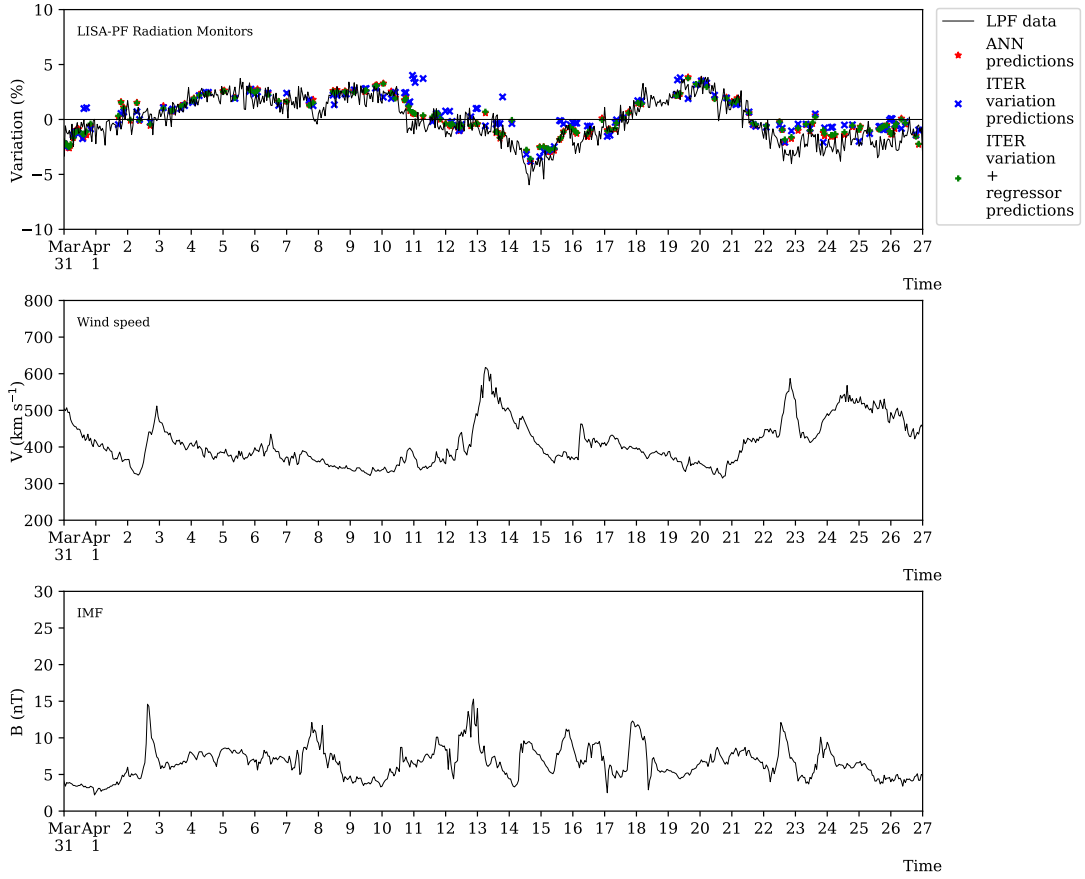



Figure 6.2: Same as Fig. 6.1 for the Bartels rotation 2492, from March 31st, 2016 through April 26th, 2016.

solar wind speed and interplanetary magnetic field intensity measurements appear in the middle and bottom panels. In the top panels of each figure predictions are also reported as outputs of the three described models: red asterisks represent the artificial neural network predictions; blue exes are those made by the variation of ITER and green crosses are the predictions of the model using ITER concepts with a linear regressor. All the predictions are performed on the test set. It is worthwhile to notice that Bartels rotations 2498, 2503, 2504 and 2506 (shown in Figures 6.8, 6.13, 6.14 and 6.16, respectively) present missing data.

The mean absolute error measured for each model results to be independent from the Bartels rotation considered for the observations. The all three model outcomes are in agreement with data within $\pm 1\%$ typical of the statistical uncertainty of the LISA Pathfinder data. However, the neural network model and that using ITER concepts with a linear regressor perform slightly better than the ITER

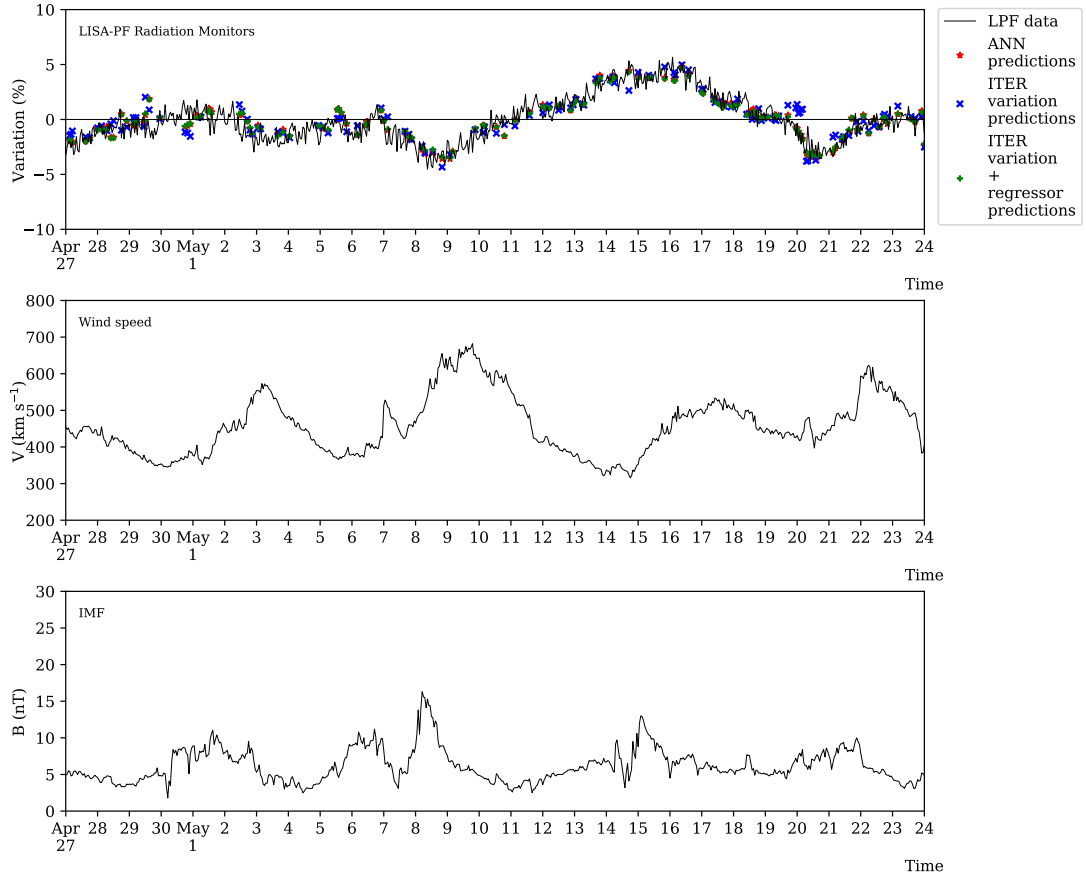



Figure 6.3: Same as Fig. 6.1 for the Bartels rotation 2493, from April 27th, 2016 through May 23rd, 2016.

model. For instance, applied to the Bartels rotation 2501, the neural network has a MAE equal to 0.66%, while the ITER variation prediction MAE is greater than 1%. The average prediction of the three models is resumed in the last row of Table 6.1, which shows that the artificial neural network and the model with the linear regressor present almost the same prediction performance, with a mean absolute error smaller than 0.75%. The ITER variation has a MAE equal to 0.94%. Since the model using the ITER concepts and the regressor is both more understandable by humans than the neural network and more accurate than ITER, it results to be the best choice for predicting galactic cosmic-ray flux variations.

As an example in the following it will be focused on the Bartels rotations 2496 (from July 17th, 2016 through August 12th, 2016) and 2507 (from May 10th, 2017 through June 5th, 2017), shown in Figures 6.6 and 6.17, respectively. The Bartels rotation 2496 is characterised by a cosmic-ray flux variation ranging from

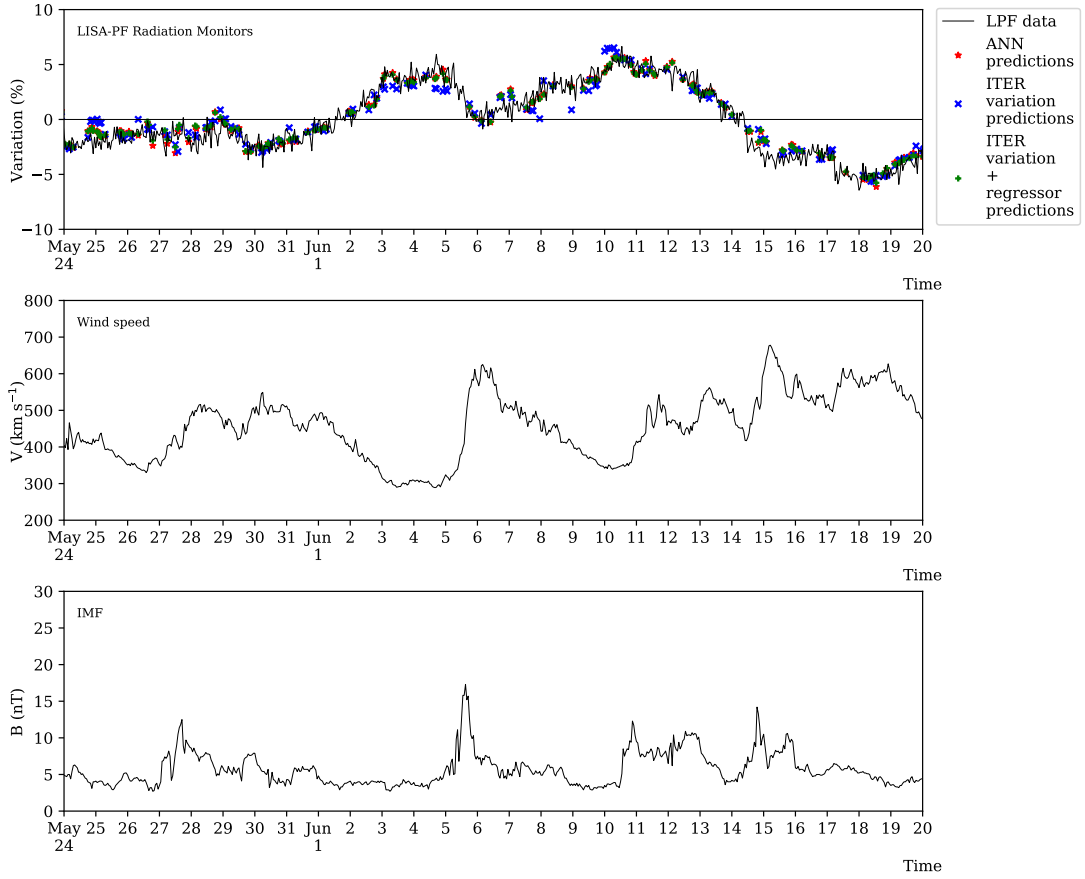



Figure 6.4: Same as Fig. 6.1 for the Bartels rotation 2494, from May 24th, 2016 through June 19th, 2016.

−6% through +9% with respect to the average monthly value. During this Bartels rotation a Forbush decrease occurred between August, 2nd and 3rd. It is worthwhile to recall that Forbush decreases are characterised by sudden drops of the galactic cosmic-ray flux due to a large interplanetary magnetic field intensities associated with the passage of the interplanetary counterpart of a coronal mass ejection. Another evident peculiarity of this Bartels rotation is the presence of multiple superposed high-speed solar wind streams, during which the solar wind speed never decreases below 400 km s$^{-1}$ and the interplanetary magnetic field has small values (starting from August, 4th through the end of the Bartels rotation). All the implemented models are able to make predictions with small errors during this Bartels rotation. The only exception is represented by the ITER variation, that has a poor prediction performance between July, 31st and August, 3rd. It must be pointed out, however, that the Forbush decrease results from an inter-

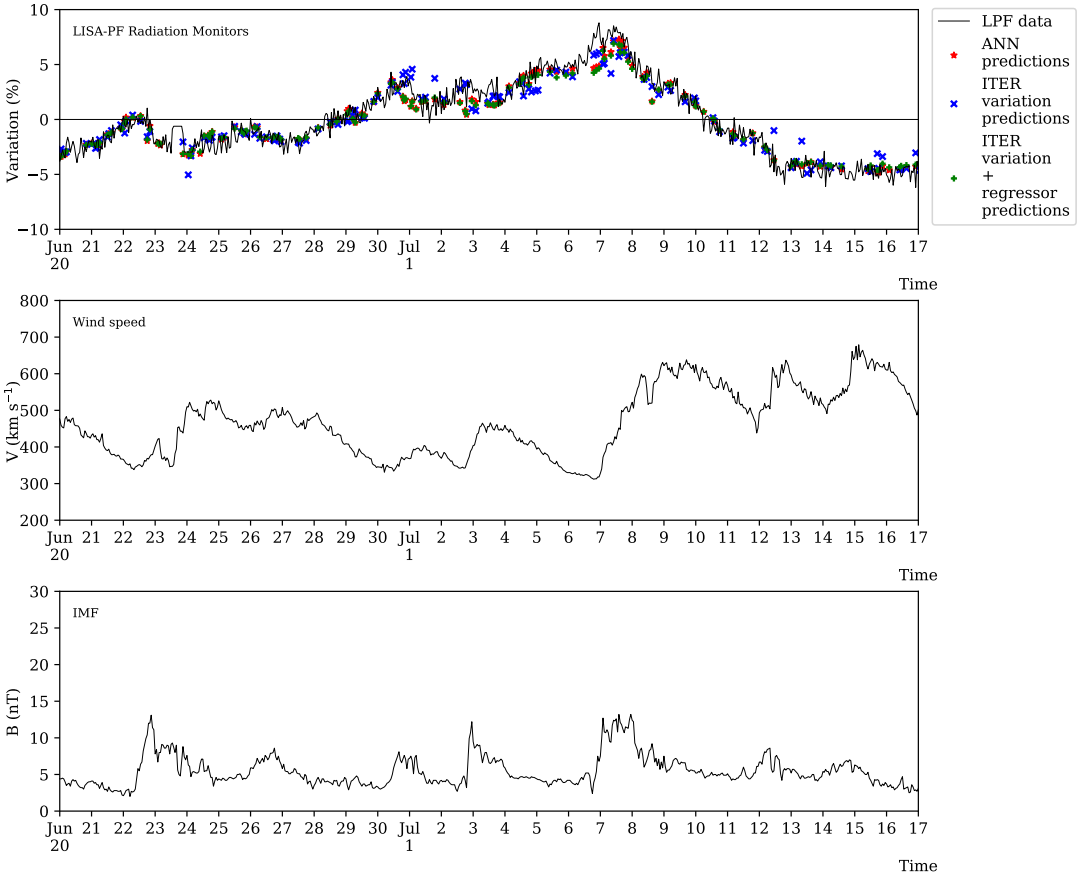



Figure 6.5: Same as Fig. 6.1 for the Bartels rotation 2495, from June 20th, 2016 through July 16th, 2016.

planetary magnetic structure that has completely different characteristics from the high-speed solar wind speed.

Differently from the Bartels rotation 2496, the Bartels rotation 2507 exhibits a flat trend (flux variations are included between $-5\%$ and $+4\%$), with the only exception represented by a Forbush decrease of small intensities between May, 27th and May, 28th 2016. Even in this case the implemented models provide predictions with errors smaller than the LISA Pathfinder statistical uncertainty, except the ITER model.

During the LISA Pathfinder mission the solar wind speed varied from 263 through 761 km s$^{-1}$, the interplanetary magnetic field intensity never exceeded 28 nT and the galactic cosmic-ray flux variations ranged between $-7\%$ and $+9.2\%$. The minimum and maximum of the flux variations are reached during the Bartels rotations 2502 and 2496, respectively. The cosmic-ray variations of the data set

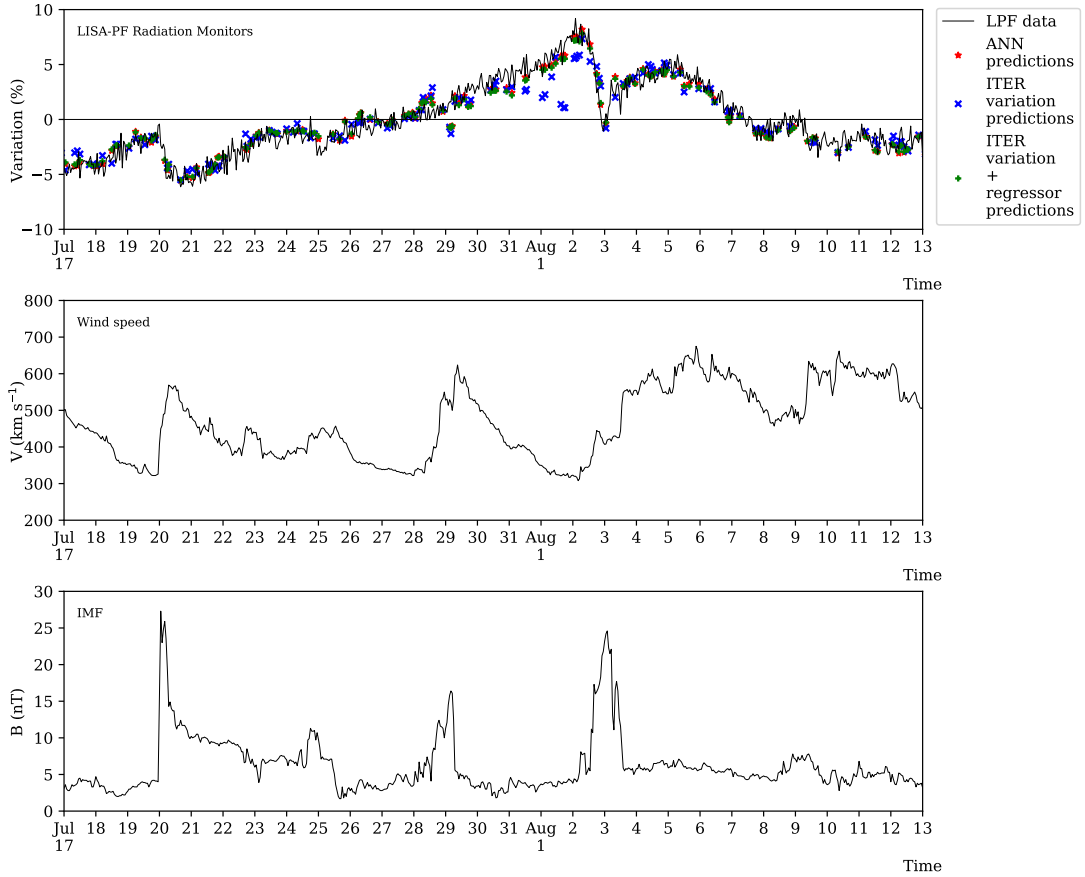



Figure 6.6: Same as Fig. 6.1 for the Bartels rotation 2496, from July 17th, 2016 through August 12th, 2016.

samples included in the test set ranges from −6.6% through +9.2%, while the training set range is between −7% and +8.7%. The difference in the training and test ranges is due to the reduced number of samples having outputs smaller than −6.6% and larger than +8.7%, i.e. only four and two, respectively. Since the data set is randomly split into training and test sets, the four samples which output is included between −7% and −6.6% are all in the training set, while the two samples with the output greater than 8.7% are both in the test set.

The three implemented models appear to be unable to reproduce the higher values of the galactic cosmic-ray flux variation range: the artificial neural network predictions vary from −6.4% through +8.2%; those of the ITER variations belong to the interval between −6.7% and +7.3%; the model using the main ITER concepts with the linear regressor provides outputs from −6.3% to +7.8%. This performance issue is ascribable, as said above, to the reduced number of available

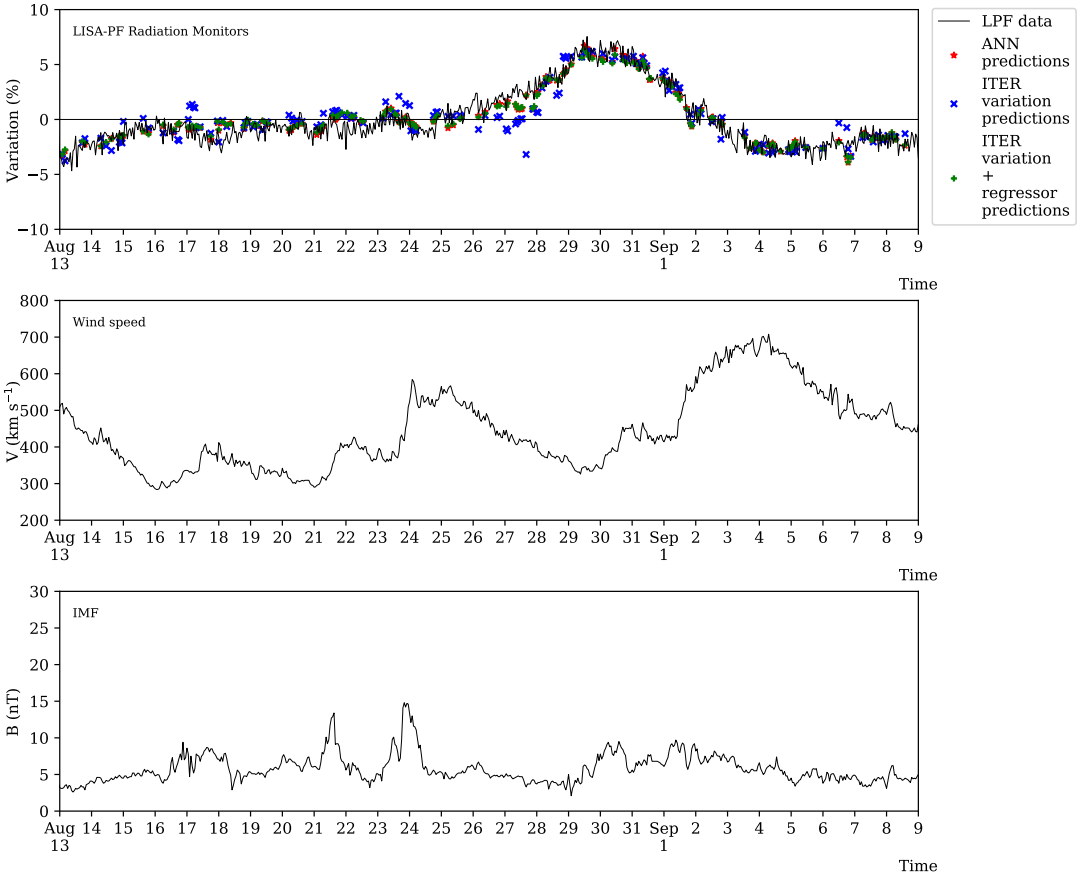



Figure 6.7: Same as Fig. 6.1 for the Bartels rotation 2497, from August 13th, 2016 through September 8th, 2016.

samples with high output values during the training of the neural network, but has little impact on the average predictive performance since those extreme values of the galactic cosmic-ray flux variations are very rare. During the LISA Pathfinder mission elapsed time 12 432 hourly observations were gathered. Only thirty-four of these are above the maximum flux variation value predicted by ITER, equal to +7.3%. The number decreases further with respect to the maximum of the model using the ITER concepts with the regressor and that of the artificial neural network: only nineteen and seven observations are larger than their maximum predicted value of +7.8% and +8.2%, respectively. A similar conclusion holds for the smaller flux variation values measured during the mission.

The implemented models have the main drawback to make predictions by only considering several observations gathered in preceding time windows. If one of these observations is not available, the models cannot be applied. The same oc-

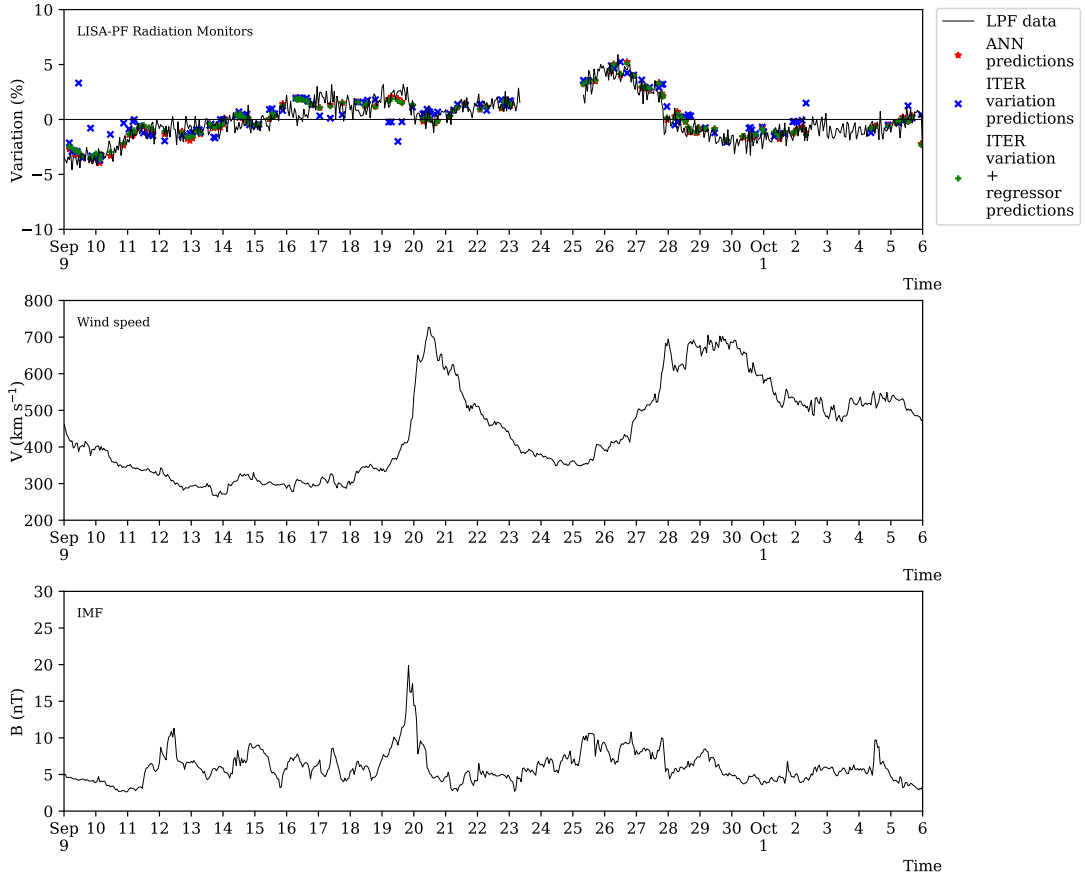



Figure 6.8: Same as Fig. 6.1 for the Bartels rotation 2498, from September 9th, 2016 through October 5th, 2016. There are missing LISA Pathfinder data during this Bartels rotation.

curs if there is no access to the past galactic cosmic-ray flux variation value used for normalisation. The data set created for this work uses solar wind speed and interplanetary magnetic field intensity observations that do not present missing data. However, some data are missing in the LISA Pathfinder time series. Since the models require those missing data to perform predictions about the flux variation values nine days after, these unpredictable periods are present neither in the training set, nor in the test set and therefore, in the corresponding plots, predictions are missing (see Bartels rotations 2498, 2504 and 2506 in Figures 6.8, 6.14 and 6.16, respectively).

As for the knowledge extraction from the neural network model, this thesis work proofs that it is possible to obtain linear functions that combine the input variable observations to obtain the output galactic cosmic-ray flux variation and

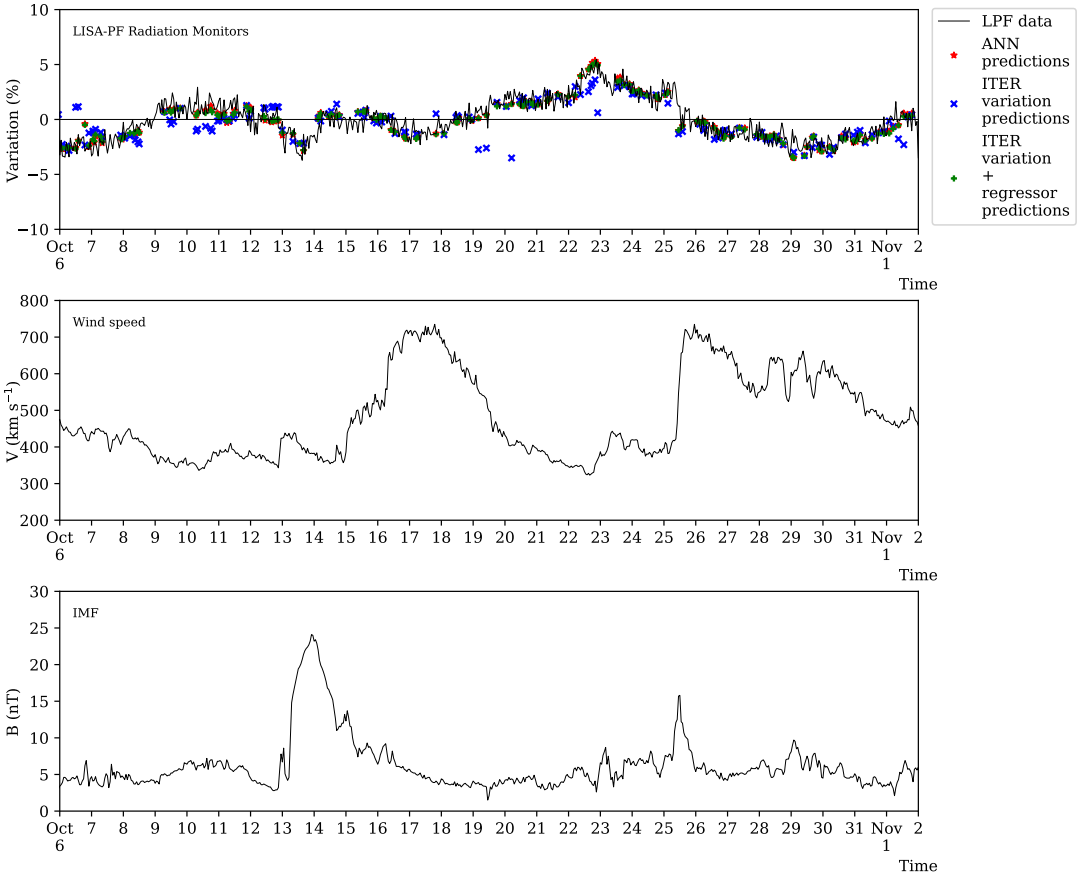



Figure 6.9: Same as Fig. 6.1 for the Bartels rotation 2499, from October 6th, 2016 through November 1st, 2016.

that these functions approximate the relationship between the input and output variables with an error smaller than the LISA Pathfinder data statistical uncertainty. This condition holds only if the rules are extracted in limited regions of the input feature space; it is not feasible to obtain a global function that is able to accurately calculate the output cosmic-ray flux variation for all the possible input solar wind speed and interplanetary magnetic field intensity observation values due to the peculiarities of the difference between interplanetary structures and solar wind high-speed streams [129]. The model for the rule extraction from the neural network described in this work consents to identify these input feature space limited regions.

Summarising, both the artificial neural network and the model based on the ITER concepts and using the linear regressor are able to provide predictions of the galactic cosmic-ray flux variations observed on board the LISA Pathfinder mission

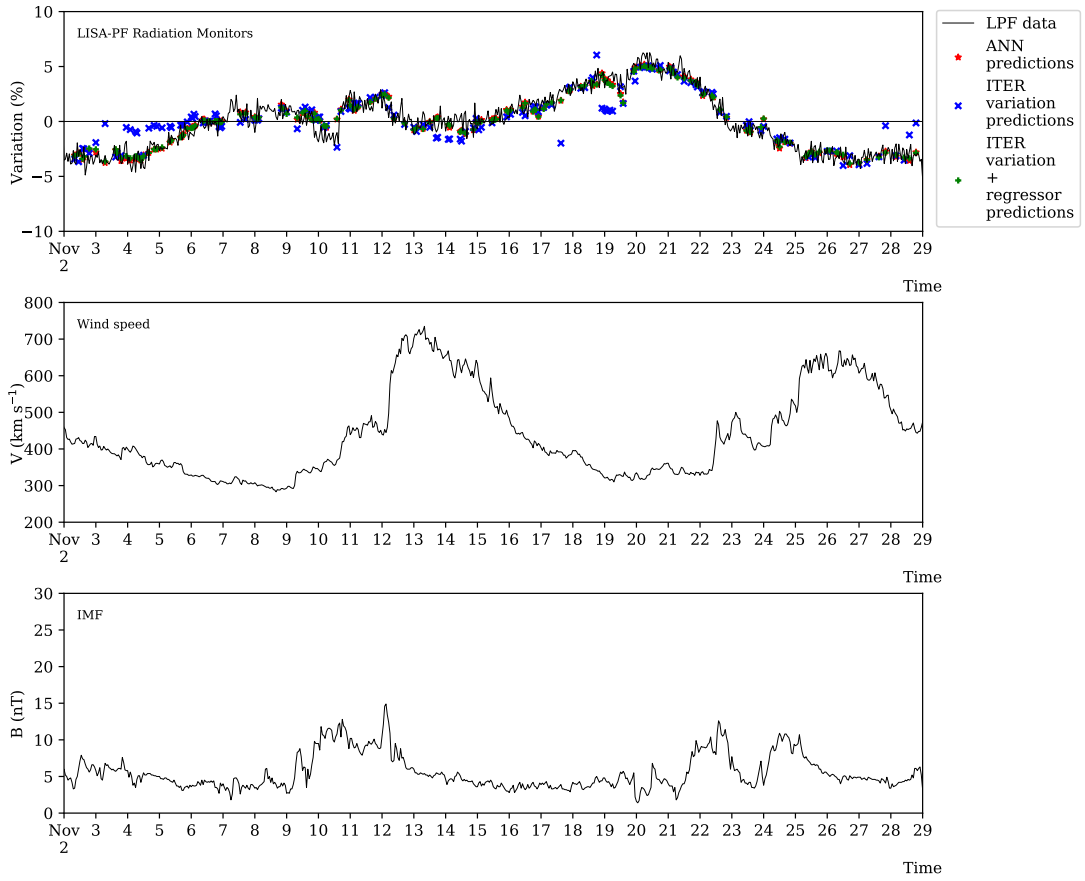



Figure 6.10: Same as Fig. 6.1 for the Bartels rotation 2500, from November 2nd, 2016 through November 28th, 2016.

with an average error smaller than the mission data statistical uncertainty, but they need past observations of the input variables and if those data are not available, the models cannot be applied. Since the neural network has a slightly better performance, it is suggested to use it when no output explanation is required. Finally, the knowledge extraction is feasible only if carried out in limited regions of the input feature space.

## 6.2 Prediction of LISA Pathfinder galactic-cosmic ray flux variation missing data

The implemented models, and in particular the artificial neural network due to its better accuracy, can be used to fill the missing data in the LISA Pathfinder

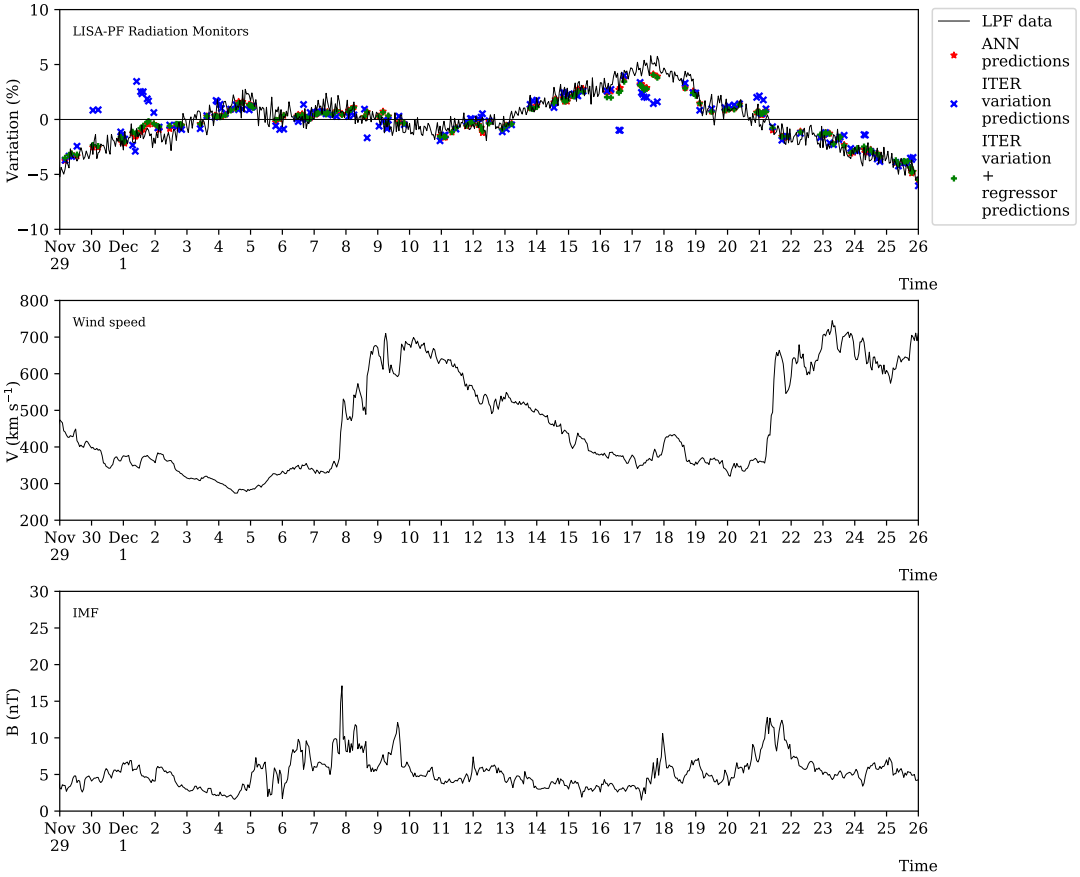



Figure 6.11: Same as Fig. 6.1 for the Bartels rotation 2501, from November 29th, 2016 through December 25th, 2016.

time series. The results of this test are reported in Figures 6.19-6.22. The neural network predictions are represented with red dots in the top panel of each figure. The predicted flux variations show a good agreement with the LISA Pathfinder data gathered immediately before and after the missing data periods.

Further studies and tests can be done on the predictions of the missing data. For instance, it is possible to perform predictions using as input flux normalisation data the artificial neural network output in order to estimate the goodness of the missing data predictions. If the predictions based upon model outcome have a small mean absolute error with respect to the LISA Pathfinder data, there is a quantitative indication of accurate missing data predictions. Within this thesis work only one test of this kind was carried out, as a hint for future application in space. The result of this test is reported in Figure 6.23. The plot refers to the Bartels rotation 2506 and shows the predictions provided by the neural network

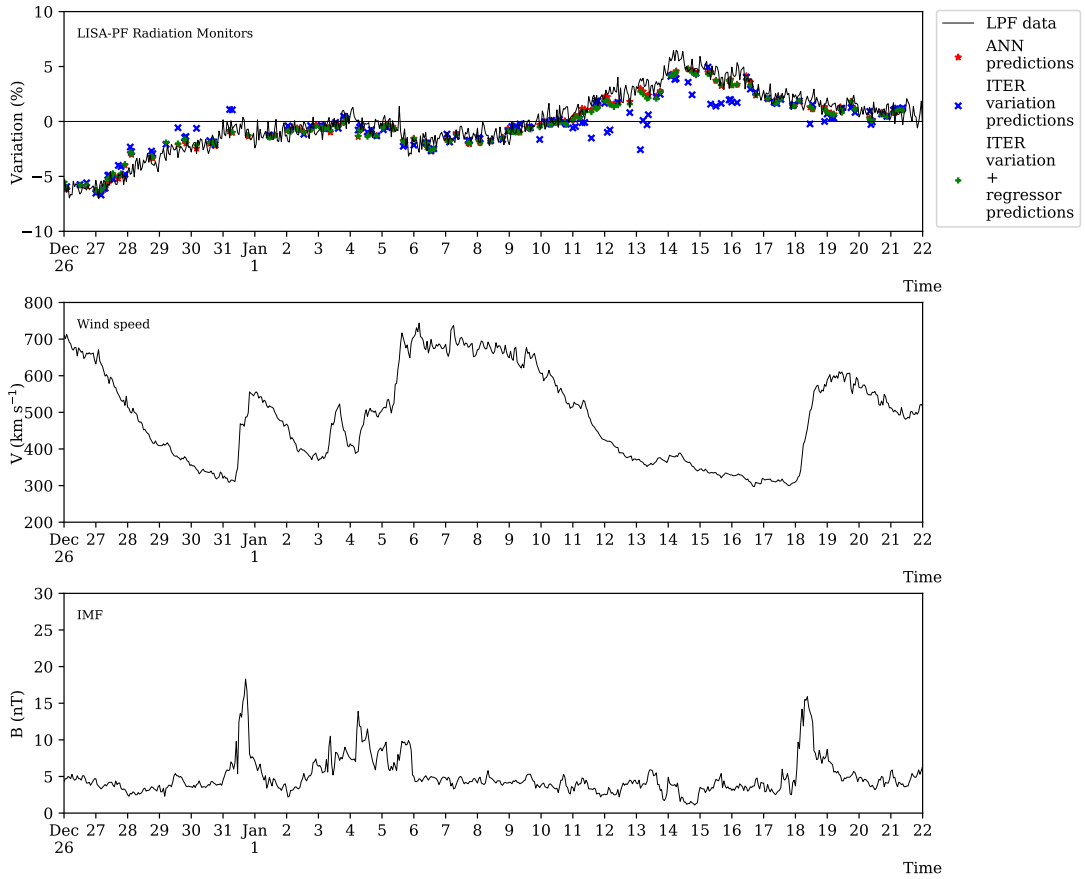



Figure 6.12: Same as Fig. 6.1 for the Bartels rotation 2502, from December 26th, 2016 through January 21st, 2017.

model for the missing data between April, 13th and April 19th (represented by the red dots in the top panel). These predictions are used, in turn, as input flux normalisation to predict the galactic cosmic-ray flux variations between April, 22nd and 28th (represented by the blue dots in the top panel). The predictions generated on the basis of the model output have a MAE equal to $0.72\% \pm 0.54\%$. The quality of these predictions is comparable to that found with the neural network on the test set.

Summarising, the presented artificial neural network model can be used to predict the missing cosmic-ray flux variation data of the LISA Pathfinder mission with a good agreement with respect to the observed data immediately preceding and following the predictions.

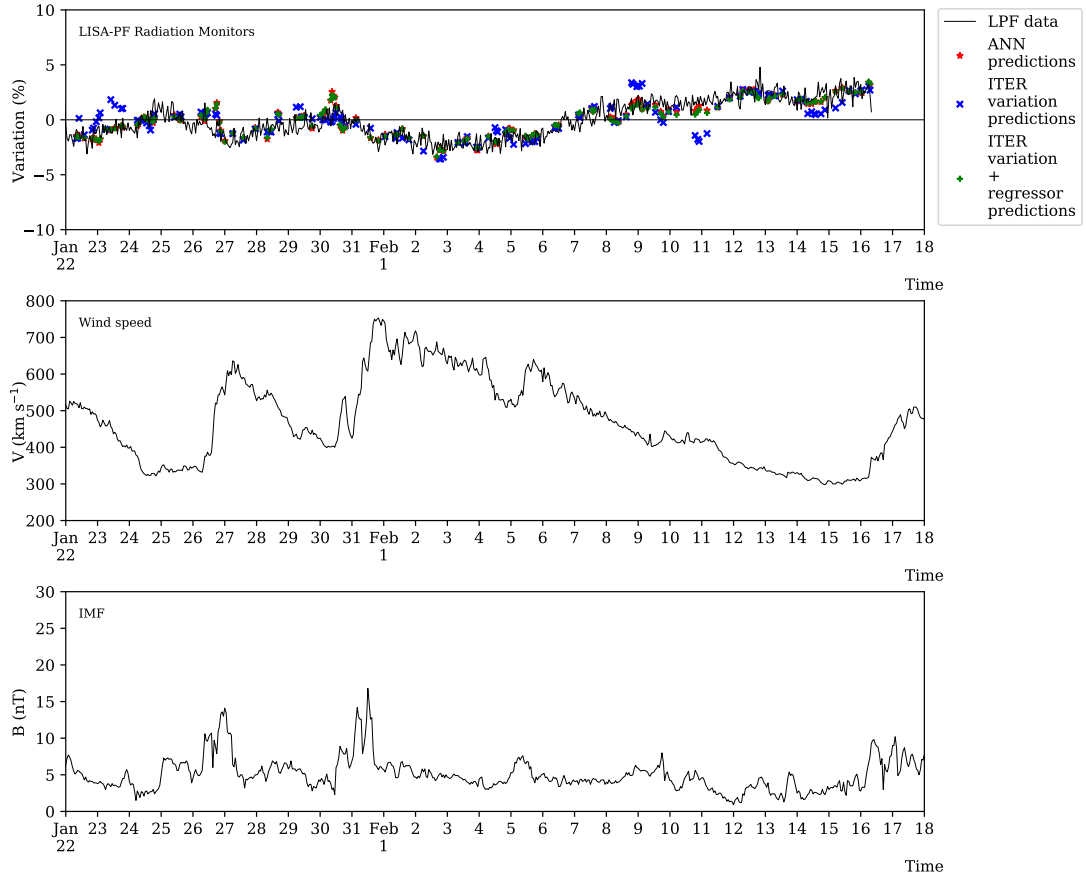



Figure 6.13: Same as Fig. 6.1 for the Bartels rotation 2503, from January 22nd, 2017 through February 17th, 2017. There are missing LISA Pathfinder data during this Bartels rotation.

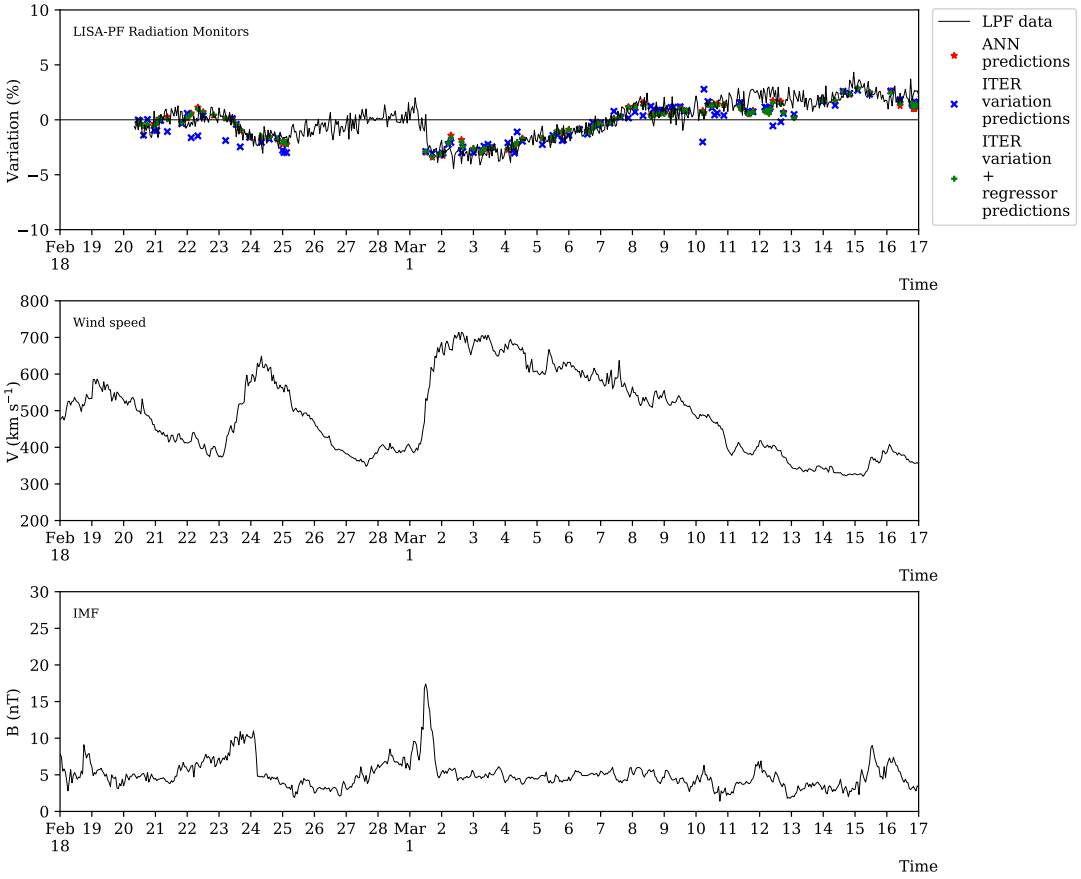



Figure 6.14: Same as Fig. 6.1 for the Bartels rotation 2504, from February 18th, 2017 through March 16th, 2017. There are missing LISA Pathfinder data during this Bartels rotation.

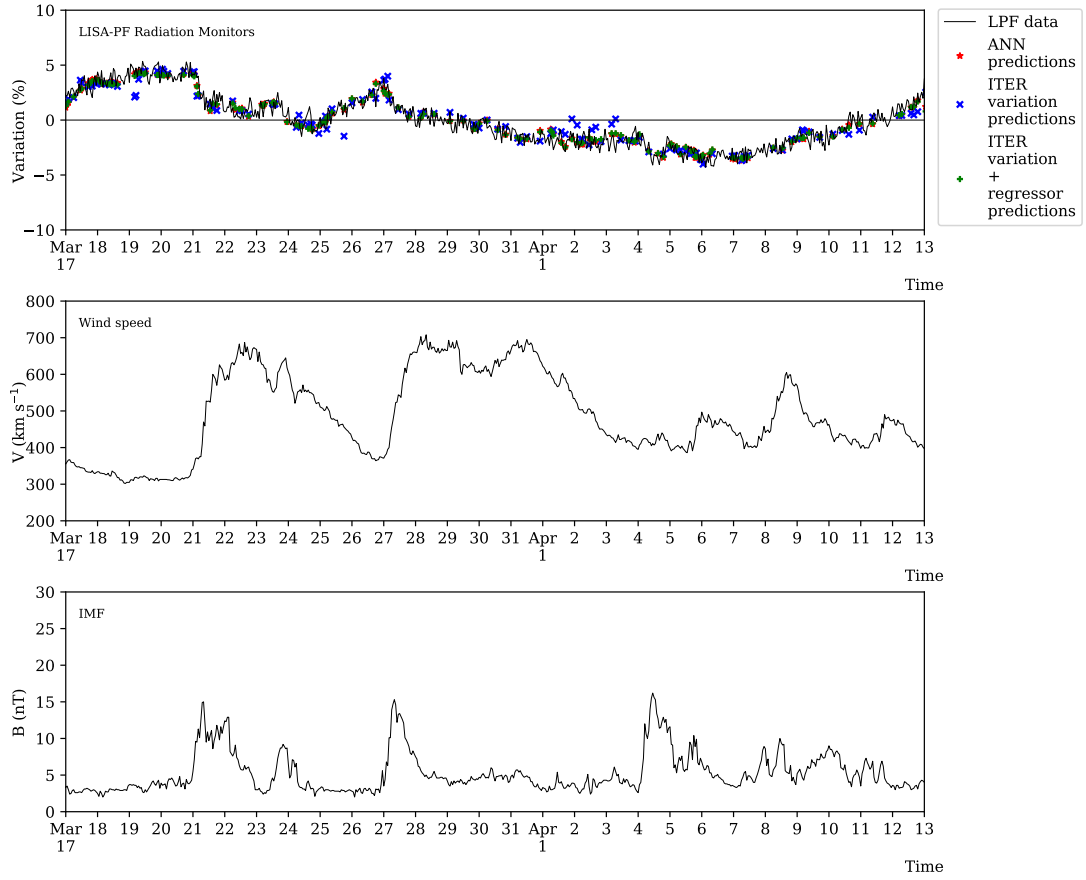



Figure 6.15: Same as Fig. 6.1 for the Bartels rotation 2505, from March 17th, 2017 through April 12th, 2017.

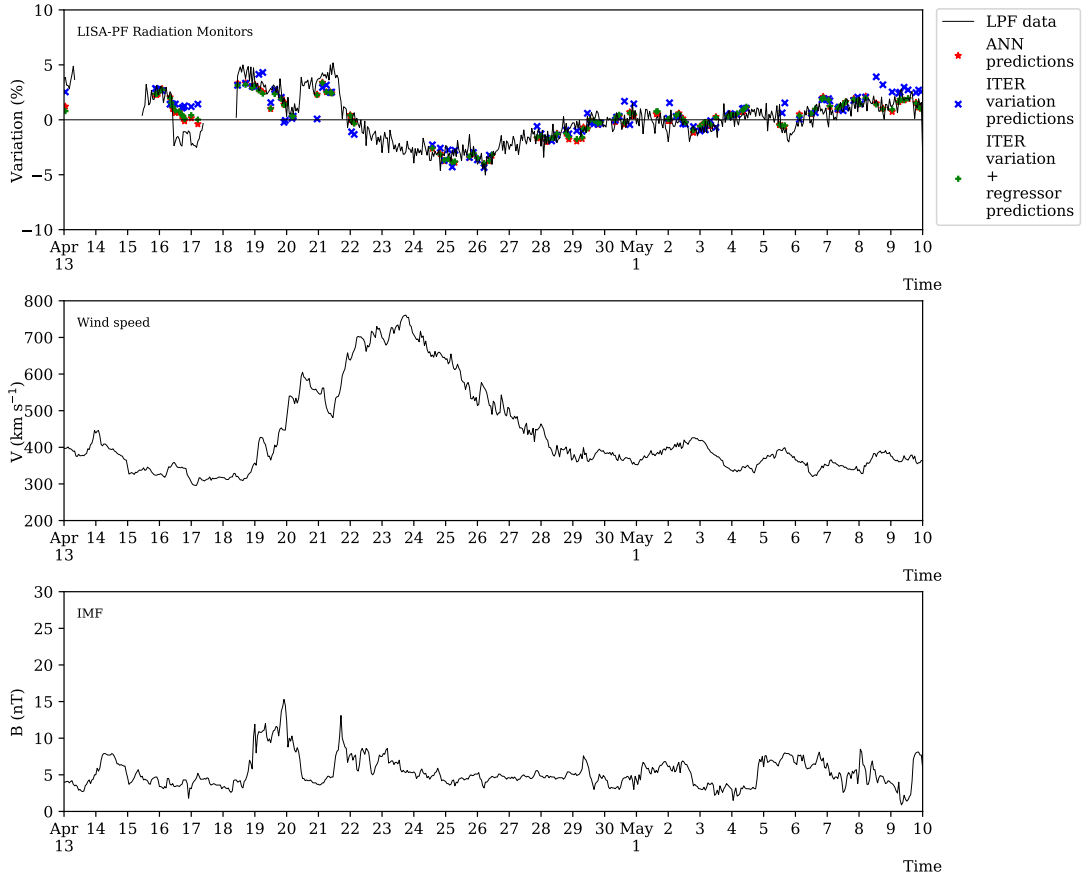



Figure 6.16: Same as Fig. 6.1 for the Bartels rotation 2506, from April 13th, 2017 through May 9th, 2017. There are missing LISA Pathfinder data during this Bartels rotation.

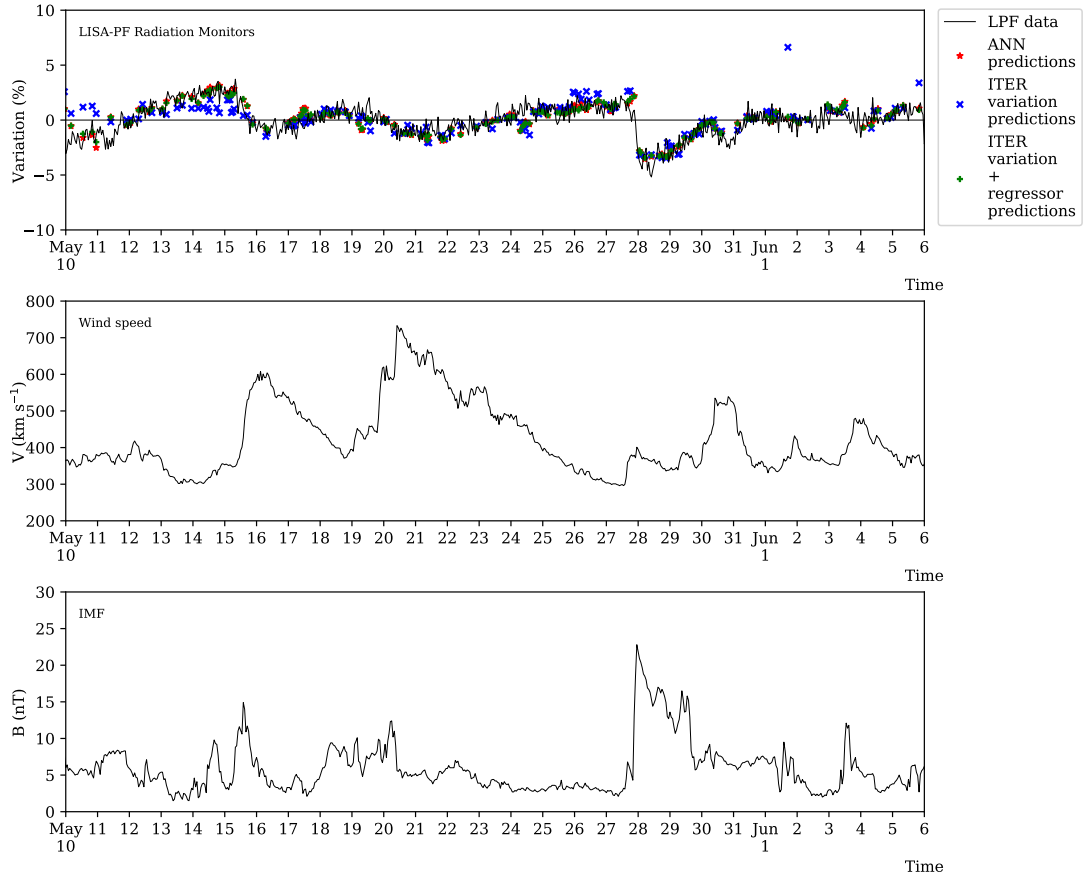



Figure 6.17: Same as Fig. 6.1 for the Bartels rotation 2507, from May 10th, 2017 through June 5th, 2017.

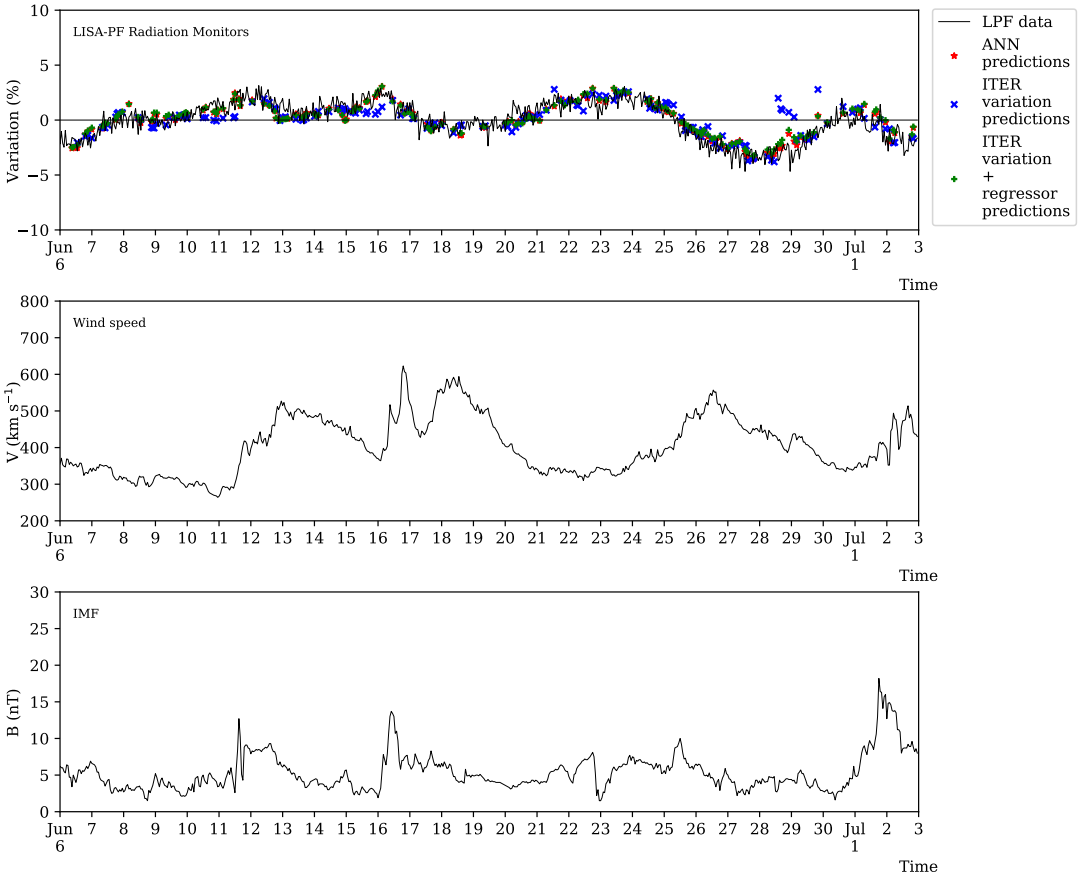



Figure 6.18: Same as Fig. 6.1 for the Bartels rotation 2508, from June 6th, 2017 through July 2nd, 2017.

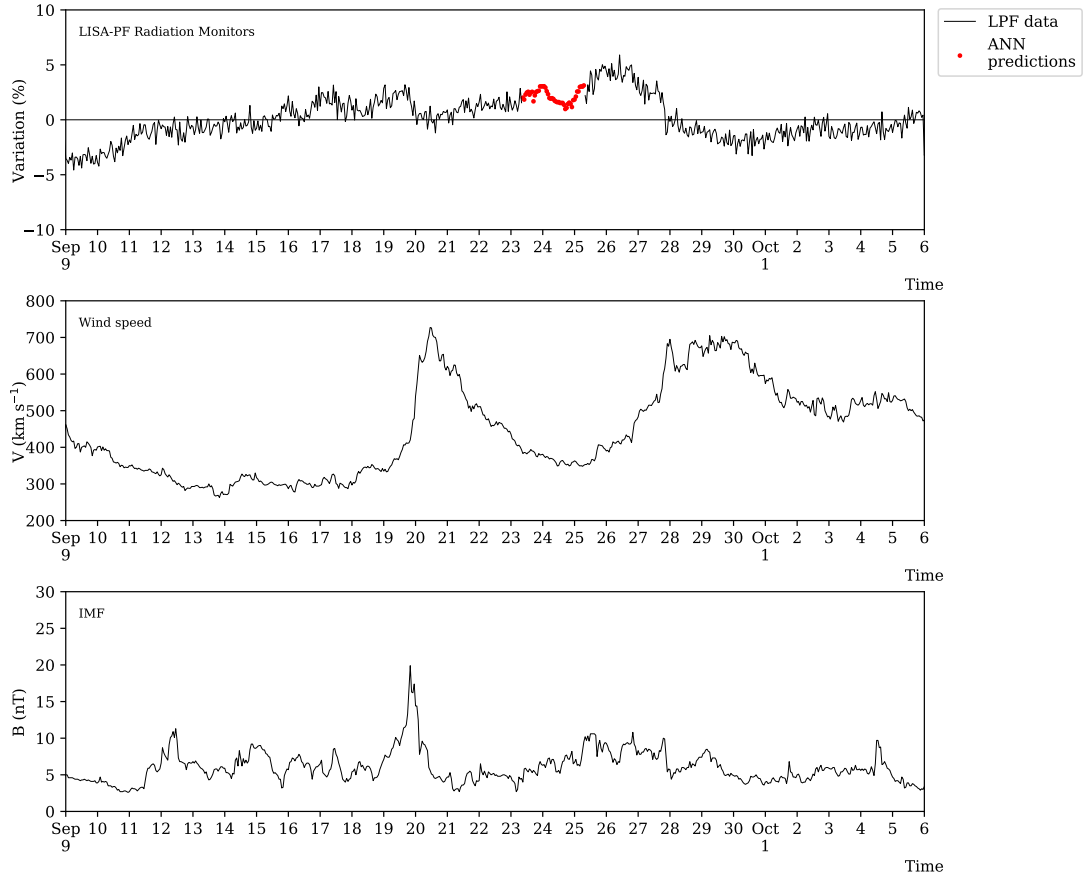



Figure 6.19: Same as Fig. 6.8 with missing data filled with the prediction of the implemented artificial neural network model (red dots in the top panel). Predictions performed by the implemented models on the test set are not reported.

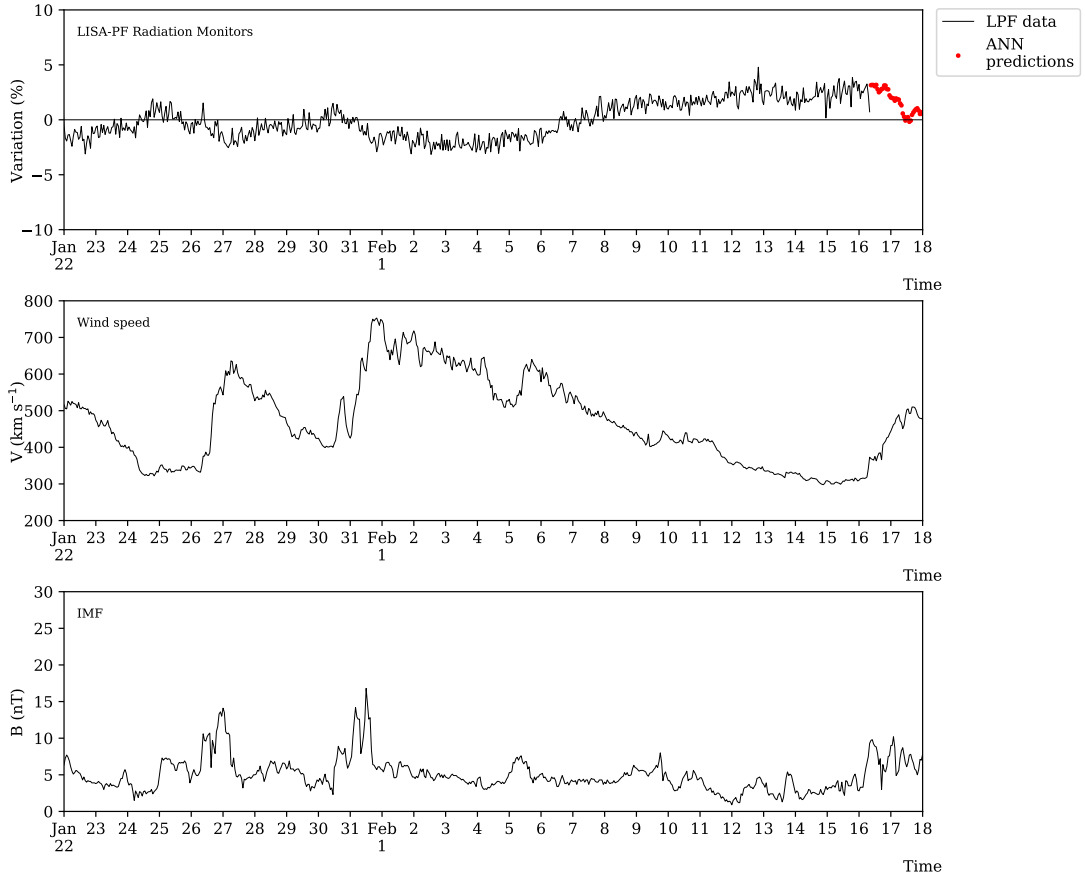



Figure 6.20: Same as Fig. 6.13 with missing data filled with the prediction of the implemented artificial neural network model (red dots in the top panel). Predictions performed by the implemented models on the test set are not reported.

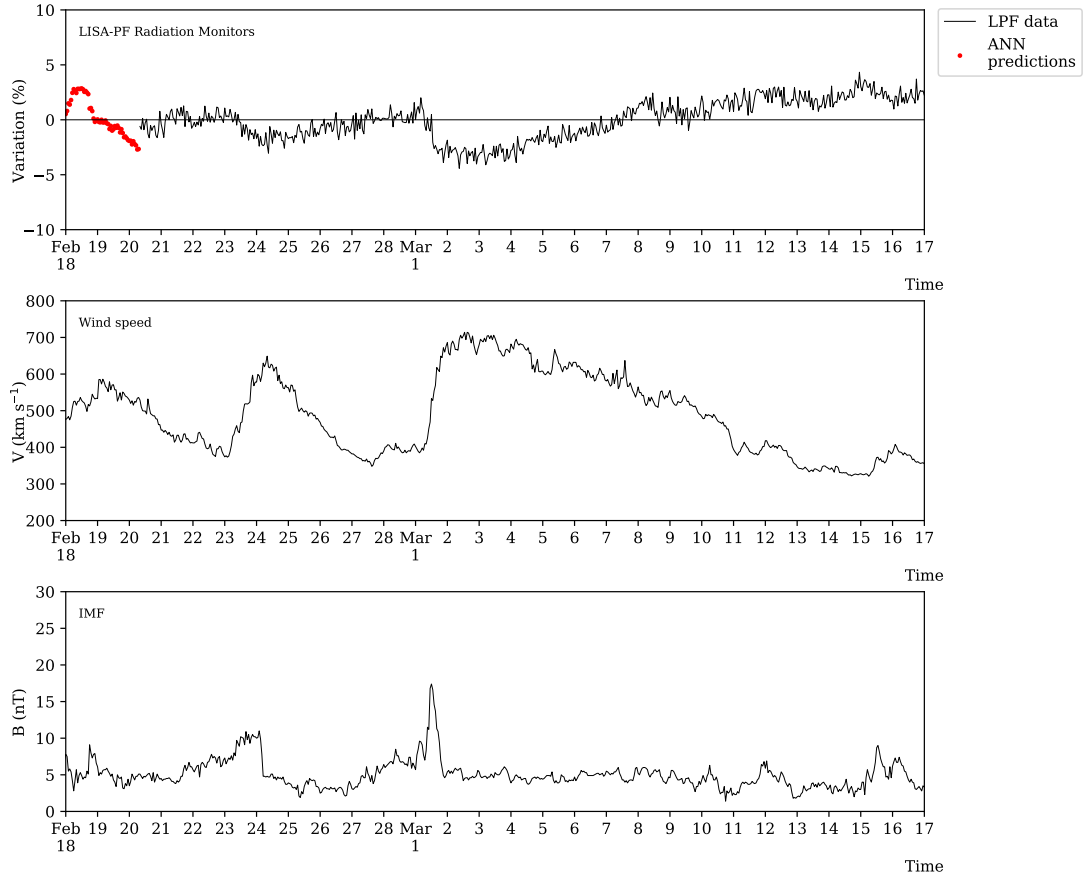



Figure 6.21: Same as Fig. 6.14 with missing data filled with the prediction of the implemented artificial neural network model (red dots in the top panel). Predictions performed by the implemented models on the test set are not reported.

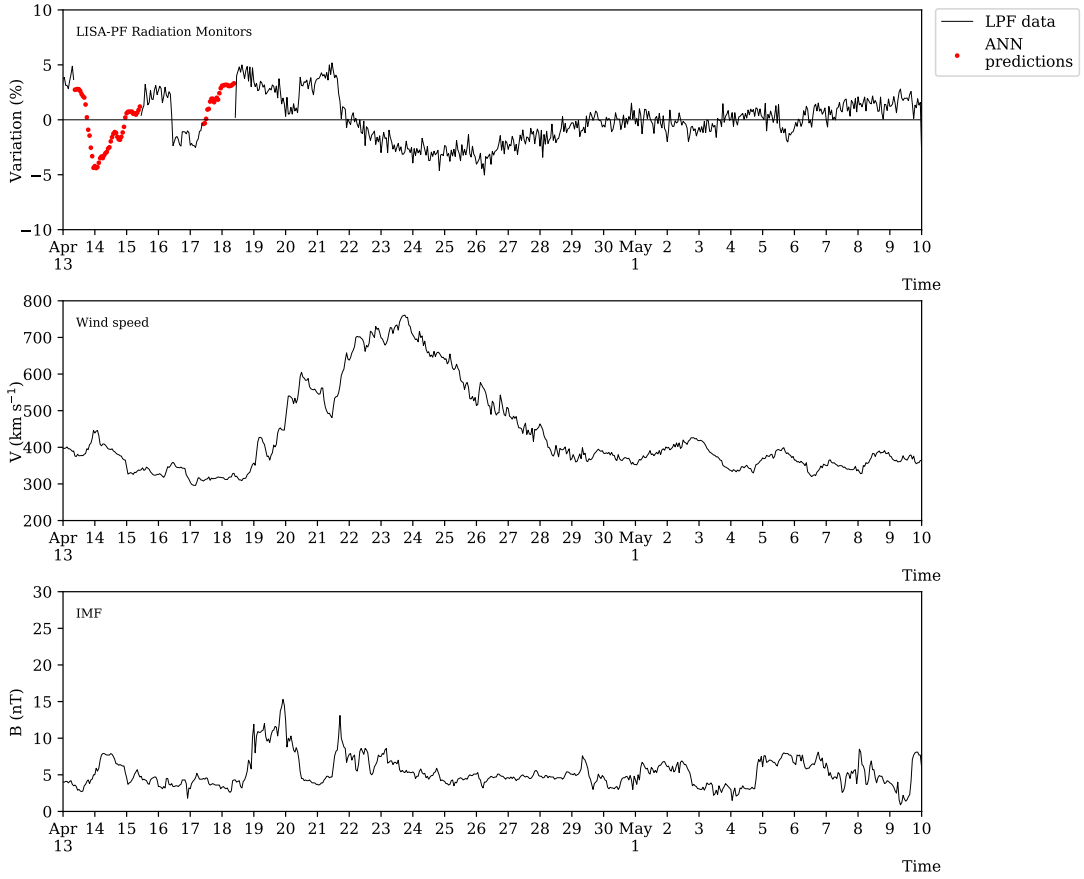



Figure 6.22: Same as Fig. 6.16 with missing data filled with the prediction of the implemented artificial neural network model (red dots in the top panel). Predictions performed by the implemented models on the test set are not reported.

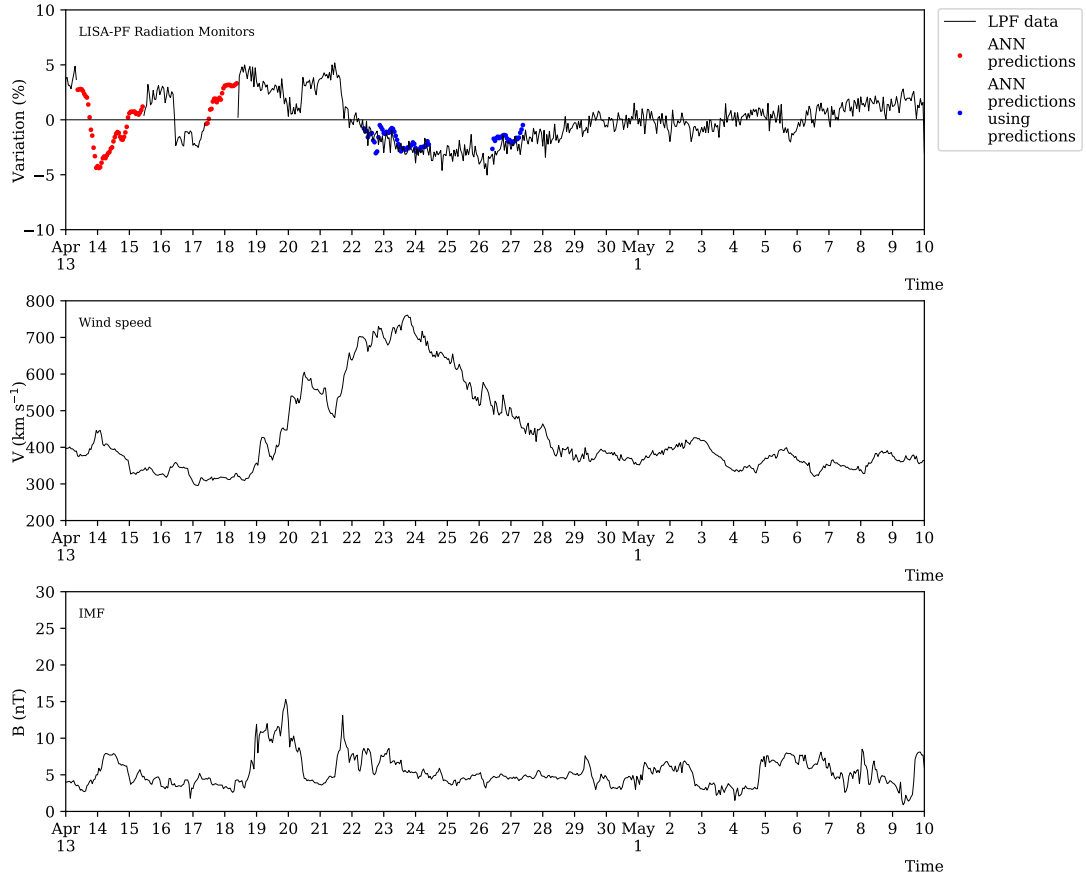



Figure 6.23: Same as Fig. 6.22 with artificial neural network predictions (blue dots in the top panel) performed by considering as input flux normalisation the preceding predictions of the same model (red dots in the top panel).

# Chapter 7

# Conclusions

In this thesis work it is shown that on board any space mission carrying instruments for interplanetary magnetic field intensity monitoring and solar wind speed measurements it is feasible to predict recurrent variations of the galactic cosmic-ray flux. Space missions can be divided into two major classes: those devoted to the measurement of cosmic rays and those for which cosmic rays constitute a limitation to the performance of on-board instruments. In this last case, it is extremely important to monitor *in situ* variations of the galactic cosmic-ray flux to study the detector efficiencies. However, if no particle detectors are present on board or the geometrical factors of those detectors are too small to allow for studying short-term galactic cosmic-ray fluctuations, it is more than recommended to benefit of models that allow for tracing the passage of interplanetary magnetic structures and solar wind high-speed streams that generate these fluctuations. This is the case, for instance, of the ESA/NASA Solar Orbiter mission launched on February 9th, 2020 from Cape Canaveral (Florida, USA). This mission is devoted to study how the Sun creates and controls the heliosphere. The spacecraft will reach a minimum distance from the Sun of 0.28 AU. The EPD/HET detector on board Solar Orbiter allows for the monitoring of the cosmic-ray flux up to 1 GeV. However, the geometrical factor of this detector does not allow to study cosmic-ray flux fluctuations with a precision of 1% down to time scales of hours as it was needed for the LISA Pathfinder mission. The development of a model for predicting short-term variations of the galactic cosmic-ray flux is the main goal of this work. This model allows to use solar wind speed and interplanetary magnetic field intensity data gathered on board the same or nearby space missions to carry out predictions on the galactic cosmic-ray flux variations with an average uncertainty smaller than 1%.

A qualitative relationship between increments of solar wind speed and interplanetary magnetic field intensity with galactic cosmic-ray flux depressions is well known in the literature. The explainable model implemented in this work allows

to extract quantitative relationships between the input and output variables in the form of linear functions. The parameters of these functions permit to learn how changes in the input variable values affect the observed cosmic-ray flux variations.

This work highlights as some of the most known knowledge extraction algorithms from neural networks solving regression tasks are not always applicable with the desired results, especially in those cases where it is not possible to extremely simplify the underlying neural network model. Both the extraction algorithms REFANN and ITER are compared. REFANN is not applicable because it is designed for artificial neural networks having a single hidden layer while the neural network model developed in this work has two hidden layers. ITER results to be no suitable for the task at hand due to the complexity of the input data set, i.e. the number of input features. Nevertheless, there is the possibility to adapt those extraction algorithms in order to apply their concepts to the problem under study. Two variants of ITER are described in this work; only one model proved to give good output predictions. The implementation of this model takes into account the main concepts of the original ITER algorithm in order to create hyper-cubes in the input feature space, where all the samples contained in each hyper-cube have a similar output value. Then a linear regressor is applied to single hyper-cubes for the knowledge extraction.

The explainable model described here uses an underlying artificial neural network with comparable prediction performance: the outputs of both models have a mean absolute error smaller than 0.75%. This result is comparable to the statistical uncertainty of the LISA Pathfinder mission data of 1%. Since the underlying neural network has a slightly better performance, it is suggested to use this model when no prediction explanations are required. Conversely, the explainable model has to be preferred when human understandable outputs are required.

It is also found that the implemented artificial neural network can be used to carry out predictions about the galactic cosmic-ray flux variations in periods during which the LISA Pathfinder data are missing. The model output exhibits a good agreement with the data immediately preceding and following in the LISA Pathfinder time series and can be used as input flux normalisation in the place of the mission missing data to perform further predictions about the cosmic-ray fluctuations in successive instants. It is shown that these predictions based on the neural network outcomes have a mean absolute error of 0.72%, showing an agreement comparable to that of the other test samples. This result is a quantitative indication of the missing data prediction goodness.

Finally, the explainable model presented in this work allows to set a quantitative relationship between the galactic cosmic-ray flux variations and the trend of solar wind speed and interplanetary magnetic field intensity. The relationship is obtainable from the described trained neural network with no losses in the predic-

tion performance. Galactic cosmic-ray flux short-term variations were observed to range from −7% through +9% during the whole LISA Pathfinder mission elapsed time. When applied to the test set samples, the explainable model is able to reproduce the data trend between −6.3% and +7.8%, due to the small number of training examples having an output value near the limits of the LISA Pathfinder data range. Nevertheless, since the observation of these very small or large values is extremely rare, the prediction performance of the model is not affected.

The minimisation of the predicting neural network model from the point of view of the network architecture as well as the reduction of the input data set dimension intended as number of input features is carried out in this work and can be further investigated and considered for future applications in space. The implementation of a different rule extraction algorithm can also be considered in the future in order to obtain different kinds of rules, such as decision trees.

The limits of the implemented model application reside in the necessity of composing input samples with past solar wind speed and interplanetary magnetic field intensity observations and with a past galactic cosmic-ray flux variation value used for normalisation. For instance, it would be impossible to make real-time predictions without real-time availability of the input variable data. In addition, the input variable observations need to be gathered *in situ* or in the proximity of the satellite taking cosmic-ray data for flux normalisation.

Since the data used in this work have a statistical uncertainty of 1% and since the implemented models provide outputs with a mean absolute error smaller than this uncertainty, the described models can be adopted to predict the galactic cosmic-ray flux variations on board space mission where particle detectors are not present or if these detectors are present but have small geometrical factors. Solar Orbiter is an example of an ongoing space mission that may benefit of the models described here. The implemented models can be put on board this mission in order to study the galactic cosmic-ray flux short-term variations when the satellite will be in the proximity of the first Lagrangian point, where orbited LISA Pathfinder. Observing recurrent galactic cosmic-ray variations is important to study recurrent geomagnetic storms. Finally, it would be possible to use the model predictions as a replacement of the cosmic-ray missing data during hardware tests carried out on board satellites.

# Bibliography


[1] Bart Baesens, Rudy Setiono, Christophe Mues, and Jan Vanthienen. Using neural network rule extraction and decision tables for credit-risk evaluation. *Management science*, 49(3):312–329, 2003.

[2] Bart Baesens, Rudy Setiono, V. De Lille, Stijn Viaene, and Jan Vanthienen. Building credit-risk evaluation expert systems using neural network rule extraction and decision tables. In *Proceedings of the 5th Pacific Asian Conference on Intelligent Systems (PACIS)*, 2001.

[3] Maria Teresinha Arns Steiner, Pedro José Steiner Neto, Nei Yoshihiro Soma, Tamio Shimizu, and J.C. Nievola. Using neural network rule extraction for credit-risk evaluation. *International Journal of Computer Science and Network Security*, 6(5):6–16, 2006.

[4] Leonardo Franco, José Luis Subirats, Ignacio Molina, Emilio Alba, and José M. Jerez. Early breast cancer prognosis prediction and rule extraction using a new constructive neural network algorithm. In *International Work-Conference on Artificial Neural Networks*, pages 1004–1011. Springer, 2007.

[5] Yoichi Hayashi, Rudy Setiono, and Katsumi Yoshida. A comparison between two neural network rule extraction techniques for the diagnosis of hepatobiliary disorders. *Artificial intelligence in Medicine*, 20(3):205–216, 2000.

[6] Guido Bologna and Christian Pellegrini. Three medical examples in neural network rule extraction. *Physica Medica*, 13:183–187, 1997.

[7] Guido Bologna. A study on rule extraction from neural networks applied to medical databases. In *The 4th European Conference on Principles and Practice of Knowledge Discovery (PKDD2000)*, 2000.

[8] Rudy Setiono, Bart Baesens, and Christophe Mues. Rule extraction from minimal neural networks for credit card screening. *International journal of neural systems*, 21(04):265–276, 2011.

[9] Alexander Hofmann, Carsten Schmitz, and Bernhard Sick. Rule extraction from neural networks for intrusion detection in computer networks. In *SMC'03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics. Conference Theme-System Security and Assurance (Cat. No. 03CH37483)*, volume 2, pages 1259–1265. IEEE, 2003.

[10] Arnulfo Azcarraga, Michael David Liu, and Rudy Setiono. Keyword extraction using backpropagation neural networks and rule extraction. In *The 2012 international joint conference on neural networks (IJCNN)*, pages 1–7. IEEE, 2012.

[11] M. Armano et al. Sub-femto-$g$ free fall for space-based gravitational wave observatories: LISA Pathfinder results. *Phys. Rev. Lett.*, 116:231101, Jun 2016.

[12] Michele Armano et al. Beyond the required LISA free-fall performance: new LISA Pathfinder results down to 20 $\mu$ Hz. *Physical review letters*, 120(6):061101, 2018.

[13] Tom M. Mitchell. Machine learning and data mining. *Communications of the ACM*, 42(11):30–36, 1999.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] John R. Koza, Forrest H. Bennett, David Andre, Martin A. Keane, and Frank Dunlap. Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on evolutionary computation*, 1(2):109–128, 1997.

[16] Christopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[17] Eneldo Loza Mencía. *Efficient Pairwise Multilabel Classification*. PhD thesis, Technische Universität, 2013.

[18] Xinchuan Zeng and Tony R. Martinez. Distribution-balanced stratified cross-validation for accuracy estimation. *Journal of Experimental & Theoretical Artificial Intelligence*, 12(1):1–12, 2000.

[19] Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and regression trees*. CRC Press, 1984.

[20] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[21] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*, 2018.

[22] H. Jabbar and Rafiqul Zaman Khan. Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study). *Computer Science, Communication and Instrumentation Devices*, pages 163–172, 2015.

[23] Helmut Lütkepohl. *New introduction to multiple time series analysis*. Springer Science & Business Media, 2005.

[24] Jonathan D. Cryer and Kung-Sik Chan. *Time series analysis: with applications in R*. Springer Science & Business Media, 2008.

[25] Daniel Klerfors and Terry L. Huston. Artificial neural networks. *St. Louis University, St. Louis, Mo*, 1998.

[26] Mark A. Gluck and Gordon H. Bower. Evaluating an adaptive network model of human learning. *Journal of memory and Language*, 27(2):166–195, 1988.

[27] Richard Granger, J. Ambrose-Ingerson, Ursula Staubli, and Gary Lynch. Memorial operation of multiple interacting simulated brain structures. *Neuroscience and connectionist theory*, pages 95–129, 1990.

[28] Dave Anderson and George McNeill. Artificial neural networks technology. *Kaman Sciences Corporation*, 258(6):1–83, 1992.

[29] M. Hiew and G. Green. Beyond statistics. a forecasting system that learns. In *The Forum*, volume 5, pages 1–6, 1992.

[30] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[31] David P. Friedman and Sue Rusche. *False messengers: How addictive drugs change the brain*. CRC Press, 2003.

[32] National Institutes of Health and National Institute on Drug Abuse. Brain's response to drugs: Mind over matter teacher's guide, revision. `http://www.sarasquest.org`, 2000. Accessed: 2020-07-19.

[33] Eric R. Kandel et al. Nerve cells and behavior. *Principles of neural science*, 2:24–25, 1991.

[34] National Institutes of Health and National Institute on Drug Abuse. Neurons, brain chemistry, and neurotransmission. `https://science.education.nih.gov/supplements/webversions/BrainAddiction/guide/lesson2-1.html`. Accessed: 2020-07-19.

[35] Tarik Rashid. *A Novel Recurrent Neural Network Model: A Case Study in Energy Load Forecasting.* PhD thesis, University of Kurdistan Hewlêr, 08 2006.

[36] Charu C. Aggarwal et al. *Neural networks and deep learning.* Springer, 2018.

[37] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[38] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[39] Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

[40] Yann Le Cun and Françoise Fogelman-Soulié. Modèles connexionnistes de l'apprentissage. *Intellectica*, 2(1):114–143, 1987.

[41] Alan S. Lapedes and Robert M. Farber. How neural nets work. In *Neural information processing systems*, pages 442–456, 1988.

[42] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[43] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural networks*, 2(3):183–192, 1989.

[44] Amanda C. Mathias and Paulo C. Rech. Hopfield neural network: The hyperbolic tangent and the piecewise-linear activation functions. *Neural Networks*, 34:42–45, 2012.

[45] George A. Anastassiou. Univariate hyperbolic tangent neural network approximation. *Mathematical and Computer Modelling*, 53(5-6):1111–1132, 2011.

[46] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[47] Kazuyuki Hara, Daisuke Saito, and Hayaru Shouno. Analysis of function of rectified linear unit used in deep learning. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[48] George E. Dahl, Tara N. Sainath, and Geoffrey E. Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013.

[49] Forest Agostinelli, Matthew Hoffman, Peter Sadowski, and Pierre Baldi. Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*, 2014.

[50] David L. Elliott. A better activation function for artificial neural networks. Technical report, University of Maryland, 1993.

[51] Sagar Sharma. Activation functions in neural networks. *Towards Data Science*, 6, 2017.

[52] Bekir Karlik and Ahmet Vehbi Olgac. Performance analysis of various activation functions in generalized MLP architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[53] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[54] Takashi Onoda. Neural network information criterion for the optimal number of hidden units. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 1, pages 275–280. IEEE, 1995.

[55] Mario Gutierrez, Jennifer Wang, and Robert Grondin. Estimating hidden unit number for two-layer perceptrons. In *IJCNN International Joint Conference on Neural Networks*, pages 677–681. Publ by IEEE, 1989.

[56] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multilayer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[57] Terrence L. Fine. *Feedforward neural network methodology*. Springer Science & Business Media, 2006.

[58] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[59] D. Randall Wilson and Tony R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural networks*, 16(10):1429–1451, 2003.

[60] Warren S. Sarle. Stopped training and other remedies for overfitting. *Computing science and statistics*, pages 352–360, 1996.

[61] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[62] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019.

[63] Ehud D. Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks*, 1(2):239–242, 1990.

[64] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[65] Philippe Thomas and Marie-Christine Suhner. A new multilayer perceptron pruning algorithm for classification and regression applications. *Neural Processing Letters*, 42(2):437–458, 2015.

[66] Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

[67] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)*, 51(5):1–42, 2018.

[68] Alex A. Freitas. Comprehensible classification models: a position paper. *ACM SIGKDD explorations newsletter*, 15(1):1–10, 2014.

[69] Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems*, 51(1):141–154, 2011.

[70] Patrick M. Murphy and Michael J. Pazzani. Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees. In *Machine Learning Proceedings 1991*, pages 183–187. Elsevier, 1991.

[71] J. Ross Quinlan. C4.5: Programming for machine learning. *Morgan Kauffmann*, 38:48, 1993.

[72] J. Ross Quinlan. Simplifying decision trees. *International Journal of Human-Computer Studies*, 51(2):497–510, 1999.

[73] Lior Rokach and Oded Z. Maimon. *Data mining with decision trees: theory and applications*, volume 69. World scientific, 2008.

[74] Andreas Henelius, Kai Puolamäki, Henrik Boström, Lars Asker, and Panagiotis Papapetrou. A peek into the black box: exploring classifiers by randomization. *Data mining and knowledge discovery*, 28(5-6):1503–1529, 2014.

[75] Yin Lou, Rich Caruana, Johannes Gehrke, and Giles Hooker. Accurate intelligible models with pairwise interactions. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 623–631, 2013.

[76] Kelvin Xu et al. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning, Lille, France*, volume 37, 2015.

[77] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.

[78] David Baehrens, Timon Schroeter, Stefan Harmeling, Motoaki Kawanabe, Katja Hansen, and Klaus-Robert Müller. How to explain individual classification decisions. *The Journal of Machine Learning Research*, 11:1803–1831, 2010.

[79] Julian D. Olden and Donald A. Jackson. Illuminating the "black box": a randomization approach for understanding variable contributions in artificial neural networks. *Ecological modelling*, 154(1-2):135–150, 2002.

[80] Julius Adebayo and Lalana Kagal. Iterative orthogonal feature projection for diagnosing bias in black-box models. *arXiv preprint arXiv:1611.04967*, 2016.

[81] Alex Goldstein, Adam Kapelner, Justin Bleich, and Emil Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1):44–65, 2015.

[82] Jacob Bien and Robert Tibshirani. Prototype selection for interpretable classification. *The Annals of Applied Statistics*, pages 2403–2424, 2011.

[83] Been Kim, Cynthia Rudin, and Julie A. Shah. The bayesian case model: A generative approach for case-based reasoning and prototype classification. In *Advances in neural information processing systems*, pages 1952–1960, 2014.

[84] Ravid Shwartz-Ziv and Naftali Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[85] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[86] Robert Andrews, Joachim Diederich, and Alan B Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-based systems*, 8(6):373–389, 1995.

[87] Geoffrey G. Towell and Jude W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine learning*, 13(1):71–101, 1993.

[88] Christian W. Omlin and C. Lee Giles. Extraction of rules from discrete-time recurrent neural networks. *Neural networks*, 9(1):41–52, 1996.

[89] Mark W. Craven and Jude W. Shavlik. Using sampling and queries to extract rules from trained neural networks. In *Machine learning proceedings 1994*, pages 37–45. Elsevier, 1994.

[90] Mark W. Craven and Jude W. Shavlik. Extracting tree-structured representations of trained networks. In *Advances in neural information processing systems*, pages 24–30, 1996.

[91] Rudy Setiono, Wee Kheng Leow, and Jacek M. Zurada. Extraction of rules from artificial neural networks for nonlinear regression. *IEEE transactions on neural networks*, 13(3):564–577, 2002.

[92] Johan Huysmans, Bart Baesens, and Jan Vanthienen. Iter: an algorithm for predictive regression rule extraction. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 270–279. Springer, 2006.

[93] Pau Amaro-Seoane et al. Laser interferometer space antenna. *arXiv preprint arXiv:1702.00786*, 2017.

[94] D.N.A. Shaul et al. Solar And Cosmic Ray Physics And The Space Environment: Studies For And With LISA. In *AIP Conference Proceedings*, volume 873, pages 172–178. AIP, 2006.

[95] H.M. Araújo, P. Wass, D. Shaul, G. Rochester, and T.J. Sumner. Detailed calculation of test-mass charging in the LISA mission. *Astroparticle Physics*, 22(5-6):451–469, jan 2005.

[96] Catia Grimani et al. LISA test-mass charging process due to cosmic-ray nuclei and electrons. *Classical and Quantum Gravity*, 22(10):S327–S332, may 2005.

[97] Catia Grimani, Michele Fabi, Alberto Lobo, Ignacio Mateos, and Daniele Telloni. LISA Pathfinder test-mass charging during galactic cosmic-ray flux short-term variations. *Classical and Quantum Gravity*, 32(3):035001, feb 2015.

[98] M. Armano et al. Charge-induced force noise on free-falling test masses: results from LISA Pathfinder. *Physical Review Letters*, 118(17):171101, apr 2017.

[99] Priscilla Cañizares et al. The lisa pathfinder dmu and radiation monitor. *Classical and quantum gravity*, 28(9):094004, 2011.

[100] Rikho Nymmik. Sep event distribution function as inferred from spaceborne measurements and lunar rock isotopic data. In *26th International Cosmic Ray Conference (ICRC26), Volume 6*, volume 6, page 268, 1999.

[101] Rikho Nymmik. Relationships among solar activity sep occurrence frequency, and solar energetic particle event distribution function. In *26th International Cosmic Ray Conference (ICRC26), Volume 6*, volume 6, page 280, 1999.

[102] M. Storini, E.W. Cliver, M. Laurenza, and Catia Grimani. Forecasting solar energetic particle events. *COST 724 final report*, page 63, 2008.

[103] Catia Grimani et al. Lisa-pf radiation monitor performance during the evolution of sep events for the monitoring of test-mass charging. *Classical and Quantum Gravity*, 31(4):045018, 2014.

[104] T.K. Gaisser et al. Cosmic rays and particle physics, cambridge, uk: Univ, 1990.

[105] P. Papini, Catia Grimani, and S.A. Stephens. An estimate of the secondary-proton spectrum at small atmospheric depths. *Il Nuovo Cimento C*, 19(3):367–387, 1996.

[106] Catia Grimani, M. Fabi, N. Finetti, and D. Tombolato. Parameterization of galactic cosmic-ray fluxes during opposite polarity solar cycles for future space missions. In *Proceedings of the 30th International Cosmic Ray Conference, Merida, Mexico*, pages 3–11, 2007.

[107] L.J. Gleeson and W.I. Axford. Solar modulation of galactic cosmic rays. *The Astrophysical Journal*, 154:1011, 1968.

[108] Catia Grimani, M. Fabi, N. Finetti, and D. Tombolato. The role of interplanetary electrons at the time of the lisa missions. *Classical and Quantum Gravity*, 26(21):215004, 2009.

[109] I. Usoskin. The monthly and annual values of the modulation parameter reconstructed from the ground based cosmic ray data. `http://cosmicrays.oulu.fi/phi/Phi_mon.txt`. Accessed: 2020-08-05.

[110] Yoshiaki Shikaze et al. Measurements of 0.2–20 gev/n cosmic-ray proton and helium spectra from 1997 through 2002 with the bess spectrometer. *Astroparticle Physics*, 28(1):154–167, 2007.

[111] Neeharika Thakur. Observed transient variations in cosmic ray proton fluxes from bess-polar i and their physical interpretations. In *ICRC*, volume 11, page 220, 2011.

[112] Koh Abe et al. Measurements of cosmic-ray proton and helium spectra from the bess-polar long-duration balloon flights over antarctica. *The Astrophysical Journal*, 822(2):65, 2016.

[113] M. Storini, N. Iucci, and S. Pase. North-south anisotropy during the quasi-stationary modulation of galactic cosmic rays. *Il Nuovo Cimento C*, 15(5):527–538, 1992.

[114] I. Sabbah and K. Kudela. Third harmonic of the 27 day periodicity of galactic cosmic rays: Coupling with interplanetary parameters. *Journal of Geophysical Research: Space Physics*, 116(A4), 2011.

[115] Ian G. Richardson, G. Wibberenz, and H.V. Cane. The relationship between recurring cosmic ray depressions and corotating solar wind streams at $\leq 1$ AU: IMP 8 and Helios 1 and 2 anticoincidence guard rate observations. *Journal of Geophysical Research: Space Physics*, 101(A6):13483–13496, 1996.

[116] Ian G. Richardson. Energetic particles and corotating interaction regions in the solar wind. *Space Science Reviews*, 111(3-4):267–376, 2004.

[117] Manuela Temmer, Bojan Vršnak, and Astrid M. Veronig. Periodic appearance of coronal holes and the related variation of solar wind parameters. *Solar Physics*, 241(2):371–383, 2007.

[118] I. Sabbah. The role of interplanetary magnetic field and solar wind in modulating both galactic cosmic rays and geomagnetic activity. *Geophysical research letters*, 27(13):1823–1826, 2000.

[119] I. Sabbah. Twenty-seven-day variation of galactic cosmic rays. *Solar Physics*, 245(1):207–217, 2007.

[120] Martin G. Mlynczak et al. Solar-terrestrial coupling evidenced by periodic behavior in geomagnetic indexes and the infrared energy budget of the thermosphere. *Geophysical Research Letters*, 35(5), 2008.

[121] Agnieszka Gil and Michael V. Alania. Rigidity spectrum of the 27-day variation of the galactic cosmic ray intensity in different epochs of solar activity. *Advances in space research*, 45(3):429–436, 2010.

[122] J.W. Bieber, E. Eroshenko, P. Evenson, E.O. Flückiger, and R. Kallenbach. *Cosmic rays and earth*, volume 10. Springer Science & Business Media, 2000.

[123] O. Adriani et al. Observations of the 2006 december 13 and 14 solar particle events in the 80 MeV $n^{-1}$ - 3 GeV $n^{-1}$ range from space with the pamela detector. *The Astrophysical Journal*, 742(2):102, 2011.

[124] A.ne.mo.s. web interface to the neutron monitor database. `http://cosray.phys.uoa.gr/index.php/esa-neutron-monitor-service/multi-station-neutron-monitor-data`. Accessed: 2020-08-06.

[125] R.P. Kane. Severe geomagnetic storms and forbush decreases: interplanetary relationships reexamined. In *Annales Geophysicae*, volume 28, pages 479–489. Copernicus GmbH, 2010.

[126] K.M. Alanko, I.G. Usoskin, Kalevi Mursula, and G.A. Kovaltsov. Effective energy of neutron monitors. In *International Cosmic Ray Conference*, volume 7, page 3901, 2003.

[127] M.E. Wiedenbeck et al. Time dependence of solar modulation throughout solar cycle 23 as inferred from ace measurements of cosmic-ray energy spectra. In *Proceedings of the 31st International Cosmic Ray Conference, Łódź, Poland*, 2009.

[128] Y.P. Singh et al. Study of short-term modulation of galactic cosmic rays: A new approach. In *Proceedings of the ILWS Workshop. Goa, India*, page 182, 2006.

[129] C. Grimani, A. Cesarini, M. Fabi, F. Sabbatini, D. Telloni, and M. Villani. Recurrent galactic cosmic-ray flux modulation in L1 and geomagnetic activity during the declining phase of the solar cycle 24. *The Astrophysical Journal*, 904(1), 2020.