UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master Degree in Physics of Data

Master Thesis

Spatialization on grid of meteorological variables with machine learning methods

Thesis supervisor

Prof. Marco Zanetti

Thesis co-supervisor

Dr. Andrea Chini

Candidate

Federico Bottaro

Academic Year 2020/2021

Abstract

In this thesis we will develop some machine learning methods in order to interpolate meteorological information and distribute the knowledge in a grid over Italy. The variables analyzed in this work are the minimum and the maximum temperature and the rain, all at the ground level. To perform the spatialization of these variables we use multi linear regressions with a machine learning approach, Boost Decision Trees and Neural Networks and we compare the results obtained. A common procedure of data preprocessing with this type of variables is also widely covered. After the comparison we show some application of these methods in order to produce operative grids over Italy.

Contents

List of Figures

Chapter 1

Introduction

The arising of machine learning has had a significant impact in the meteorological field. The ability to predict unknown data is an intrinsic feature of machine learning algorithms and since in meteorology the prediction of data is a fundamental building block, machine learning is a tool of great interest. The first thing that comes to mind when thinking about the application of this type of algorithms in meteorology is the forecasting of weather conditions for the next days or weeks. Certainly, the forecast of meteorological variables is a very broad research front where tools such as neural networks may be used [1] but it is not the only field of this discipline where predictive algorithms can be employed. This type of technique may be applied also to historical data and the purpose of this work focuses on this category.
Historical data, as the name suggests, contains information regarding time in the past. This type of information finds applicability not only in meteorology but also in fields such as agriculture, insurance, public and private transport ... In this work the focus will be on one of the most common scenarios: spatial interpolation of information. With spatial interpolation we mean the process that is able to generalize information that, at first glance, appears as scattered points in a plot to information that simulates a surface distributed uniformly in the considered area. In other words, spatial interpolation techniques allow the estimation of a variable at locations where it has not been recorded using observation of that variable (or other ones correlated with that variable) made at other sites [2]. In meteorology one of the most reliable means of information is meteorological stations. These instruments are able to measure the most common meteorological variables directly on site so the response of the various sensor is sparse over the territory. It happens very often, however, that information is needed at locations where there are no stations or at least they are far enough from the considered point. In this scenario, different solutions can be considered.
One of those approaches may be to take values of the interested variables directly from the nearest station. In this case the final product will be a patchy map that will appear denser and narrower where there are more stations. One of the problems with this approach can be found in the heterogeneity of the networks of stations that cover Italy that can leads to conflicts or misunderstanding.
For example, let's consider a point (P) between two stations which we will call A and B. Suppose we want to know the maximum temperature in P and suppose that this point is 2 km away from A and 4 km away from B. Now the two stations report a maximum temperature of respectively 19° and 19.4°; one possible solution to our problem is to assign a maximum temperature to P of 19.2° and we are probably not making a big mistake. But what could we conclude if the two stations report values like 19° and 14°? Is still the mean of the values reported by the two nearest stations a good solution? Can a mean weighted on the distance be better? And what about the altitude?
In this work, we are going to explore this field for what concerns some typical variables in meteorology: minimum and maximum temperature and rain. To answer the questions above we try to extract some relations not only with the information on neighbors (when we talk about neighbors we mean the station closest to the considered point) but also with terrain variables such as altitude, ground type, curvature... In order to obtain these variables, we manipulate different Digital Elevation Mod-

els (DEMs) as explained in section 2.2 or other datasets containing morphological and geographical information.
In general, if we talk about machine learning is hard to not take into consideration the concept of big data. To have algorithms that work properly in this field in fact there is a need to work with huge amounts of data but if on one hand work with large dataset is quite mandatory, on the other hand we have to take into consideration the quality of that data. In sight of this, a large description of the data used is presented in section 2.2; in the same section we can find information on the preprocessing techniques that were exploited to clear the initial dataset and select what we consider *good data*. Once that the cleaning process is done, it is possible to develop different algorithms to reach our main goals. In this chapter we are going to present from a theoretical point of view the main algorithm that we use in our work: the Neural Networks. Since there are also different kinds of machine learning models that can be trained for this task, for comparison we will also present some results retrieved from multi-linear regressor and from boosted decision trees and so also a brief theoretical introduction to these will be presented in this chapter. We are going to delve a little bit into the theory behind Neural Networks because as we will see the core of the work is on these models. In chapter 3 we explain the procedure used to train the algorithms, the input that we use and the strategy to select the best parameters of the various models. In chapter 4 we will present the results obtained from the training of these learners and the grids over Italy built that are the final product of our work and the operative one. To conclude the document we discuss the results obtained in the last section of chapter 4. We will see also the comparison of the performances and we try to extract important information on this comparison. In addition, some future implementation proposals will be exposed.

1.1 Learning Paradigm

The impressive development in recent years of applications related to the world of machine learning have led to have a pool of algorithms that can be adapted to the most diverse needs. The machine learning approach is seeing application in the most varied fields and data scientists are going to explore new frontiers of knowledge thanks to it. To get a gentle introduction to this work we have to explain the basis of machine learning (ML) starting from the main question: what is machine learning?
Machine learning is a sub-field of artificial intelligence which is the collection of programs capable of reasoning and make predictions/assumptions like a human being. The goal of machine learning is instead to develop algorithms capable of learning from data automatically without programming them explicitly. With this scope we have to setup the environment so that there are the conditions to learn. For example, how do we use weather data of a given point and data from nearest stations as in the examples presented above to predict what temperature would be observed if we change setup? In addition, methods from ML tend to be applied to complex high-dimensional problems that perfectly fit in the meteorological field. Mathematically speaking we choose some observable quantity x of the system we are studying that is related to some parameters Θ. Now, we perform an experiment to obtain a dataset X and use these data to fit the model. Fitting the model involves finding a set of parameters Θ_{ML} that provide the best explanation for X [3]. In our specific case we are going to test three principal tools: multi-linear regressors, boosted decision trees and neural networks with more attention to the latter.
Before talking about the specific of every single model, we want to give an overview of which paradigm of learning we are going to follows in every application: supervised learning. Abstractly we can describe supervised learning as the scenario in which we "use experience to gain expertise" [4]. Suppose to have a framework where the minimum temperature is described as function of other variables such as altitude, longitude and latitude. The experience in this example is the knowledge of the minimum temperature related to other variables in our training sample; then once that we gain expertise we can apply it in a new dataset, with unknown temperatures in order to make some prediction. So the fundamental ingredient in the supervised paradigm is the presence of the "label". The algorithm aims to extract fundamental connection between the so called "independent" variables and the "target" variable. It is important to stress that our model has to learn from the practice and does not have to memorize from the practice. This subtle difference actually intrinsically hides the subdivision between

an algorithm that can be defined as good and a bad one. If we want to translate this concept in the technical field we have to require to our models to not overfit the data and to learn enough to not underfit the data. Figure 1.1 is a graphical representation of the concept of underfit the data, properly generalize the information during learning and overfit the data respectively.

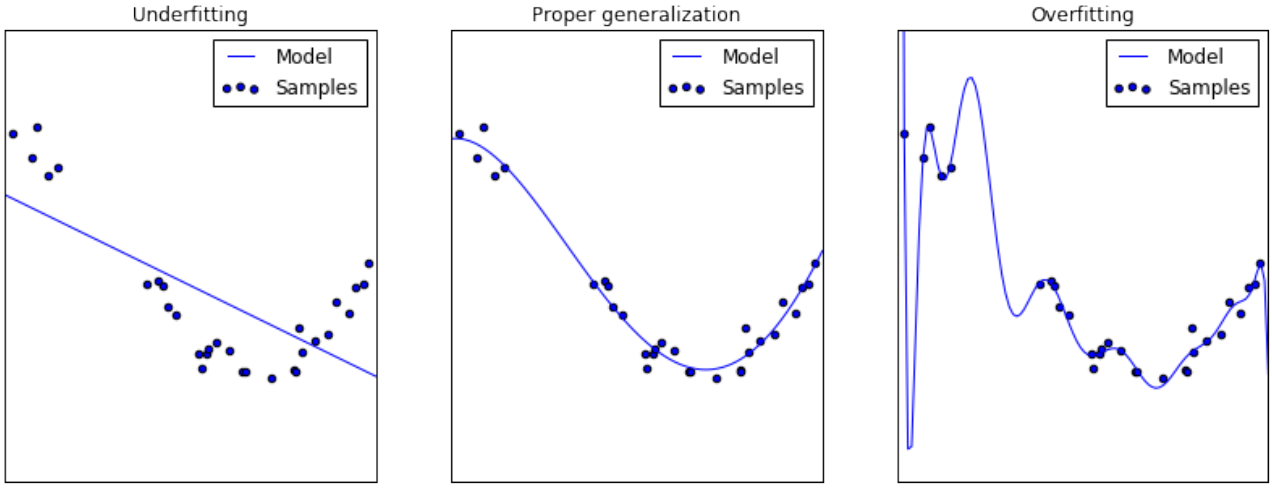


Figure 1.1: Representation of a model that underfit the data, properly generalize the information learned or overfit the data.

To overcome the problem of overfitting/underfitting from a practical point of view we have to better explain how the learning procedure works. Without delving into the detail of the optimization, to evaluate the performance of a model we can use a cost function, namely the loss function. There are several types of available loss functions, each application can require a specific loss; for example there are losses for binary classification and others related to regression task and so on. The common goal of the learning is to minimize that loss function to reach an high accuracy on the prediction/classification. We have that in general a decreasing value of the loss function leads to good performances without underfitting. For what concerns the overfitting we can evaluate the loss function of the training set (the data used to learn) but also of a new set of data, called test set, never seen during the training procedure. If we evaluate the loss function for the test set and for the training set during the learning, can happen that at a given point the loss of the training decrease while the loss of the test stops to decrease or start to increase. This is a typical behavior if we are overfitting and so we need to stop the learning and change strategy. On the other hand if we have that the training loss decrease during training and the loss function of the test follows the same behavior, the training is proceeding correctly and we are avoiding overfitting. Obviously this is only a brief introduction to the concept of over/under-fitting and literature provides a great variety of tools and techniques to handle this problem such ad regularization, dropout and others. One possible question that can arise in this operation is: when can i stop the learning procedure? Which value of loss function can I set as a target? Certainly some choices of this type depend on the field of application of our algorithms. For example if we are developing a classifier to distinguish dogs from cats we probably can accept a less restrictive value of the loss function with respect to an algorithm that has to make decision on the self-drive car field. Another strategy to reduce the overfitting is to set a number of iterations before stop the learners if the test loss does not decrease. As we will see, when we train our neural networks we follow this second strategy called early stopping. Once that the concept of supervised learning is clear, we can move to the specific algorithms.

1.2 Multi linear regressor

One of the simplest algorithms that falls within the macro-category of supervised learning is the multi linear regressor. As we can understand from the name, this algorithm is part of a class of methods intended for regression in which the target value is expected to be a linear combination of the features

(independent variables). In mathematical notation, if $\hat{y}$ is the predicted value:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \cdots + w_p x_p \tag{1.1}$$

where w are the coefficients of the regression and x are the independent variables. The linear regression fits a linear model to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation. Namely the coefficients are estimated based on the ordinary least square optimizer [5] :

$$\min_w ||y - Xw||^2 \tag{1.2}$$

These models are developed in Python using the library *"scikit-learn"* which provides different tools in the machine learning field. Looking at the algorithms provided by the library, it can be seen that there are methods based on non-linear regressions that may work better in our case for the estimation of complex relations between variables such as temperature, rain ... The choice of this particular regressor is not based on a "direct" use of it. In fact the idea behind this choice is to be able to compare a non linear model such as neural networks with a linear one. As we will see in the evaluation of the temperature, an important aspect of the variables can be understood from this comparison. We therefore have that the multilinear regressor is useful if we compare it with other (non linear) algorithms.

1.3 Boosted decision tree

The second model that we introduce is the class of regressors obtained with some decision trees. As for the multilinear regressor we are working with the paradigm of supervised learning. Before moving in detail to the regressor used, let introduce what an ensemble of trees is with an example. The tree ensemble model consists of a set of classification and regression trees (CART). Here's a simple example of a CART that classifies whether someone will like a hypothetical computer game X.

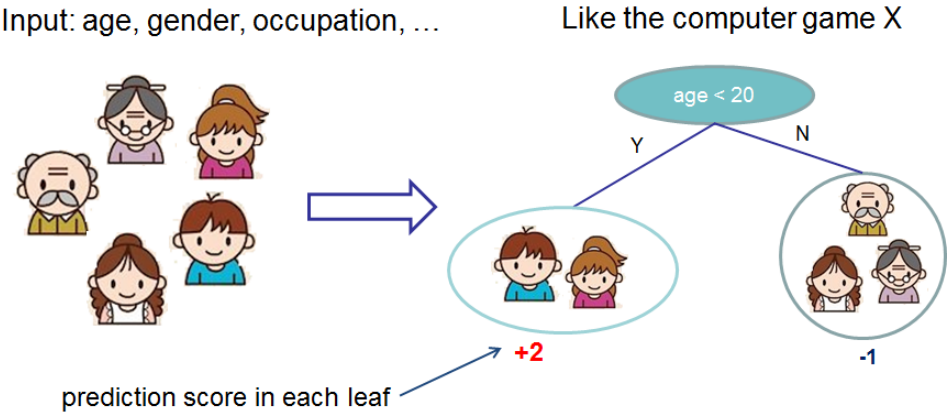


Figure 1.2: Example of a classification with a tree

Usually, a single tree is not strong enough to be used in practice. What is actually used is the ensemble model, which sums the prediction of multiple trees together. Figure 1.3 show an example of combination of two decision trees.

The prediction scores of each individual tree are summed up to get the final score. If you look at the example, an important fact is that the two trees try to complement each other. Mathematically, we can write our model in the form:

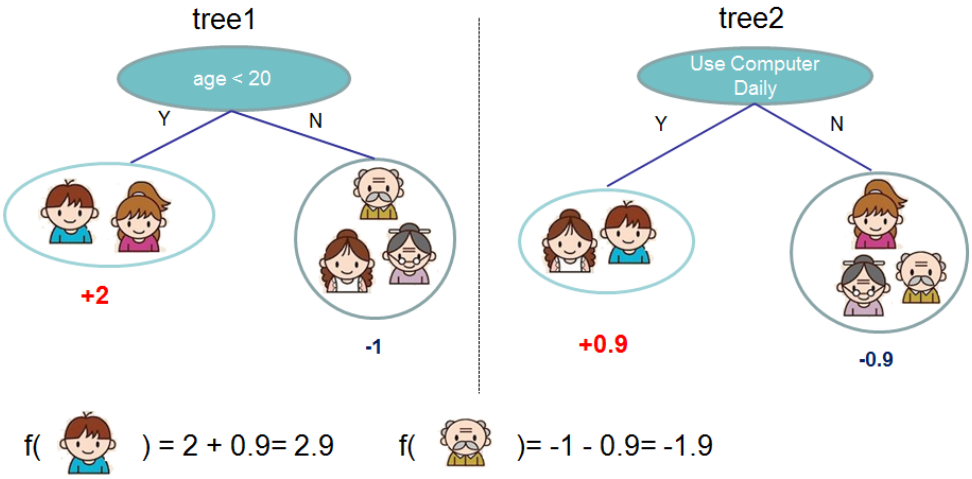


Figure 1.3: Example of a classification with a more trees

$$\hat{y}_i = \sum_{j=1}^{M} g_j(x_i), \quad g_j \in F \tag{1.3}$$

where M is the number of trees, g is a function in the functional space F and F is the set of all possible CART's [6].

Now that we have an idea of what an ensemble of trees is we can move to specify what we mean by boosting. Since we are evaluating predictions from different trees, we need a way to combine those predictions. With boosting we associate a weight (a_k) to each contributes to the aggregated result. All the ensembles are developed in python with *"XGBoost"* library. The main question that we want to answer is: how XGBoost can be trained?

Before delving into the mathematical details we need to properly parametrize the trees. A decision tree j, denoted as $g_j(x)$ with T leaves is parametrized by two quantities: a function $q(x)$ that maps each data point to one of the leaves of the tree and a weight vector w that assigns a predicted value to each leaf. In other words, the decision tree's prediction for the datapoint x_i is simply: $q(x_i) = w_{q(x_i)}$. Like others models we need first to introduce a function to minimize. The literature, referring to trees, calls this function the cost function. Generally we have two terms in this function: the first term that measures the goodness of the predictions on each datapoint (l_i) and a regularization term (Ω) for each tree in the ensemble that does not depend on the data. The entire cost function can be expressed as:

$$C(X, g_a) = \sum_{i=1}^{N} l(y_i, \hat{y}_i) + \sum_{j=1}^{M} \Omega(g_j) \tag{1.4}$$

with i and j that respectively run over the data points and the decision trees in our ensemble. In XGBoost the regularization term is chosen to be

$$\Omega(g) = \gamma T + \lambda/2||w||^2 \tag{1.5}$$

During the training we form our ensemble iteratively. We define a family of predictors $\hat{y}_i^t$ as:

$$\hat{y}_i^t = \sum_{j=1}^{t} g_j(x_i) = \hat{y}_i^{t-1} + g_t(x_i) \tag{1.6}$$

Since for large t each tree becomes a perturbation of the predictor, we can perform, a Taylor expansion on our loss function to second order

$$C_t = \sum_{i=1}^{N} l(y_i, \hat{y}_i^{t-1} + g_t(x_i)) + \Omega(g_t) \approx C_{t-1} + \Delta C_t \tag{1.7}$$

with

$$\Delta C_t = a_i l(y_i, \hat{y}_i^{t-1}) g_t(x_i) + \frac{1}{2} b_i g_t(x_i)^2 + \Omega(g_t) \tag{1.8}$$

where

$$a_i = \partial_{\hat{y}_i^{t-1}} l(y_i, \hat{y}_i^{t-1}) \tag{1.9}$$

$$b_i = \partial^2_{\hat{y}_i^{t-1}} l(y_i, \hat{y}_i^{t-1}) \tag{1.10}$$

So we choose the t-th decision tree g_t to kinimize ΔC_t.
We can actually derive an expression for the parameters of g_t that minimize ΔC_t analytically. To simplify notation, it is useful to define the set of points x_i that get mapped to leaf j : $I_j = i : q_t(x_i) = j$ and the functions $B_j = \sum_{i \in I_j} b_i$ and $A_j = \sum_{i \in I_j} a_i$. Notice that in terms of these quantities, we can write

$$\Delta C_t = \sum_{j=1}^{T} [B_j w_j + \frac{1}{2}(A_j + \lambda) w_j^2] + \gamma T \tag{1.11}$$

To find the optimal solution we make the gradient with respect to w_j and set it to zero, to get

$$w_j^{opt} = -\frac{B_j}{A_j + \lambda} \tag{1.12}$$

and replacing it in the previous expression we find the optimal ΔC_t:

$$\Delta C_t^{opt} = -\frac{1}{2} \sum_{j=1}^{T} \frac{B_j^2}{A_j + \lambda} + \gamma T \tag{1.13}$$

ΔC_t^{opt} measures the in-sample performance of g_t and we should find the decision tree that minimizes this value [3]. To do that in practice an approximate greedy algorithm is run that optimizes one level of the tree at a time by trying to find optimal splits of the data. This leads to a tree that is a good local minimum of ΔC_t^{opt} which is then added to the ensemble. This is only a very high level sketch of how the algorithm works. In practice, additional regularization such as shrinkage [7] and feature subsampling [8] [9] is also used. In addition, there are many numerical and technical tricks used for the approximate algorithm and how to find splits of the data that give good decision trees [6]. Anyway since we use trees ensembles as a comparison and not as the main model, as we will see in the results section, we do not investigate more in this field.

1.4 Neural network

Over the last years, neural networks have emerged as one of the most powerful supervised learning techniques. Due to the generalization introduced in the approach to the problems by neural networks, they are seeing an high variety of applications. It is possible to find a huge literature on the usage of neural networks in economics [10], medicine [11], self driving car [12] and so on. Despite the recent development and "intensive" use of this technology, neural networks have a long history. The first works on models that we can define as ancestors of neural networks were presented in the late 1940s for example from D. O. Hebb [13] that create a learning hypothesis based on the mechanism of neural

plasticity that became known as Hebbian learning. Research stagnated in this field around the 1970s due to the main problem: computers didn't have enough processing power to effectively handle the work required by large neural networks. The interest in this discipline awakened after the introduction of a practical learning method for neural networks: backpropagation algorithm by P. J. Werbos [14]. Since in our work we are going to use fully connected networks, the backpropagation algorithm for the training will be explained later in rigorous mathematical terms. The second strong revolution in the entire subject of machine learning but particularly for neural networks comes with the new millennium. We have in fact that the strong development in the world of computer hardware has made it possible to train large neural networks and reach levels of performance that were not technically achievable before. The arising of the GPUs in the last decade has speeded up again the computational time for the training and opened the frontiers for the development of new structures of networks and data to analyze.

After describing from a historical point of view the introduction of these models in the scientific world, we want to give a theoretical overview of what a neural network is and how it works. Neural networks are neural inspired nonlinear models for supervised learning. The basic building blocks of a neural network are the neurons. To produce a mathematical scheme of how an artificial neuron works, we follow the principle of how a real neuron works. We can see a schematic representation of the morphology of a neuron in figure 1.4. The principal components of a neuron are:

- Soma: contains the cell nucleus
- Dendrites: ramifications that receive signals from other neurons and propagate them toward the soma
- Axon: single ramification that transmits the signal toward other neurons
- Synapses: chemical or physical junctions placed in the axon terminal allowing for transmission of the signals

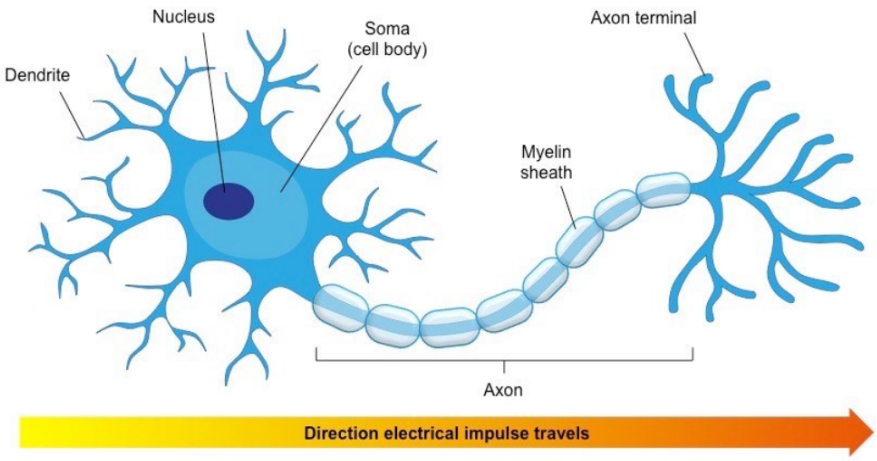


Figure 1.4: Schematic representation of a neuron

The functioning of a neuron therefore revolves around the transmission of an electrical signal. To maintain a voltage difference between the interior of the neuron and the exterior we have a so called membrane that acts a potential difference. In the membrane then we find some ion channel that can open as a result of chemical change or voltage change. What we want to model is the transmission of signals in a neuron. In particular how does electric signal propagate along the axon? The role of the membrane is to divide Na^+ from K^+ but due to the ion channel we can have the opening of these channels and the interaction between these elements. The first part of the axon (close to the soma) depolarises and an action potential is initiated by the opening of Na channels. Shortly after,

K channels also open allowing potassium ions to exit. For small voltage increases from rest, the K current exceeds the Na current and the voltage returns to its normal resting value. However, if the voltage surpasses a critical threshold the sodium current dominates: this causes even more channels to open, producing a greater depolarization of the cell membrane. The process proceeds explosively until all of the available Na ion channels are open. We refer to this behavior of the neuron by the name of "fire" or "spike" of the neuron. The Na influx spreads to the adjacent part of the axon, while Na channels in the current location close. Now K current dominates and repolarization begins, returning the electrochemical gradient to the resting state [15].

In the light of this information we want to model an artificial neuron so that it receives input signals from other neurons, processes the information and based on some function it fires and outputs a signal to pass to other neurons.
So let's call this basic unit as i that take as input a vector d of features $x = (x_1, x_2, \ldots, x_d)$ and produces a scalar output $a_i(x)$. A neural network consists of many such neurons stacked into layers. Now there are different way to connect layers and neurons. In our applications the output of one layer serving as the input for the next. The first layer in the neural net is called the input layer, the middle layers are often called "hidden layers", and the final layer is called the output layer (figure 1.5).

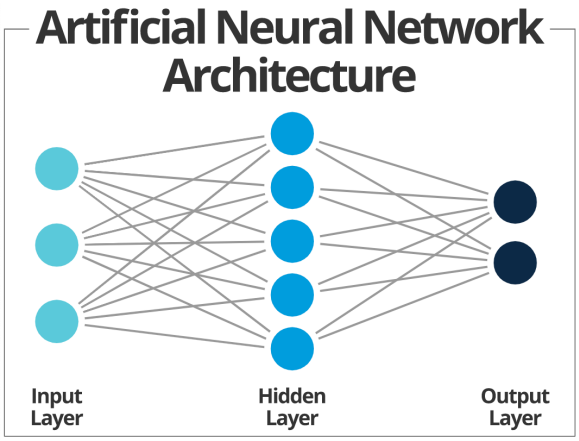


Figure 1.5: Schematic representation of a neural network with one hidden layer

In principle a_i can be decomposed into a linear function that give a weights to all the input received from the neurons and a non linear transformation $\sigma_i(z)$. The most common linear function is the dot product between the set of weights of a specific neuron $w^{(i)} = (w_1^{(i)}, w_2^{(i)}, \ldots, w_d^{(i)})$ and the input to the neuron and summing a bias to the result:

$$z^{(i)} = w^{(i)} \cdot x + b^{(i)} \tag{1.14}$$

A graphical representation of the process is shown in figure 1.6. The choice of the linear part is quite mandatory but there are different types of non linear functions to choose from. As we will see in the training section, various type of activation function are tested but the best performances in the majority of the case are reached with the ReLu function: $max(0, z)$.

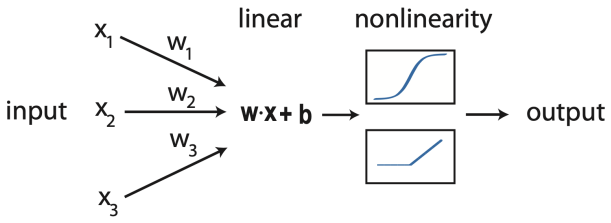


Figure 1.6: Mathematical process inside an artificial neuron

Once that is clear how a single neuron works, to construct an artificial neural network we have to organize neurons in layers and decide which type of connection to adopt between them. In our work we develop feed-forward neural networks and we will focus on that also for the theoretical introduction. A feed-forward is a structure where the input layer takes the input x and produces the output based on the current weights. This output information is used as input for the first hidden layer and the process is repeated until the output layer. The whole network can tush be considered as a complicated non linear transformation of the input x into an output $\hat{y}$. The use of the hidden layers expands the representational power of the neural networks and it is possible to demonstrate the so called universal approximation theorem which states that a neural network with a single hidden layer can approximate any continuous, multi-input/multi-output function with arbitrary accuracy [16]. Despite the theorem we are going to explore a more complex structure meaning that we add more than one hidden layer (deep networks). As we will see the best structure for a given task will be derived empirically. We want now to move to the training task for the feed forward networks. In this work we focus on fully connected networks meaning that we do not find any connection between neurons inside a single layer but all neuron in a layer is connected with all neuron in the previous and next layers, nevertheless the training structure is the same if some connections are not present.

Like the other supervised learning algorithms, to train a neural network we have to define a cost function (we will call it also loss function) and use minimization techniques to find the optimal minimum of this function. The main difference with the other algorithms is that the presence of hidden layers makes the evaluation of the gradient computationally more difficult. A brute force calculation is out of the question since it requires us to calculate as many gradients as parameters at each step of the gradient descent so we need a different procedure. An algorithm that exploits the layered structure of neural networks to more efficiently compute gradients is the already mentioned "backpropagation". This algorithm is simply the chain rule for partial differentiation and we can explicitly write the equations.
Let's call the layer with an index $l = 1, 2, \ldots, L$. Denote with w^l_{jk} the weight of the connection between the k-th neuron in layer $l-1$ and the j-th neuron in layer l, the bias of this neuron is b^l_j. Following the construction of the feed forward structure we can describe the activation a^l_j of the j-th neuron in the layer l-th as :

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right) = \sigma(z^l_j) \tag{1.15}$$

If we call the cost function E we have the dependency on the activities of the output layer but indirectly it depends also on the activities of all the other layers. Let us define the error Δ^L_j of the j-th neuron in the last layer as the change in cost function with respect to the weighted input z^L_j

$$\Delta^L_j = \frac{\partial E}{\partial z^L_j} \tag{1.16}$$

The generalization of the error of a neuron in layer l gives:

$$\Delta^l_j = \frac{\partial E}{\partial z^l_j} = \frac{\partial E}{\partial a^l_j} \cdot \sigma'(z^l_j) \tag{1.17}$$

where $\sigma'(z^l_j)$ denotes the derivative of the non-linearity $\sigma'(\cdot)$ with respect to its input evaluated at x. Notice that the error function Δ^l_j can also be interpreted as the partial derivative of the cost function with respect to the bias b^l_j, since

$$\Delta^l_j = \frac{\partial E}{\partial z^l_j} = \frac{\partial E}{\partial b^l_j}\frac{\partial b^l_j}{\partial z^l_j} = \frac{\partial E}{\partial b^l_j} \tag{1.18}$$

We need two other equations to characterize backpropagation and we derive that equations using the chain derivation rule. Since the error depends on neurons in layer l only through the activation of neurons in the subsequent layer $l+1$, we can use the chain rule to write

$$\Delta_j^l = \frac{\partial E}{\partial z_j^l} = \sum_k \frac{\partial E}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \Delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_l^l} = \left(\sum_k \Delta_k^{l+1} w_{kj}^{l+1} \right) \sigma'(z_j^l) \tag{1.19}$$

This is the third backpropagation equation. The final equation can be derived by differentiating of the cost function with respect to the weight w_{jk}^l as

$$\frac{\partial E}{\partial w_{jk}^l} = \frac{\partial E}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} = \Delta_j^l a_k^{l-1} \tag{1.20}$$

Together these four equations define the four backpropagation equations relating the gradients of the activations of various neurons a_j^l, the weighted inputs $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$, and the errors Δ_j^l . These equations can be combined into a simple, computationally efficient algorithm to calculate the gradient with respect to all parameters [16]. The algorithm follows five main steps:

- Activation at input layer: calculate the activations a_j^1 of all the neurons in the input layer.
- Feedforward: starting with the first layer, exploit the feed-forward architecture through Eq. 1.15 to compute z^l and a^l for each subsequent layer.
- Error at top layer: calculate the error of the top layer using Eq. 1.17 . This requires knowing the expression for the derivative of both the cost function $E(w) = E(a^L)$ and the activation function $\sigma(z)$.
- "Backpropagate" the error: use Eq. 1.19 to propagate the error backward and calculate Δ_j^l for all layers.
- Calculate gradient: use Eq. 1.18 and 1.20 to calculate $\frac{\partial E}{\partial b_j^l}$ and $\frac{\partial E}{\partial w_{jk}^l}$.

The algorithm consists of a forward pass from the bottom layer to the top layer where one calculates the weighted inputs and activations of all the neurons. One then backpropagates the error starting with the top layer down to the input layer and uses these errors to calculate the desired gradients. The power behind this algorithm is the computational "low cost" [3].

With this notion we finish the theoretical section of the work. Other specific information on the algorithm such as activation functions used, loss, structure and so on will be exposed in the training section in particular for the neural networks. We now move to the "practical part" of the work starting with the presentation of the data used.

Chapter 2

Data presentation

2.1 Overview

In this section we explore and describe the foundations on which all the work rests: the dataset. The trend of the most important private company is to use a cloud approach to handle each application and Radarmeteo, following this trend, develops its own applications exploiting Amazon Web Service(AWS). AWS provides a huge variety of services, from storage services to computational machines up to access to entire servers. Since Radarmeteo provides many real-time applications to customers, both storage and computation are managed using the AWS infrastructure.
In particular a database stores meteorological information obtained from various sources, from meteorological stations, to radars up to satellite information.
This work has the goal of spatializing some meteorological variables of fundamental interest in various fields over a uniform grid starting from information derived from the stations which, at first glance, we take as truthful or other sources that we want to exploit to retrieve some relation with the variable of our interest.
As we will see, different sources of information are exploited to make the predictions but each supervised learning algorithm uses as target variables temperatures and precipitations that come from the stations. To train all the algorithms, a subsample of the database on AWS was created in a time window that starts from 2018 until the end of 2020 (November and December excluded). In practice we make a screenshot over this time windows and we create an initial "csv" file. The starting date was selected in accordance with the implementation by Radarmeteo of a validation process for the raw data detected by the stations. The following sections provide an overview of this data used during the entire work. Once that the data is visualized, a preprocessing procedure is needed because independently from the algorithm, the preprocessing of the data is of fundamental importance in machine learning for two main reasons:

- the input of different algorithms may be different, for example a Neural Network (NN) require as input for categorical information a so called one hot vector (process that assigns a vector to each category) as we will see later.
- Even if a validation process takes place in the data we are considering, some wrong information may pass the test, so the preprocessing is used also to clear once again the input. As we will see later, this procedure arises some problem of the actual validation procedure and the results obtained with machine learning methods can help also under this aspect.

2.2 Data visualization

In this section we will present the data that we use to perform the predictions. Each algorithm developed in this work then uses as input all or some variables we are going to present. More information

we will given in the algorithms section.

2.2.1 Station

The majority of the meteorological information that we will use becomes from stations. A meteorological station (figure 2.1) is a set of measuring instruments that can register several meteorological variables depending on the model and the company.

The most frequent instruments present in this type of equipment are able to detect temperature, wind speed and direction, solar radiation, rainfall and so on. In addition to give an idea of the heterogeneity of the networks of stations we can find providers that give raw data, processed one, information with different time steps and so on. Radarmeteo can boast a dense network of stations belonging to various public and private entities as ARPA, Protezione Civile Italiana and others. These stations cover the whole national territory and future prospective concerns the extension of the monitored territory outside our country. To have uniform information over a territory like Italy, that presents a lot of different aspects, not only latitude and longitude are vastly explored and some stations are located also at high altitude.

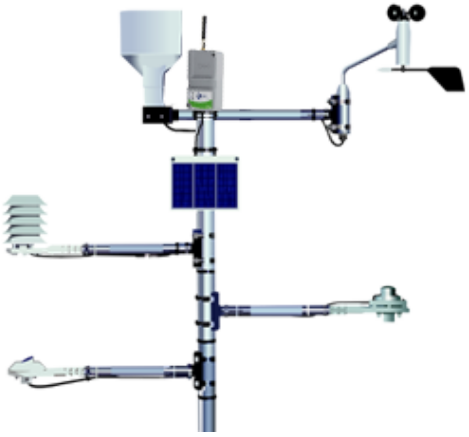


Figure 2.1: Examples of a professional meteorological station.

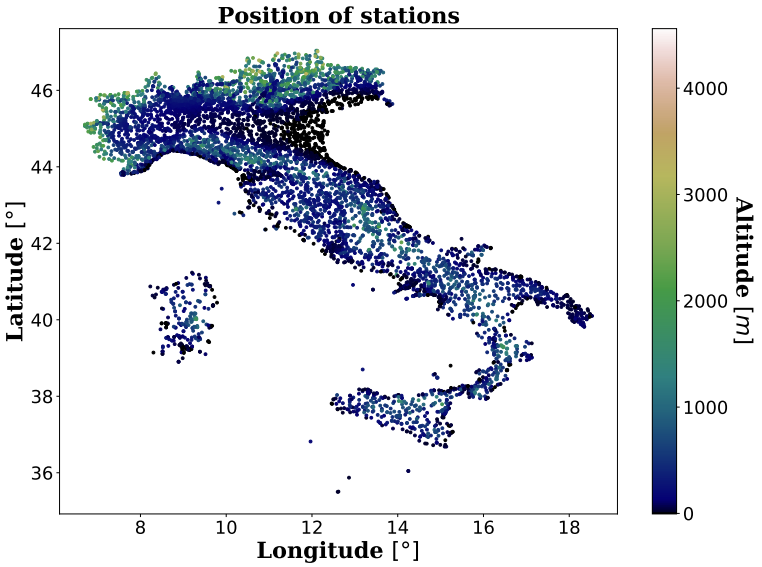
Figure 2.2a represents the cover of the network of stations used. At first glance we can recognize that almost every single province is covered by many stations. Nevertheless, as we will see, there is still the need to relate this data with other sources to have a homogeneous information.

2.3 Land Cover

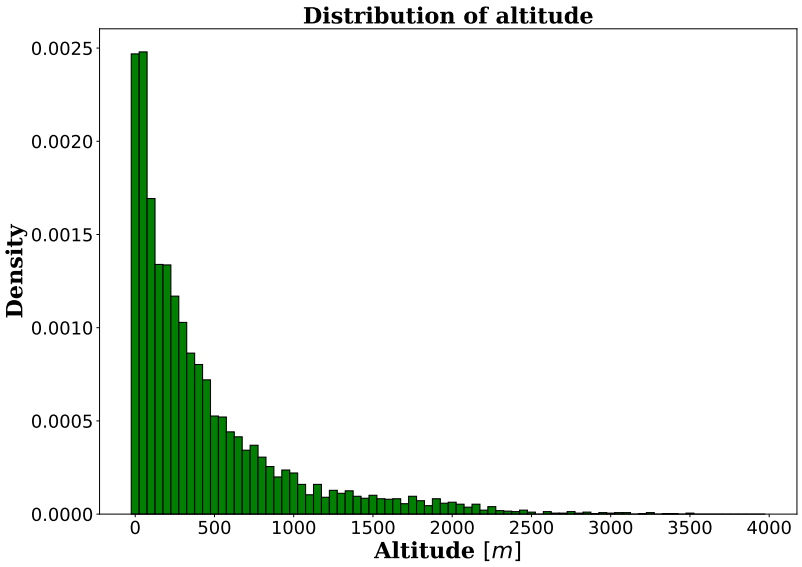
Since Italy is a very heterogeneous territory, altitude alone is not able to represent this heterogeneity in a proper way. In order to have a better description of the territory, several other variables are considered. The first of this variables that we decide to implement is the land cover that represents the type of terrain in a grid. The grid has a scale of 1 km (each cell is a square of side 1 km) equal to the scale of the final grid that we want to built to perform the spatialization of our main variables. Technically it is inappropriate to talk of km because all the grids that we use are split based on latitude and longitude and the euclidean distance is not the proper distance to use. Anyway to simplify the understanding of the concept we may refer to a distance which is actually 0.01 point in latitude/longitude as 1 km [1].
The netCDF file (usual extension in meteorology) that contain land cover information was build with 4 different categories:

- category marked with value of 1 takes the description of "Crop". With this we mean cultivated areas or rural areas without particular vertical vegetation developments.
- Category marked with value of 2 takes the description of "Trees + Water". Cells with woods, forests and areas with vertical vegetation developments fall into this category. In addition lakes and areas close to a river are marked with 2.

[1] For the implementation of the distance we will see that different distance was tried and a compromise between accuracy and computational time is to use Euclidean distance with some height to the x and y entries. More information will be given in section 3.1.1



(a) Positioning of the stations with the relative altitude described in the colorbar.



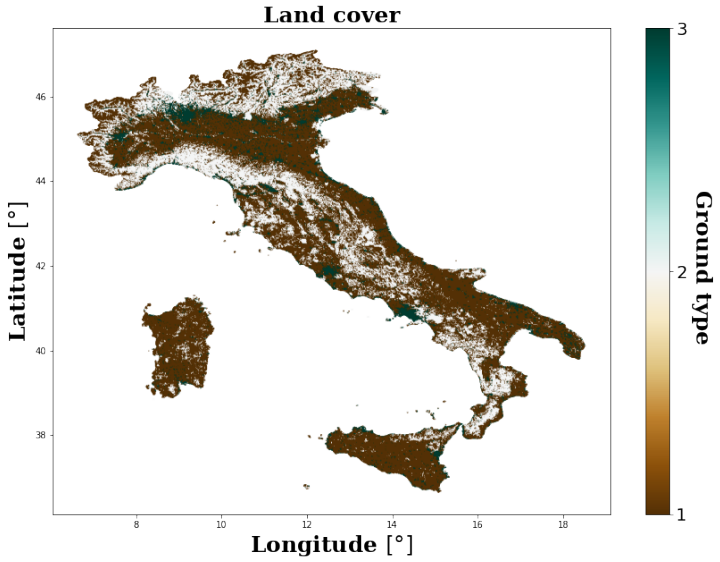
(b) Distribution of the altitude of the stations.

Figure 2.2: Information on altitude on the stations

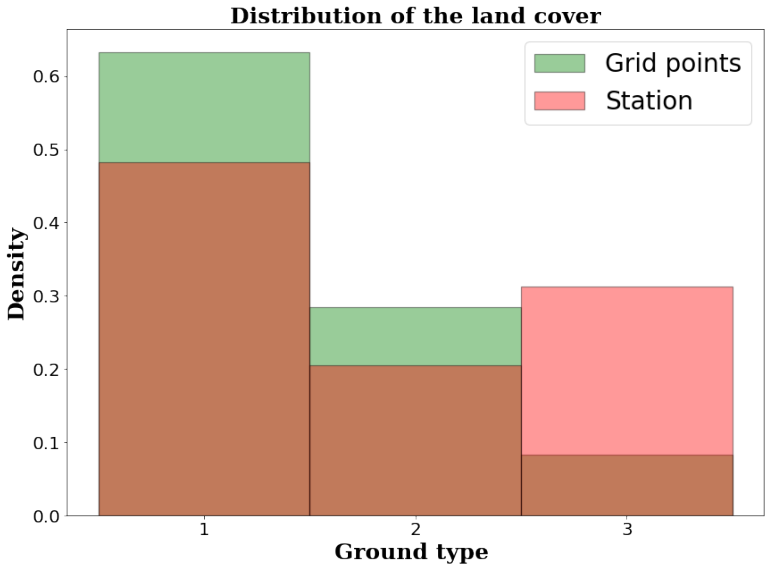
- Category marked with value of 3 takes the description of "Urban". This category contains not only big city and urban center bot also sufficiently large expanses of concrete, motorway junctions or industrial centers.
- The filler value is used to mark sea water and non-Italian territories.

To build this map we manipulate a pre-existing land cover database supplied by Copernicus [17] in the project *Land Monitoring Service*. The initial land cover map has a resolution of 250 m and contains almost 50 terrain category. In order to obtain the final map we perform a grouping of all the categories in the four main categories introduced above and then a regriding is performed. With regriding we mean the process that from a $4x4$ square of size 250^2 m^2 extracts a single cell with area 1 km^2. As we can see from figure 2.3 the most common ground type is Crop. From 2.3a it can be underlined once

again how the territory is heterogeneous. We can recognize big cities like Milano or Roma, the mountain where forests arise and places like Pianura Padana where there are an abundance of cultivated fields that reflect the crop category.
From a machine learning point of view figure 2.3b is an index of consistency of our dataset. Independently from the algorithm used, we are going to explore the paradigm of supervised learning and as said we take as truthful the data from the stations. Due to this, in a training phase, we use stations to feed our algorithms. On the other hand we are interested in produce some grids that contain many more points than stations and there is a need for the two datasets to be compatible with each other. Due to the heterogeneity of the network of stations used by Radarmeteo we have that the two distributions are comparable. The abundance of stations in urban areas is associated with the fact that small companies put stations close to their locations to reduce maintenance costs.



(a) Land cover of all the national territory. Ground type of 1 corresponds to crop, 2 corresponds to trees + water, 3 corresponds to urban.



(b) Distribution of the the land cover for both stations and grid points

Figure 2.3: Information on land covering of Italy

2.4 Slope riserun

Since in our work, or at least for what concern the temperatures, we want to build relationship between terrain variables and meteorological ones, the morphology of the considered area is of fundamental importance. Starting from this section we will present the variables that we can build from the manipulation of different DEMs. Each application that we will develop during this work will contain information on which of these variables are used to construct the grids.
Slope represents the inclination of a given point in the grid. To obtain the results presented in figure 2.4a the library Richdem [18] was exploited. This library provides several functions that perform algorithms based on the idea of the 2D derivation to builds the scalar field of the slope and in this particular case we use a DEM of Italy with a scale of 1 km.
In figure 2.4b the distribution of the slope is shown. Again we have that the distribution of the slope of the grid points is very similar to the distribution of the slope of the station. As mentioned before this is a signal of consistency between the training phase and the final spatialization phase.

2.5 Aspect

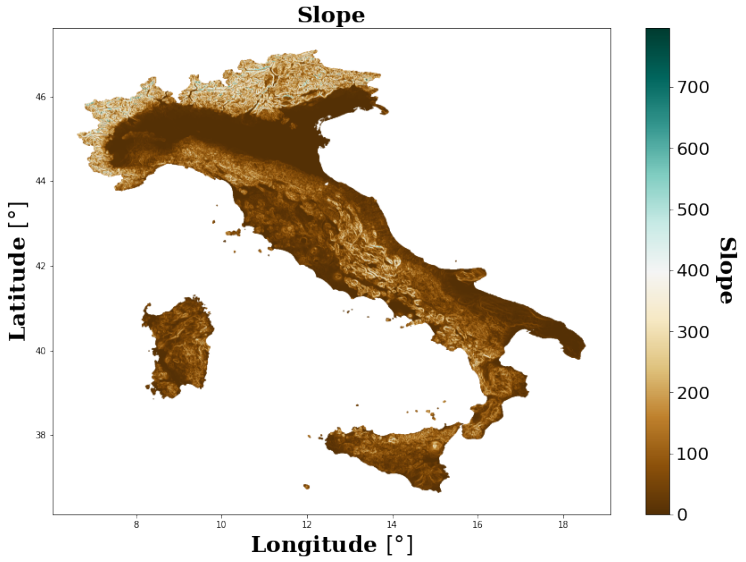
The second variable that we obtain from a manipulation of the DEM with a resolution of 1 km is the aspect of the points. With aspect we mean the direction of the maximum slope of the considered cell. The value is in degrees. The usage of this variables is related to the exposition of a point to solar radiation that can significantly change the behavior of certain areas considering some meteorological variables. For example if a wind blows from south to north, a station in a point with a value of aspect close to 270°, which is located on the north side of a mountain, will probably detect a wind intensity lower than a station with a aspect of 90°.
Figure 2.5a shows the aspect of the grid points, we can appreciate the difference of the aspect in the mountain ranges. As regards the comparison of this variable between grid points and stations, conclusions similar to slope and land cover can be drawn. In practice, and in particular for maximum temperatures and for the precipitations where we expect a significant relation with this variables, we manipulate the values of the aspect in order to obtain a discrete variable from a continuous one. To do so we assign four descriptions based on the numerical value: north, south, west, east.

2.6 Difference of DEMs

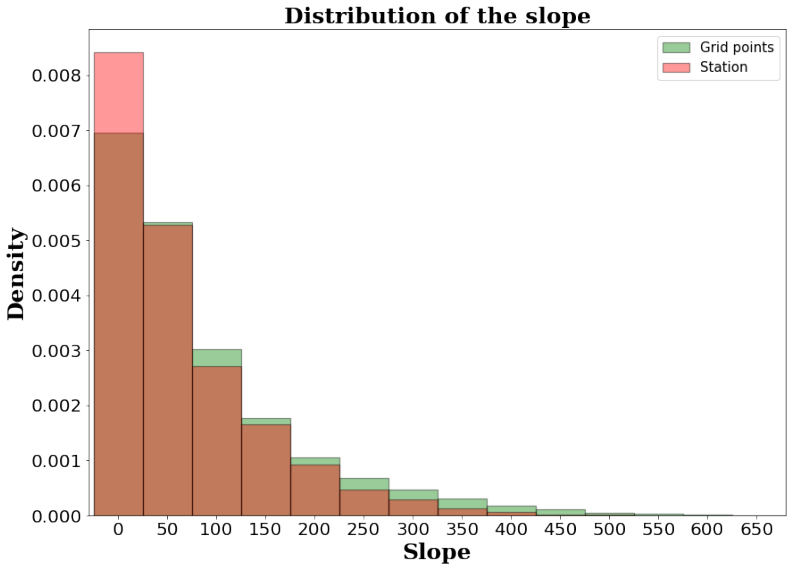
Not only the Dem with resolution of 1 km is available. Radarmeteo provides different scales and the combination of DEMs at different resolutions could be very useful. In particular the difference between a DEM with a resolution of 4 km and the DEM that we use to build the grid (with resolution of 1 km) gives information about the curvature of the considered point. Depression or peaks will be highlighted with this variables. To emphasize this difference and characterize points where important meteorological phenomena can occur, such as thermal inversions, an offset is used before calculating the difference. The final netCDF is obtained as the altitude of points of the grid with 1 km of resolution minus the altitude of points of the grid with 4 km - 30 m. The value of 30 m is selected by Radarmeteo as optimal value to recognize depression in specific points where is known that the effect of the thermal inversion is present.
With difference of DEMs we close the section of the raw data that we extract from DEMs and from their manipulations. In the next session we start to present data relative to stations or that becomes from different sources like the information from meteorological models. Almost all this type of data is in a certain way marked as target variables (or at least those coming from the stations) meaning that our algorithms try to reproduce the values of such variables in the final grids.

2.7 Minimum temperature

One of the most important variables in the field of insurance and agrometeorology is the minimum temperature. The minimum temperature is of fundamental importance, for example, to assign a risk



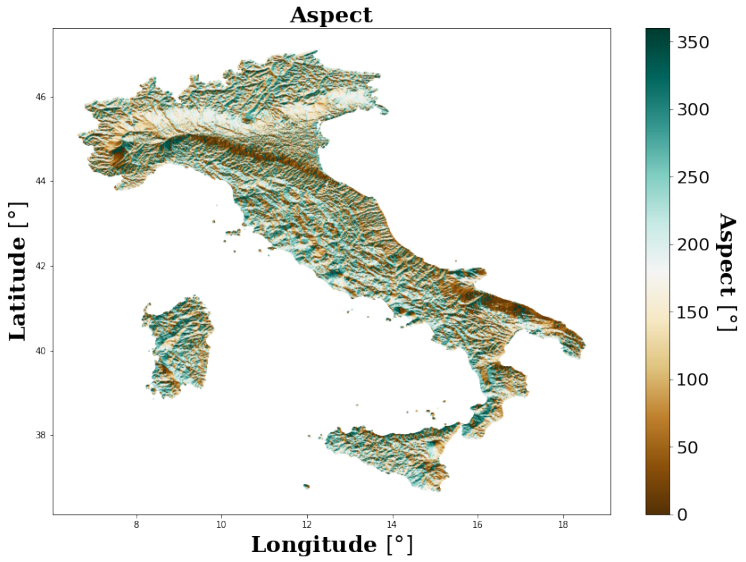
(a) Slope of the points of the grid



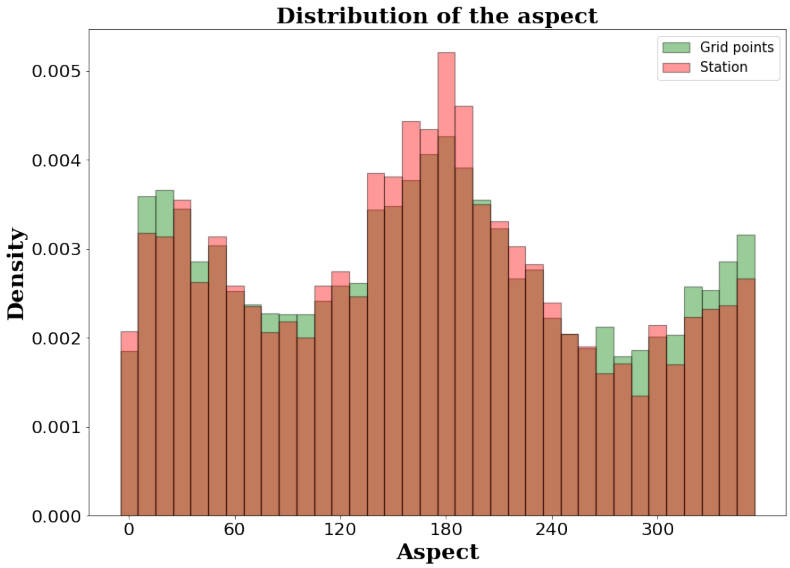
(b) Distribution of the slope for both stations and grid points

Figure 2.4: Information on the slope of Italy

band to a certain area. We have so that the forecasting of minimum temperature needs to have good accuracy. Apart from this application where forecasting techniques are required, a huge work regards also historical data. The knowledge of the event under study in a given location is required for example by law if there are claims for compensation in agriculture or other categories. The importance of these fields of application from a social and economical point of view requires that the estimation of the variables directly involved be highly reliable. The first tool we rely on to estimate the minimum temperatures are stations. Stations with a temperature sensor are very widespread so in fact we can construct a network with a high number of nodes. The database stores the information by date and figure 2.7 shows two typical behavior of the minimum temperature during winter and during summer. Again we can see how Italy is a very varied territory. This aspect of Italy is for sure interesting from a meteorological point of view but it is also challenging in a machine learning perspective because the risk of overfitting areas where stations are more dense is very high.



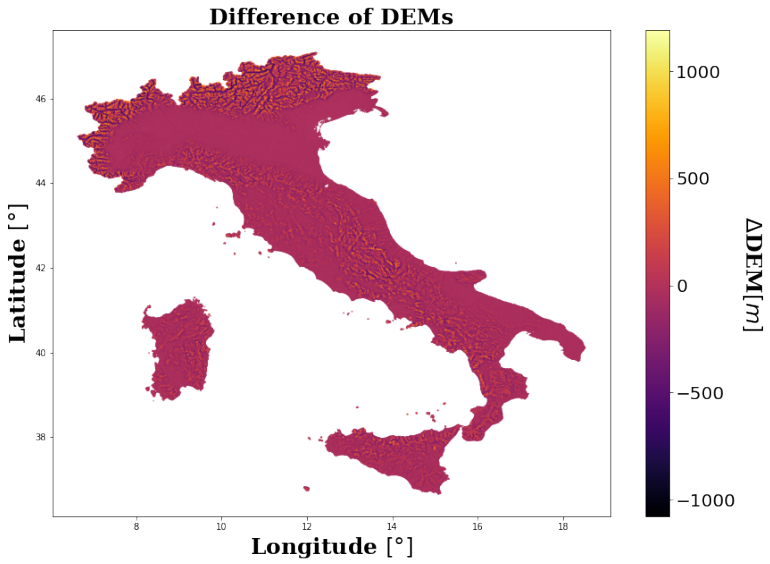
(a) Aspect of the points of the grid



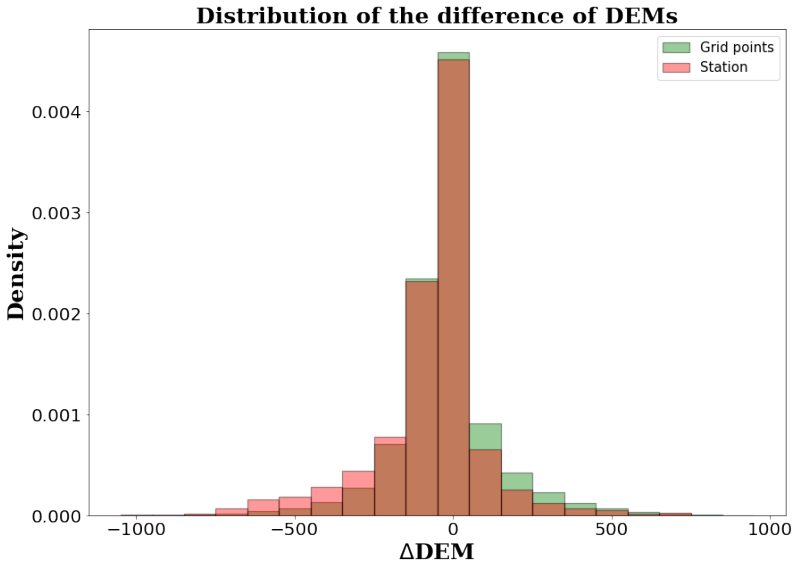
(b) Distribution of the aspect for both stations and grid points

Figure 2.5: Information on the aspect of Italy

An interesting phenomenon affecting minimum temperatures is thermal inversion. Thermal inversion is a complex combination of meteorological factors that can be summarized as an inversion on the normal relation between temperature and altitude; normally as we evaluate the temperature at higher altitudes we will find lower and lower values. In a situation of thermal inversion we have that increasing the altitude we find higher values of temperature up to a threshold altitude. This behavior happens for the most in points of depression with respect to the surrounding area and this is the first explanation of the introduction of all the variables retrieved from DEMs. Certainly the complexity of the phenomenon cannot be entirely explained within the curvature; other information could help to explore this behaviour of the temperature. For example the cloud cover can gives good hints. As we will see in the next chapter we do not directly use this variable because if on one hand it helps us, on the other hand we have the problem of the fog (often detected as cloud) that has a meteorologically opposite role to the cloud at high altitude. For this reason, information without misunderstanding is



(a) Difference of DEMs on the points of the grid



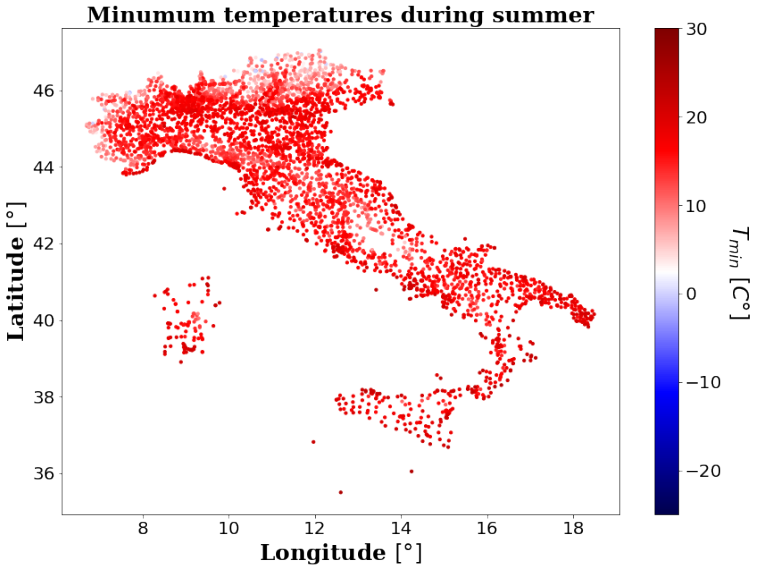
(b) Distribution of the difference of DEMs for both stations and grid points

Figure 2.6: Information on the difference of DEMs of Italy

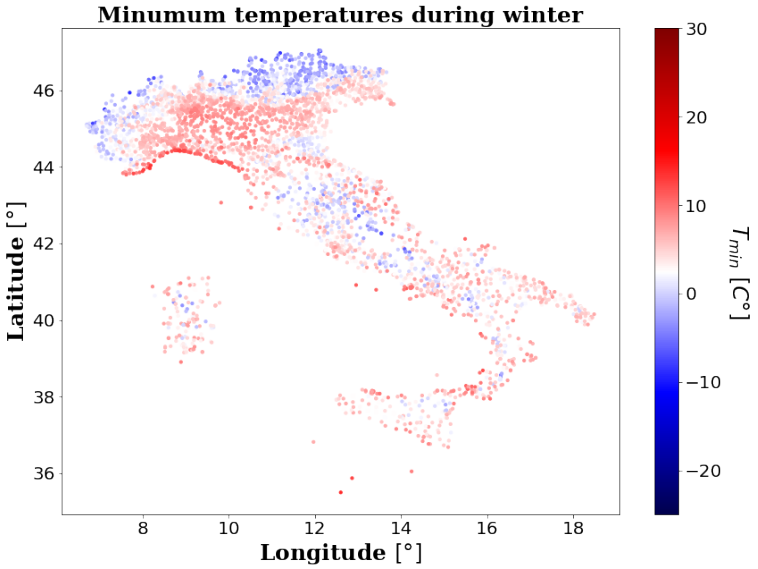
identified in the concavity of the terrain. We will present in detail the strategy to evaluate thermal inversions in the analysis section.

2.7.1 Minimum temperature from model

One of the most diffused methods to construct the value of a meteorological variable to fill a grid is to use meteorological models. The basis of these models is the resolution of the governing system of equations. Generally these equations cannot be solved analytically, they must be converted to a form that can be. The numerical methods are typically used to accomplish this task. We can summarize the equations to solve and that serve as the basis for most numerical weather and climate prediction models: the momentum equations for a spherical Earth represent Newton's second law of motion, which states that the rate of change of momentum of a body is proportional to the resultant force



(a) Minimum temperature registered by stations in a typical day during summer.



(b) Minimum temperature registered by stations in a typical day during winter.

Figure 2.7: Information on minimum temperature of Italy retrieved by stations

acting on the body, and is in the same direction as the force. The thermodynamic energy equation accounts for various effects, both adiabatic and diabatic, on temperature. The continuity equation for total mass states that mass is neither gained nor destroyed, and the analogous applied only to water vapor. The ideal gas law relates temperature, pressure, and density. A complete model will also have continuity equations for cloud water, cloud ice, and the different types of precipitation. The equations are called the primitive equations, and models that are based on these equations are called primitive-equation models. This terminology is used to distinguish these models from ones that are based on differentiated versions of the equations, such as the vorticity equation. As we can see the complexity increases rapidly in the resolution of these type of systems. To be able to handle that problem it is possible to use some approximations which inevitably introduces an error. Different type of models are possible and a common procedure to evaluate the forecast and to improve the accuracy

of these models are to make an ensemble approach in which different runs (also from different models) are performed and compared [19]. Anyway the scope of this work is to delve into machine learning researches and techniques and in light of this we will use the data from models as retrieved from a black box and we will also leave a rigorous theoretical description to the reader (see citation [19]) .

To make a comparison with the stations the first variable that we present is the minimum temperature. For consistency we produce plots on the same days of figure 2.7. As we can see from figure 2.9 the situation is more or less the same found by the stations. From these two variables and their manipulation we can obtain interesting information. The distribution of these two variables is presented in figure 2.8.
First of all, since in our approach we consider stations as truthful, we can compare the minimum temperatures given by the model with the "real" values to make an evaluation of the model itself. In practice the model is built over a DEM with a resolution of 4 km and every point of Italy can be reconstructed by interrogating this grid. To do so we need to divide each cell of side 4 km in a 16 sub-cells of side 1 km in order to reach our final goal. In the analysis section we will return on this approach as an estimator of minimum temperature but we want to anticipate the Mean Absolute Error (MAE) of the meteorological model: 1.94 C°. So the evaluation of the model arise a mean absolute error of almost 2 C° that we can consider a good starting point for our algorithms.
Since the entire work regards a machine learning approach to the problems, we need to have some data to learn from. Since data in this field are of fundamental importance, not only the number of events recorded is important but also the quality of the information. Learning from bad data in fact intrinsically drives to a bad predictor with a high value of the loss function. For this reason a preprocessing on the minimum temperature consist in exploit the combination of the information from model and from stations to clean our dataset from bad data that in some way passes the validation procedure. If we plot one variable as a function of the other as in figure 2.10a we can see that some problems arise. Due to the mean error of the model, it is very strange to have data with a difference of temperature higher than 20 C° as the points inside the red area in the plot so we decide to eliminate these points. Delving a bit into the problem we find that some pre-existing scripts corrupt the output values of temperature and the validation process can't detect this problem very well. A common example is shown in figure 2.10b where a value of -6.3 C° is reported at the north of Isola d'Elba on 22 July 2019 that is clearly wrong. The strategy applied to clear the entire dataset is to eliminate each row (that corresponds to a station in a given day) where the value of minimum temperature registered differs from the value reported by the model, in absolute value, by 15 C°.
Another anomaly detected graphically thanks to 2.10a is the strange behavior at $T_{min} = 0.0$ C° where there is a clear horizontal line. These are spurious data that we eliminate with the following filter: if the minimum temperature registered by the station is 0.0 C° and the relative value of the model differ more than 2 C° in absolute value, the line is eliminated from the dataset. At the end of the filtering phase we clear from our dataset the 1.3% of the events meaning almost 60 000 events and we have a final number of "good event" of around 4 550 000.

Once that the dataset is clean we can comment on the behavior of the model itself. Looking at the scatter plot in figure 2.10a and in particular the relationship with the altitude, we have a cluster of yellow points in the bottom-left part of the plot. This behavior means that at high altitude ($> 3800/4000$ m) the model tends to overestimate the minimum temperature. Graphically the center of the cluster is between a model temperature of $-10/-20$ C° and a station temperature of $-15/-25$ C°. We can explain this behavior if we consider the grid on which the model is constructed (we remember that each cell has side of 4 km). With this low resolution the model, regardless of the quality of the estimate, is not able to see high peak of the mountain because, inside the cell, peak tends to be planed and the loss in altitude can explain that loss in temperature.

2.7.2 Dependency of minimum temperature from other variables

Since T_{min} is an important target variable, we want to present the explicit relation between the other variables introduced above and the minimum temperature itself.

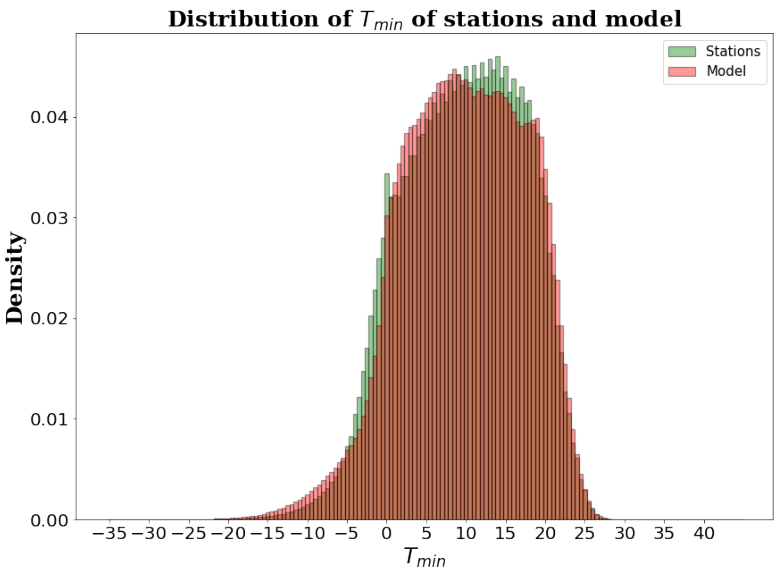
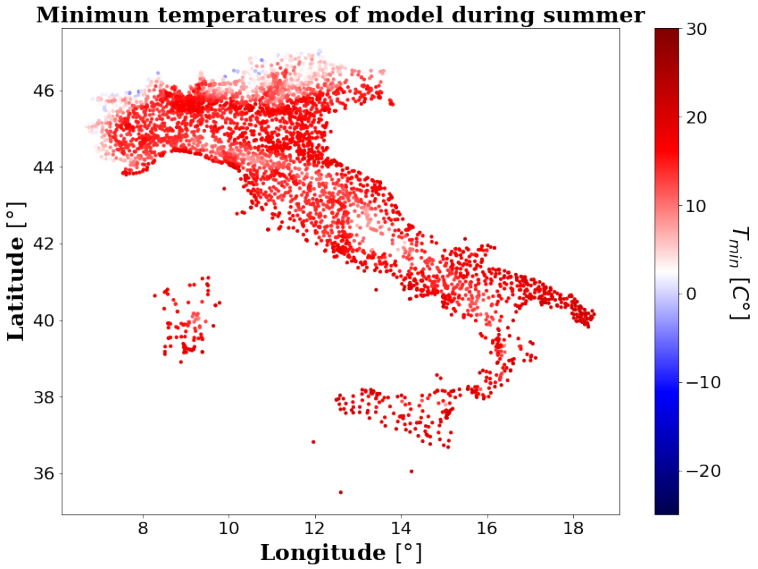


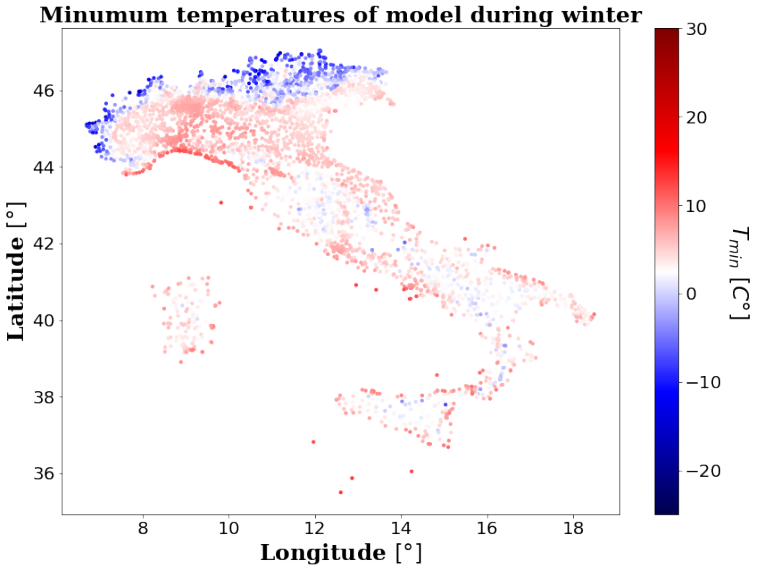
Figure 2.8: Distribution of T_{min} given by stations and model.

Since Italy does not cover a huge territory in terms of longitude we do not present the correlation with this variable of the minimum temperature; on the other hand we have that latitude is interesting. We have from the Köppen climate classification that Italy is divided into two categories: the northern area and the central part is relatively cool with a mid-latitude version of the Humid subtropical climate. For what concerns the mid-south of Italy we find a Mediterranean climate profile. Figure 2.11a shows what we expect for such a climatic category: as the latitude increase we are moving to the north and we encounter more rigid temperatures. The colorbar gives information also about the altitude of the points in the plot. High altitudes correspond to the possibility of having more cold days and in addition we can see that the plot seems to have more light colors on the right side; this corresponds to stations located in the Alpi that is a mountain range in the north of the territory.
Figure 2.11b shows instead the correlation between minimum temperature and the altitude. The behavior of the temperature is totally expected; we have indeed that as the altitude increase, the temperature trend is to decrease as confirmed from the regression line. Since the colorbar represent the ground type, we can also comment on the distribution of the land cover: urban points cover relatively low altitude while in high mountain we found for the most crop terrain. From the point of view of the temperatures we have that urban territories appear to be less cold, the transient zone of the plot is dominated by water + trees and the coldest type of ground is the crop one.

For what concern slope, aspect and ΔDEM we will see from figure 2.12 that the correlations are much lower with respect to the variables previously presented. In particular we have that the aspect is uniformly distributed with respect to temperatures. To explain this behavior we do not expect a great correlation between these two variables in fact generally the minimum temperature is reached during the night and the aspect of a point is more interesting in other fields, during the day for example to give information on the exposition to solar radiation or considering variables such as wind. Based on this consideration, as we will see, we take off aspect from the inputs of the neural networks because it does not provide important information. The situation is a bit different for both slope and ΔDEM because in this case we expect that these variables intervene in particular situations and their behavior will be more a correction behavior rather than a correlation one. With this we mean that we expect from the algorithms to retrieve important information based on other variables and then slope and ΔDEM correct the prediction. Another role for this two variables will be presented when we talk about the algorithm based on information on neighbors; in that case this information serve to select the neighbor with the most equal condition of the unknown point. Since one of our goals in the field of the minimum temperature is to detect inversions, ΔDEM and slope were developed



(a) Minimum temperature of the model evaluated in station points in a typical day during summer.



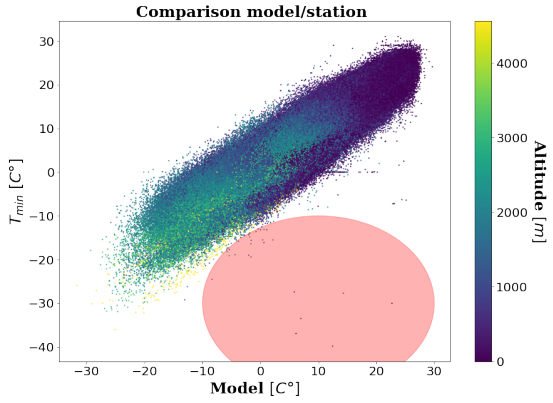
(b) Minimum temperature of the model evaluated in station points in a typical day during winter.

Figure 2.9: Information on model

ad-hoc for this situation and we do not require and do not expect that in a standard evaluation they become dominant. Nevertheless we can comment the plots since they have some intrinsic information: ΔDEM has a roof shape centered in 0 that mean that in places with very similar surrounding area (plains) we can reach higher temperatures. The same for the slope: as the slope increases we find lower temperatures (generally in mountains).

Nevertheless we perform several other test before discarding some variables as input for our algorithms because in a Neural Network for example we have non linear activation function and non linear relation can be found between variables in a way that a simple correlation plot cannot catch.

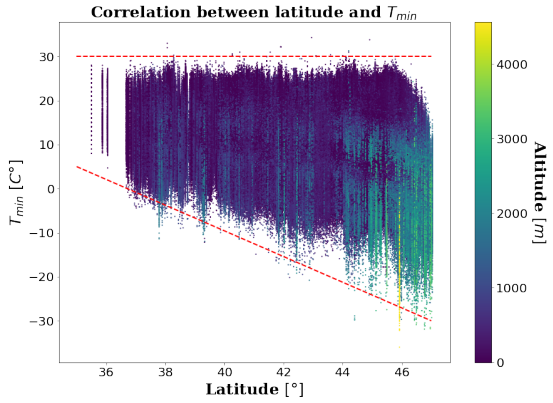
As will be described in the next chapter we develop our model based on information about the



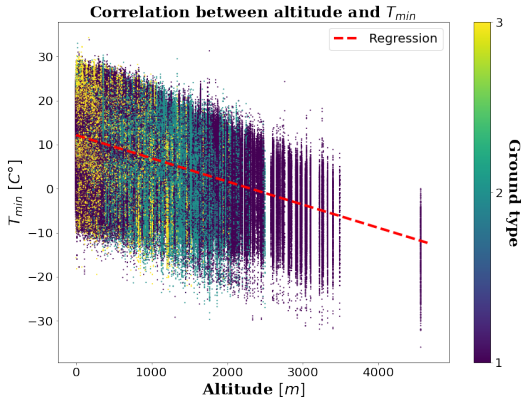
(a) Minimum temperature of stations as function of the model evaluated in the same points.

(b) Example of bad data detected by the difference between model and stations.

Figure 2.10: Data inside the red area is clearly outlayers in the distribution of the difference between minimum temperature of station and model. These points are station that pass the validation test but was corrupted by some script. After this identification we can clear our dataset to improve the learning of our algorithms.



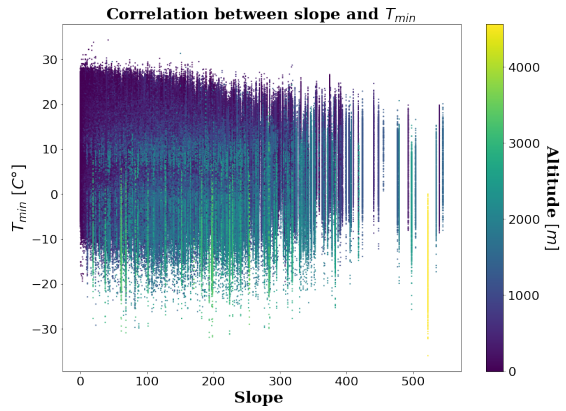
(a)



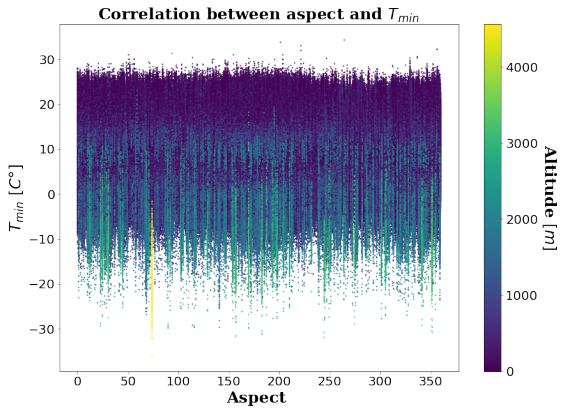
(b)

Figure 2.11: (a) Correlation between minimum temperature and latitude. As the latitude increase we encounter more rigid temperatures. All points of the database was displayed so we can see the behavior during all the months of the years. The colorbar represents the altitude of the points. (b) Correlation between minimum temperature and altitude. As the altitude increase, the mean value of the temperature decrease. The colorbar represents the ground type of the points.

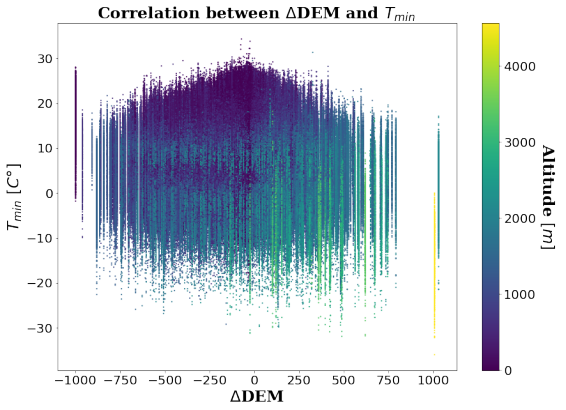
six neighbors (stations) of a point and in particular the most useful information are the minimum temperatures of these points. Since we do not expect a clear correlation with others information such as land cover, distance ... we do not present the correlation plot of these variables.
For what concerns the correlations with the minimum temperature reported by the nearest station we can see from figure 2.13 that as we increase the index of the neighbor, the points in the plots seem to be more spread according to the increasing of the distance and the consequent change of situation. As a confirm we have that from the colors in the plots nearest neighbor 6 is more blue/green with respect to the other; in fact the color corresponds to the distance from the considered point.



(a)



(b)



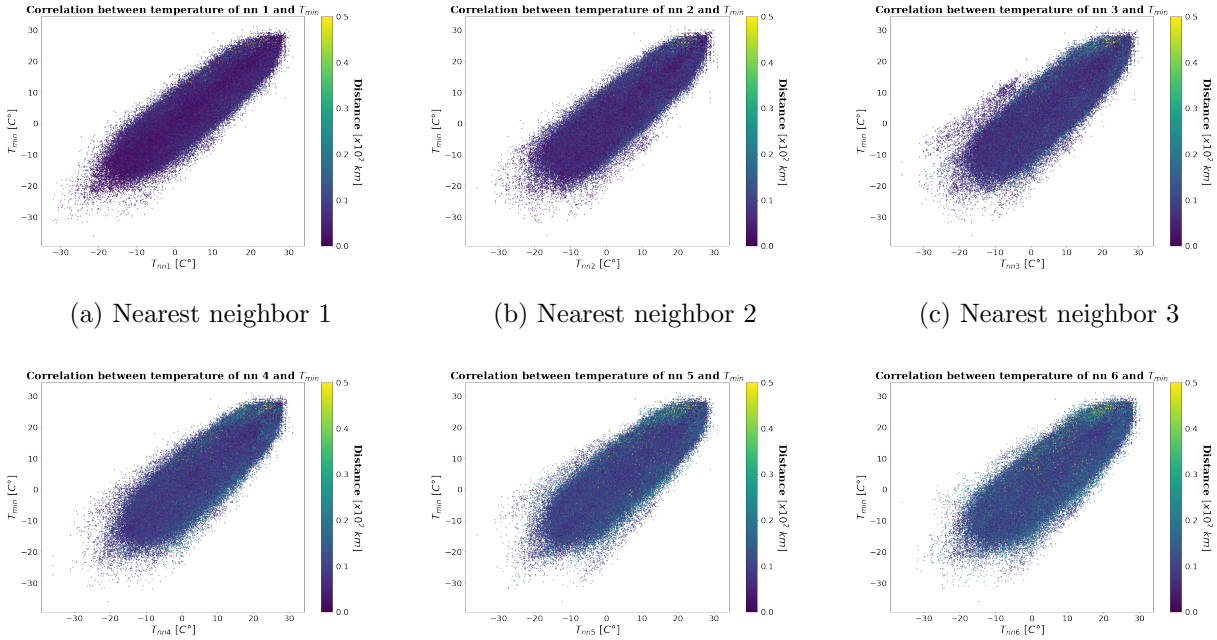
(c)

Figure 2.12: Correlation with other morphological variables: (a) Correlation between minimum temperature and slope. There isn't a clear signal of correlation but at high value of slope the minimum temperature appears to vary in a smaller range. (b) Correlation between minimum temperature and aspect. At first glance no correlation at all appears. (c) Correlation between minimum temperature and ΔDEM. At the extremes of the plot the field of variation of the temperature is restricted with respect to the middle.

2.8 Maximum temperature

The second variable we are interested in is the maximum temperature reached during the day from a point in a grid. We will follow the same structure of the presentation of information regarding minimum temperature and same subsection may be redundant.
We want to start with an example of maximum temperature during winter and summer: figure 2.14.

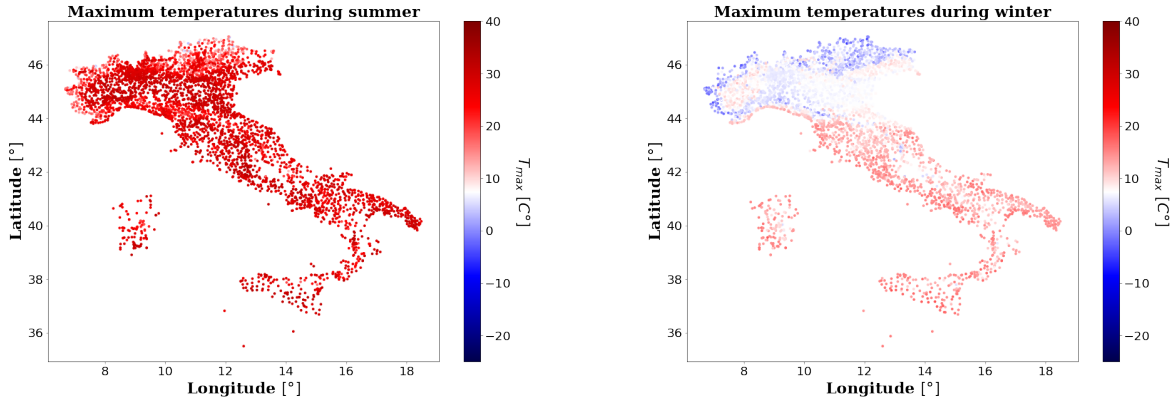
As we can see from this figure we have found confirmation of the above introduced Köppen climate classification. Especially in winter in fact the north of Italy reach lower maximum temperatures with respect to the south and the islands. On the other hand we can comment the behavior of the temperatures on summer when high values are reached also at high altitudes.
Maximum temperatures are very important in some fields where also the minimum temperatures are required. For example in agriculture is fundamental to know how hot it has been in a particular place to determine if a plantation is suitable for this area or to give an impartial response to a crop loss dispute due to high temperatures. For these reasons a spatialization of maximum temperatures is necessary in a meteorological company. Unlike what happens for minimum temperatures we expect that the prediction of the maximum temperature to be easier. In fact in this field we do not have strange



(a) Nearest neighbor 1 (b) Nearest neighbor 2 (c) Nearest neighbor 3

(d) Nearest neighbor 4 (e) Nearest neighbor 5 (f) Nearest neighbor 6

Figure 2.13: Correlation between the minimum temperature and the values reported by the neighbors stations



(a) (b)

Figure 2.14: Typical behaviour of the maximum temperatures during summer (a) and during winter (b)

behavior like thermal inversion and we can retrieve some useful information with less complexity. As we will see in the results section a multi linear regressor performs very well with a value of error higher than a deep neural network but still competitive. With this we mean that a linear model could already be sufficient to achieve the desired goal that intrinsically corresponds to a sign of a low complexity of the problem being treated.

2.8.1 Maximum temperature from model

The same models used and present above that regards minimum temperature are able also to report maximum temperature. From figure 2.15 we can see the distribution of the maximum temperature registered by stations and given by the models. If we make a comparison with the behaviors of the model for the maximum temperature and the minimum temperature (fig 2.8) we have two different

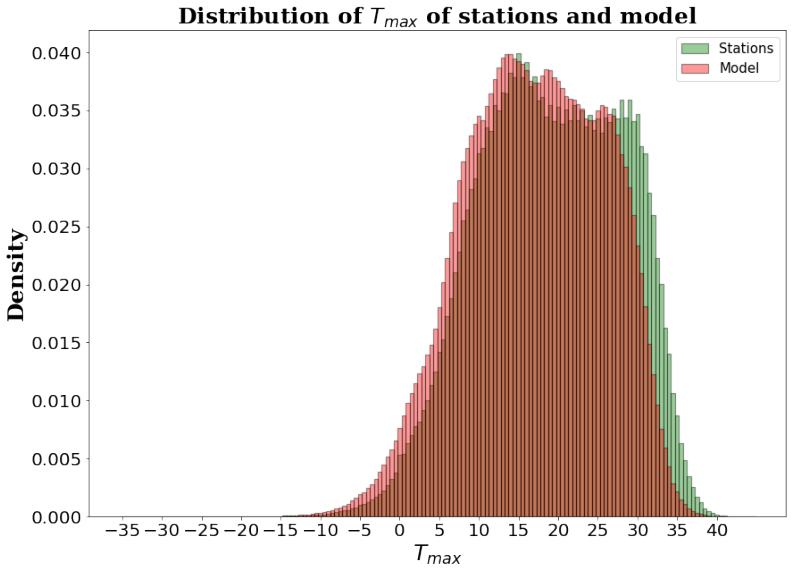
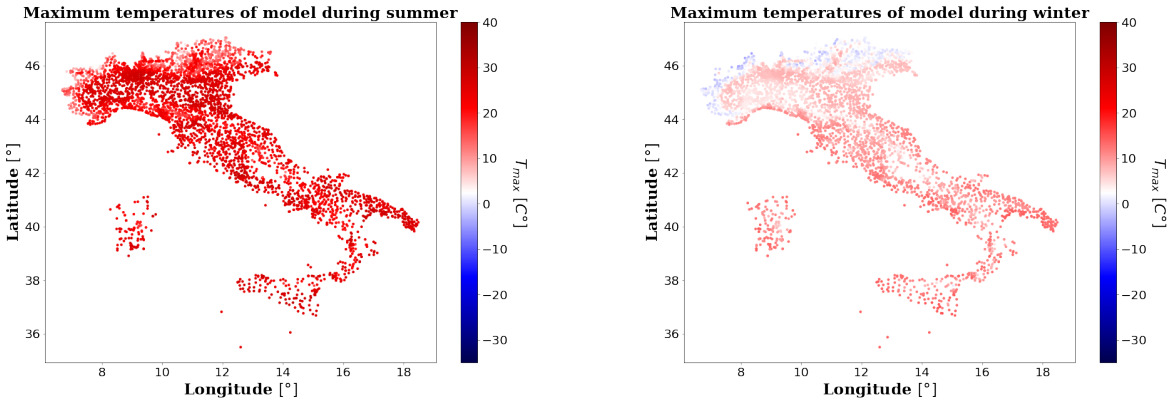


Figure 2.15: Distribution of T_{max} given by stations and model.

situations. If for T_{min} we have that range where the model overestimates and some where it underestimates with multiple passages from a condition to another, for what regards T_{max} we have a clear transition. Before almost $30C^{\circ}$ the model seems to have more values with respect to the station while there is a clear change after that value meaning that the model in not able to recognize very high or very low values of T_{max}. For consistency we also show the two days presented in figure 2.14 from the point of view of the model in figure 2.16. Again we cannot almost distinguish the model from the station only by looking at that plots.



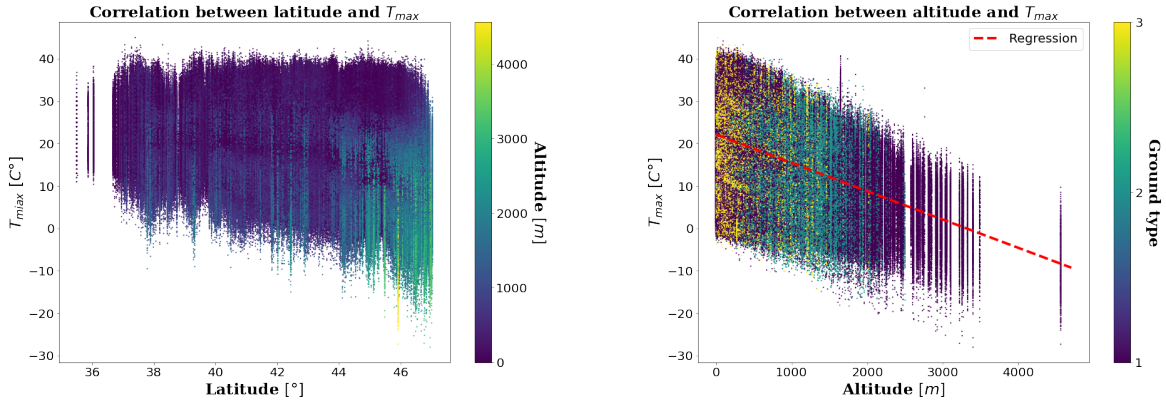
(a) (b)

Figure 2.16: Typical behaviour of the maximum temperatures during summer (a) and during winter (b) reported by the model

To prepare the maximum temperature dataset a preprocessing technique equal to the one introduced for T_{min} is adopted. As a reminder we select as bad data each event in the dataset that reports an absolute value of the difference between T_{max} from stations and model greater than 15 C°. For the station that reports a value of exactly 0.0 C° we put a more restrictive condition and we keep only value that reports an absolute value on the difference between T_{max} from stations and model lower than 2 C°. The filtering procedure ends up with a clean dataset that reports almost 4 600 000 events.

2.8.2 Dependency of maximum temperature from other variables

With a clean dataset we want to present the correlation between the maximum temperature and other variables as made for the minimum temperature. As made before we will present the information using scatter plots and again some relation may directly emerge from this visual section, others, more hidden, cannot be caught by a simple 2D plot.
The first two variables that we want to compare with the maximum temperature are the latitude and the altitude (again the plot with the comparison of longitude and T_{max} is not really interesting due to the low cover in terms of longitude of the considered territory).



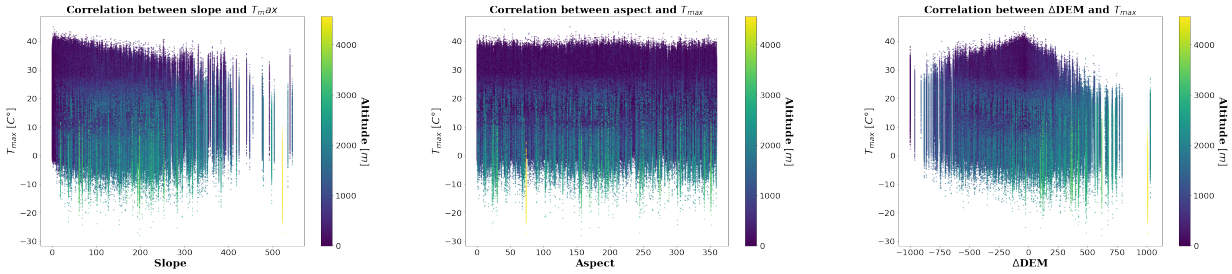
(a) (b)

Figure 2.17: Scatter plot of the events where T_{max} is presented as function of (a) latitude and (b) altitude.

Looking at the plots we can comment that the two behavior are expected. In fact we have that exploring higher latitude we find more rigid temperature due the mountain encountered. This behavior is confirmed also from the relation that emerges from figure 2.17 b, as the altitude increase, lower temperatures are explored. Another brief comment on this last plot emerges if we compare this graphic with the respective made for the minimum temperature. The slope of the fitting regression in the case of T_{max} is lower than that parameter for T_{min} (-0.0052 for the minimum temperature and -0.0066 for the maximum); we can interpret this signal as a more rigid change of maximum temperature with the constraint of looking to a mean behavior in fact the points are really spread around the regression line.

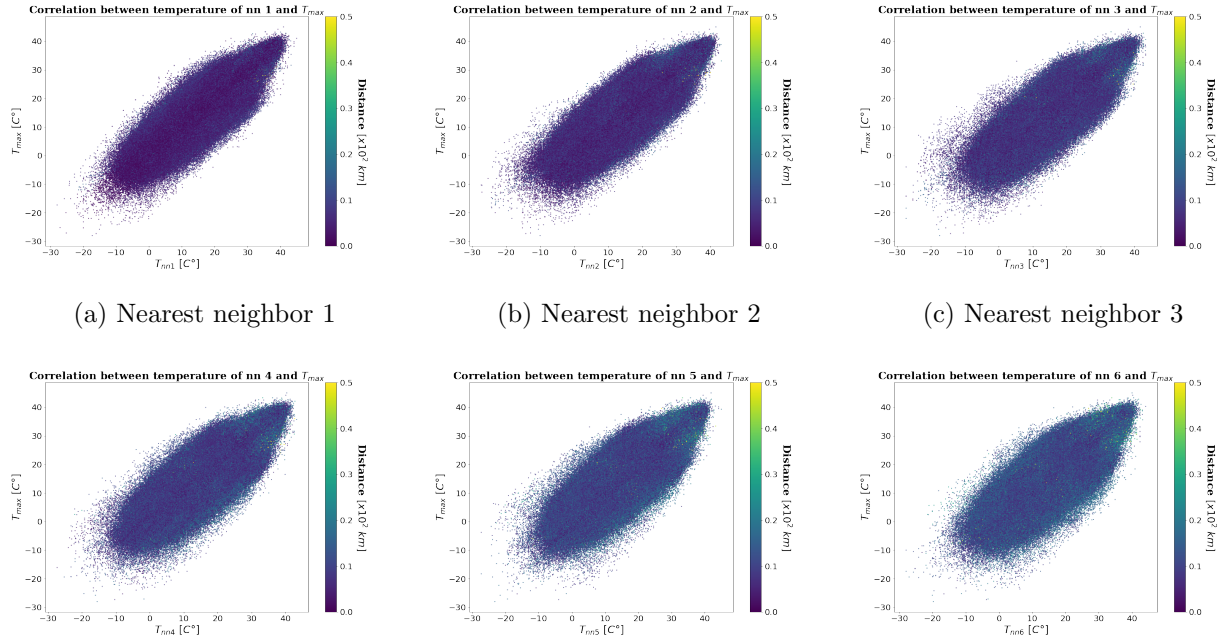
Figure 2.18 shows instead the plots of the maximum temperature as a function of slope, aspect, and ΔDEM respectively. The same conclusion drawn for the minimum temperature applies to the maximum temperature with some exceptions. The most important is the role of the aspect; despite the scatter plot representing T_{max} as function of this variable does not show interesting relation at first glance, we expect0 that in a given day, if we evaluate two similar points with different exposures, the one exposed at north may have a lower maximum temperature. For this reason we do not take off the information of the aspect from the inputs of our algorithms that evaluate T_{max} and as we will see in the analysis section we have a better MAE considering the variable with respect to not considering it. The second important difference with T_{min} is that this time we use variables such as slope, aspect and ΔDEM to retrieve useful information but we do not intend these variables useful for explicit tasks such as the identification of thermal inversions.
Since also the algorithms for the spatialization of the maximum temperature exploit information on nearest neighbors, the last plots presented are the relation of T_{max} of a considered point with its nearest neighbors. Same conclusions of T_{min} can be done. Figure 2.19 shows this information.

With this we close the section that regards information of variables related to the temperatures. In the next chapter we will return to these topics especially when we have to pass this information to



(a) (b) (c)

Figure 2.18: Correlation with other morphological variables: (a) Correlation between maximum temperature and slope. Like minimum temperature at high slope the range of variation seems to be reduced. (b) Correlation between maximum temperature and aspect. So signals of correlation appear. We will see that exploiting a neural network the aspect is useful. (c) Correlation between maximum temperature and ΔDEM. We have more or less the same behaviour of the minimum temperature. At the extremes of the plot the field of variation of the temperature in restricted with respect to the middle .



(a) Nearest neighbor 1 (b) Nearest neighbor 2 (c) Nearest neighbor 3

(d) Nearest neighbor 4 (e) Nearest neighbor 5 (f) Nearest neighbor 6

Figure 2.19: Correlation between the maximum temperature and the values reported by the neighbors stations

the different algorithms. As we will see for example neural networks perform better if we modify their inputs and this way of processing data is different from the preprocessing needed by decision trees. We leave these details to the dedicated sections and we now move to the next topic: the precipitations.

2.9 Rain

One other very important meteorological variable which cannot be disregarded in the generation of meteorological grids is the rain. Since water is at the base of life, we have that not only agriculture, insurance, transport and other fields introduced with the temperatures are interested in information on rainfall. Precipitation is one of the most basic meteorological and hydrological elements and has intricate tempo-spatial variability. Accurate information about the precipitation distribution in space is the basis for scientifically understanding global or regional changes in processes involving water and its associated materials and energy, which is of great significance for the promotion of meteorological and hydrological monitoring and forecasting to enhance the capability to cope with natural disasters and optimize water resources management [20]. Therefore, spatial estimation of precipitation has been a vital scientific issue of common concern in many fields, such as meteorology, hydrology, ecology, geology, and so on . In recent years, numerous efforts on quantitative rainfall spatial estimation have been made. At present, the number of rainfall spatial estimation methods available is relatively high, and new methods are still proposed continuously. However, from the information source perspective, it appears that the development of rainfall spatial estimation has generally experienced three stages of development [21]. The first stage was spatial interpolation based on the measurement of rain gauges (what we call as stations), which was the traditional way to obtain distributed rainfall information. However, the density as well as the spacing required for this method restricts the results obtained by rain gauges. The second stage was remote sensing retrieval and atmospheric model calculation, both of which are areal inherent and can obtain spatial continuous rainfall across a certain region. Since the 1980s, remote sensing retrieval and atmospheric model calculation have gradually become important methods for rainfall spatial estimation. However, due to the influences of remote sensing instruments, atmospheric model capacity, and other factors, their uncertainty is relatively prominent. Since the 1990s, rainfall spatial estimation of precipitation has developed to the third stage, that is, the stage of multi-source data merging. Recently it has become one of the most important hotspots in meteorology and hydrology and can be used to obtain better precipitation information by integrating various kinds of information, such as gauge observations, remote sensing retrieval, and atmospheric reanalysis estimates [22] [23].

This is the starting point of our analysis. This third stage is also the main difference that we want to introduce in the process of spatialization regarding rain with respect to the process presented above for the temperatures. We in fact have that while for T_{min} and T_{max} we want to retrieve some correlations between the values of the temperatures reported by the considered point and some terrain variables as well as information on nearby stations, for what concern the precipitation we do not expect a strong correlation with the type and the morphology of the ground. On the other hand for this type of variable we have a new great source of information: the radar.

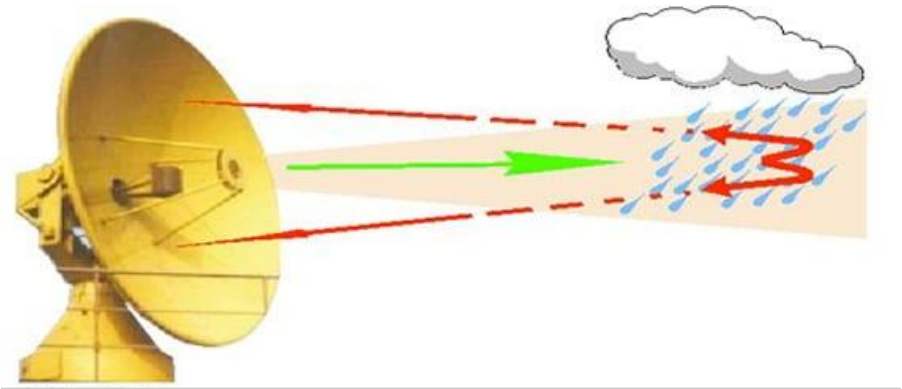
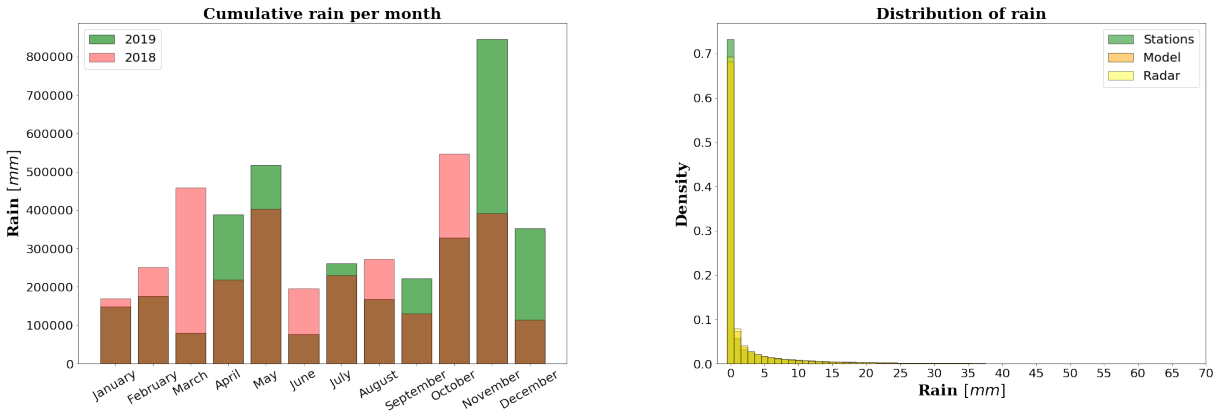


Figure 2.20: Graphical representation of how a radar works for detects precipitation.

Radar is the acronym of "**RA**dio **D**etection **A**nd **R**anging". The principle of operation of radar system consists in producing electromagnetic waves in the radio domain (in general also other domain can be used) and that radio waves when encounter some objects, reflects back and return to the receiver, giving information about the object itself. Radar finds application in a lot of different fields, starting from military usage to the meteorological one. To give an introduction to the measurement of rainfall by radar, its work is to evaluate the intensity involves the determination of the radar energy scattered back to the radar aerial by the populations of raindrops within a number of volumes, each volume being defined by the width of the radar beam and the length of the radar pulse. The reflectivity factor Z is related to rainfall intensity R through the equation $Z = aR^b$ where a and b are empirical "constants" which depend on the size distribution of the raindrops [24]. Although radar necessarily observes volumes above ground level and increasingly so at longer ranges its measurements are usually interpreted in terms of the intensity of rainfall at ground level. The accuracy of radar is normally assessed against raingauges. Rain is a highly variable phenomenon and so raingauges are not a good way of measuring rainfall over an area if we use that information from station alone. This is for the most related to the low density of the station network on a territory. Nevertheless they remain the standard point for measurements and as we will see we use precipitation from raingauges as target for our supervised algorithms and we still consider that information as truthful.

As we will see we still use some information of the ground but we do not expect a great correlation between these variables and rain, but as for the slope and for ΔDEM in the temperature, the use of this information is for the characterization of the stations.
Returning to the merging of various sources of information we will see meteorological model, information from radar and precipitation of neighbors as for the temperatures. We can so move to the visualization of the data regarding precipitation.
Figure 2.21 gives general information on rain. The cumulative rain over all gauges for two years of example are showed in the first pictures. As we can see the variation of the precipitation is very high (see March) but in general we have the most rainy months are in autumn. The second subfigure (2.21b) shows the typical distribution of rain retrieved by stations, model and radar. As we can see a huge number of events report 0 mm and this is not a surprise because the climate in our country favors non-rainy days.

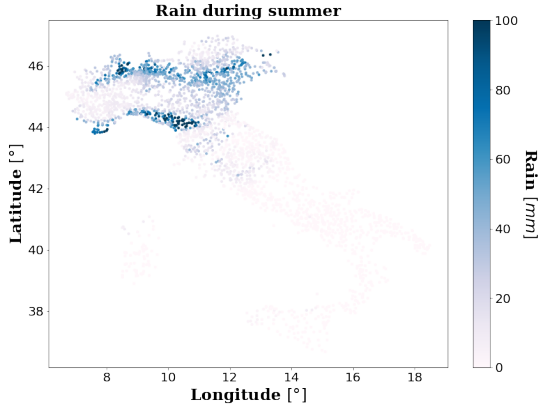


(a) (b)

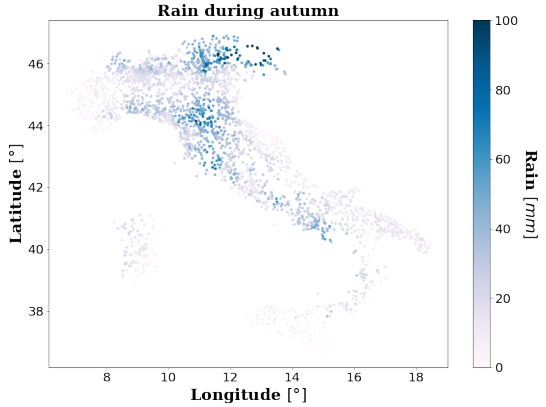
Figure 2.21: Cumulative precipitation per month (a) and distribution of rain(b)

Looking at that distribution we can first of all comment that both radar and model fallow the behavior of the stations but the abundance of events without rain makes the task hard from a machine learning point of view. Due to the fact that this dataset seems to be unbalanced we may have some problems during the learning phase. One of the solutions that we will present in the analysis section regards the oversampling of fake events to balance the dataset. We can anticipate that the technique of merging information and the usage of a deep neural network can overcome this problem also with the original

dataset. Another problem that we have considering the precipitations is the fact that it interacts with others variables such as snow and hail that can distort the measurement of an instrument. A common example is that an unheated raingauge reports a value of precipitation that comes from some snow fallen the previous days that melts and give a bad input to the sensors of the station. Reason like that makes the study of rain a complex task. The complexity is reflected also to the validation procedure already implemented by Radarmeteo that from 2018 to the end of 2020 gives us about 3 760 000 events, almost one million less with respect to the temperatures. Nevertheless two other important filtering techniques of the dataset are performed because as already said if on one hand we need a lot of data to train algorithms, on the other hand we need high quality data. The first selection is made based on the value of T_{max} reached during the day to overcome the problem with the snow described before. All the events that report a maximum temperature lower than $0.5C°$ are discarded. The second preprocessing is done with a neural network and the explicit description of the filtering procedure will be exposed in the analysis section. Despite this, all the plots presented in this section will show events of rain already filtered. Another consequence of the complexity of the task is that we choose only neural networks to analyze this variable but different solutions based on this algorithm are tested.
Returning to the data visualization topic we can see from figure 2.22 two typical situation of rains during summer and autumn.



(a)



(b)

Figure 2.22: Typical situation of rain during summer (a) and during autumn (b)

As we can see is more common in summer to have isolated precipitations, also with heavy rain while in autumn we generally find periods of more days with constant precipitation of lower intensity.

2.9.1 Dependency of rain from other variables

The first comparison is shown in figure 2.23 displays relation between altitude and rain. From the plot we can see that for values close to the upper limit we find lower intensive precipitation.

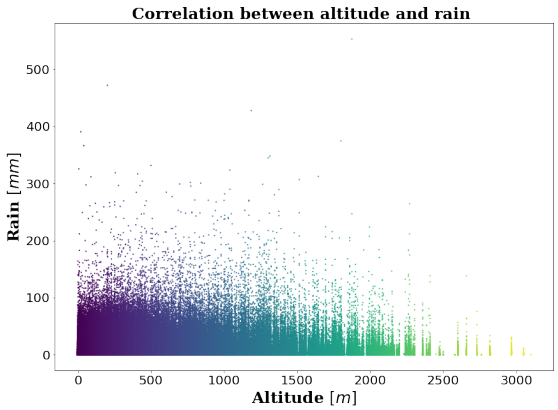
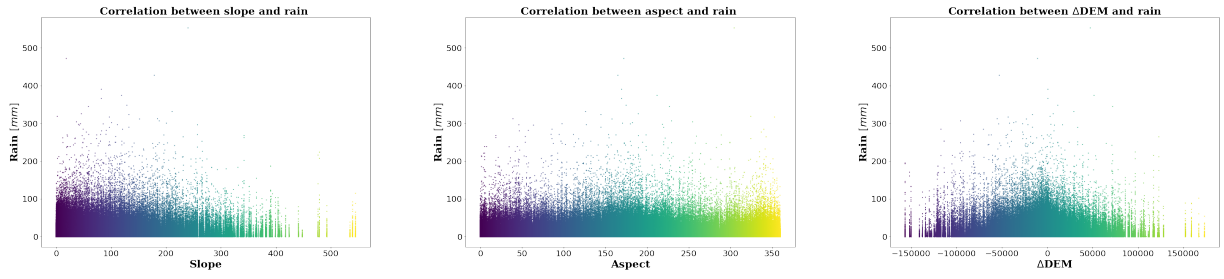


Figure 2.23: Correlation between altitude and rain.

Just for completeness we present also the scatter plot representing the relation between rain and the other terrain variables in figure 2.24. For the same reasons presented in the section on temperatures we have that high values of the slope are more common at high altitude and the profile of rain in the plot that compares this variable with the slope is similar to the profile found in plot of the relation with altitude. The same behavior can be explained for ΔDEM while aspect at first glance does not show correlation with the precipitation. Despite this behavior of the aspect, as for the maximum temperature we still want to use it for the characterization of the station and we manipulate this variable to obtain discrete information from a continuous one. As for the maximum temperature in fact we pass to the neural network a value of aspect based on exposure to one of the four cardinal points.



(a) Slope (b) Aspect (c) ΔDEM

Figure 2.24: Correlation between the rain and (a) slope, (b) aspect and (c) ΔDEM

We can now move to more interesting comparisons that in fact reflect the real essence of the work behind the rain: the comparison with the multi model that we want to merge to achieve the final goal. Starting from the model we have that figure 2.25 shows the behavior of the rain reported by stations as a function of the rain reported by the model (we will call in the plots rain the events from the point of the stations). In particular we have that a general behavior of the model is to follow the bisector that represents the optimal predictor but if we focus on specific months we find relevant differences. In fact the performance of the model during autumn seem to be higher than the performance reached during summer; as we can see from the last two plots of this figure we have a situation of low spread around the perfect predictor line in autumn while in summer the situation seems to be the opposite.

This fact is not a surprise because of the characteristics of summer precipitation compared to autumn precipitation. Since during summer is more common to have precipitation related to also very intense isolated convective phenomena, the complexity of the equations that make up the model may drive the model itself to fail the evaluations of correlated variables, determining a bad prediction for the entire dynamics of the event. For example the failing in the evaluation of the exact position of the convection trigger of few kilometers leads to predict a completely wrong behavior of the precipitation. On the other hand in autumn is more rare a situation of isolated convective phenomena and so the model is able to predict first of all the exact position of the precipitation and then the relative intensity very well.

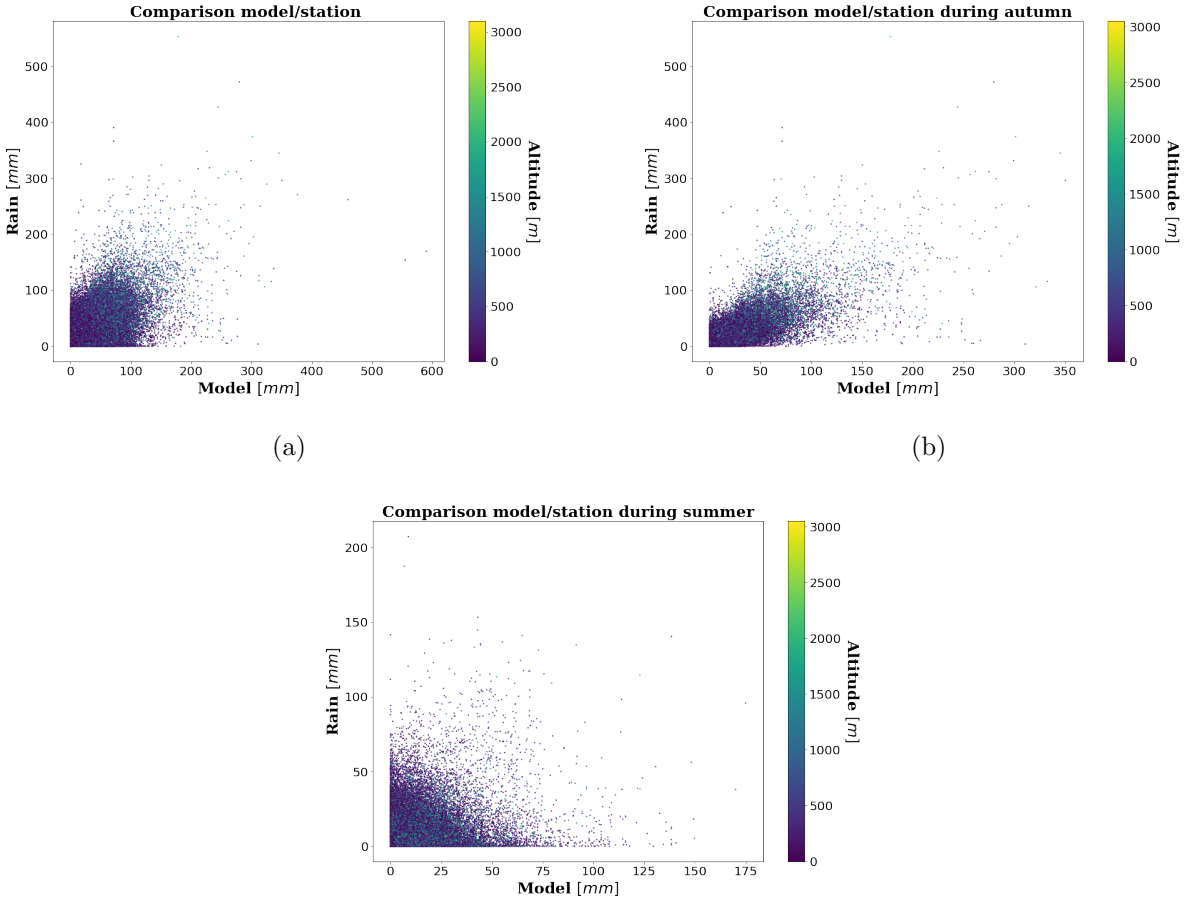
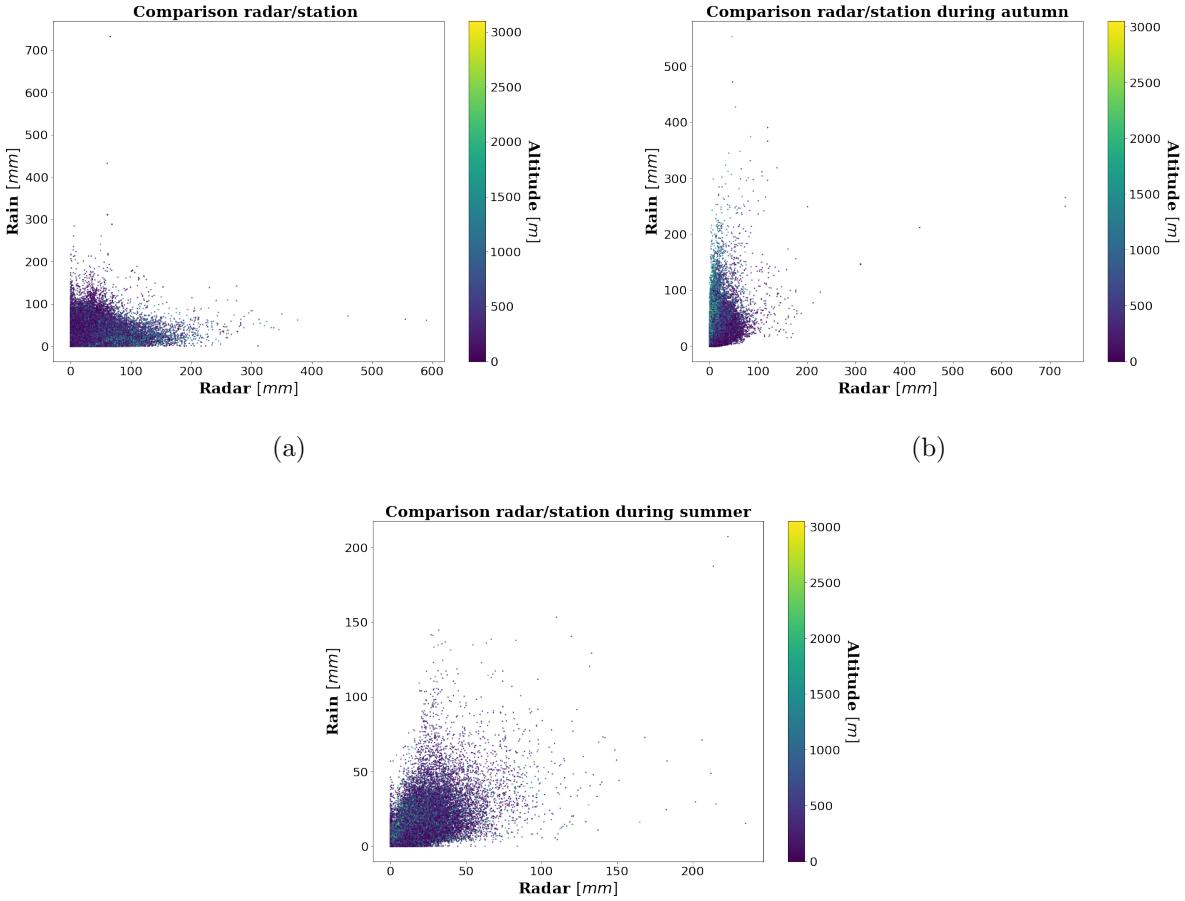


Figure 2.25: Correlation between the station and model of all the events in the dataset (a); during autumn (b) and during summer (c).

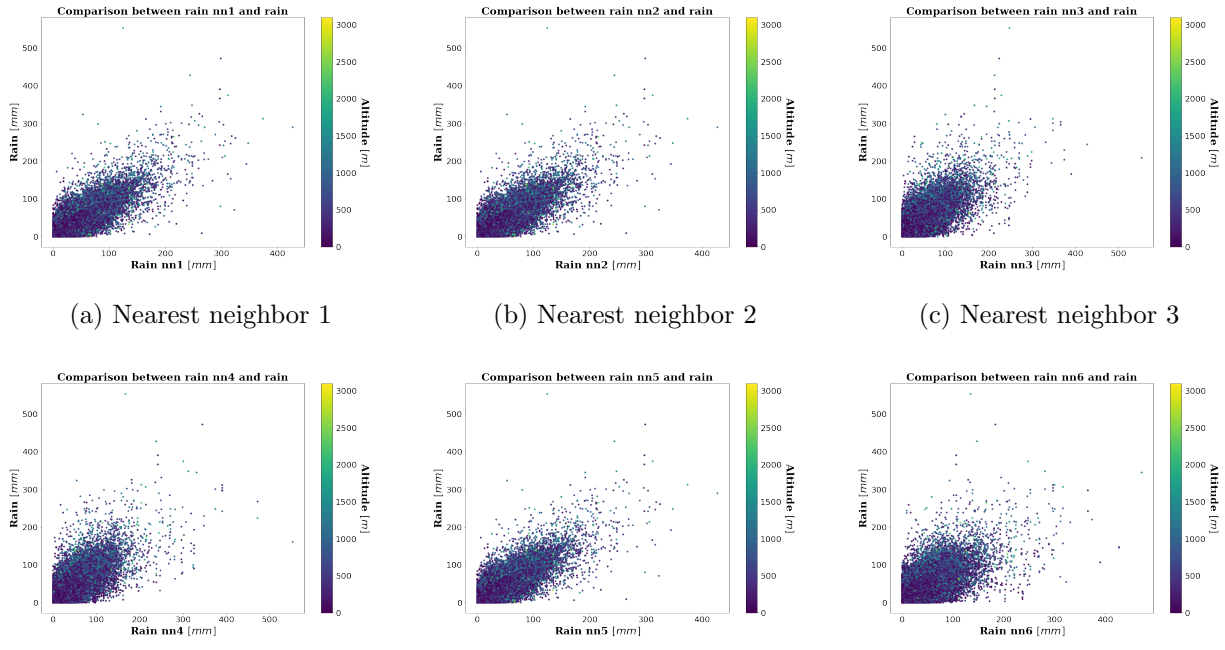
As we will see in the results section we have that in general the model will perform better also with events located in the Alpi or in the high mountains with respect o the radar due to the nature of the mountainous precipitations.
So the second source of information that we want to introduce is the radar. From figure 2.26 we have an opposite trend with respect to the model presented in figure 2.25. This time in fact during autumn the radar is not able to see a lot of rainy events while during summer it is able to catch also very intense events and has a general better prediction behavior. For comparison we present also the values of rain reported by the nearby stations in figure 2.27. The same conclusions of the case of temperatures can be done.



(a) (b)

(c)

Figure 2.26: Correlation between the stations and radar of all the events in the dataset (a); during autumn (b) and during summer (c).



(a) Nearest neighbor 1 (b) Nearest neighbor 2 (c) Nearest neighbor 3

(d) Nearest neighbor 4 (e) Nearest neighbor 5 (f) Nearest neighbor 6

Figure 2.27: Correlation between the rain and the values reported by the neighbors stations

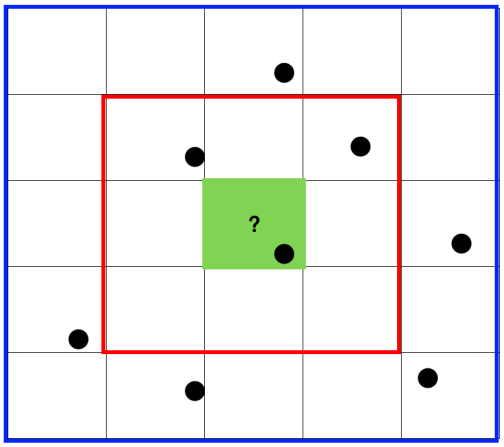
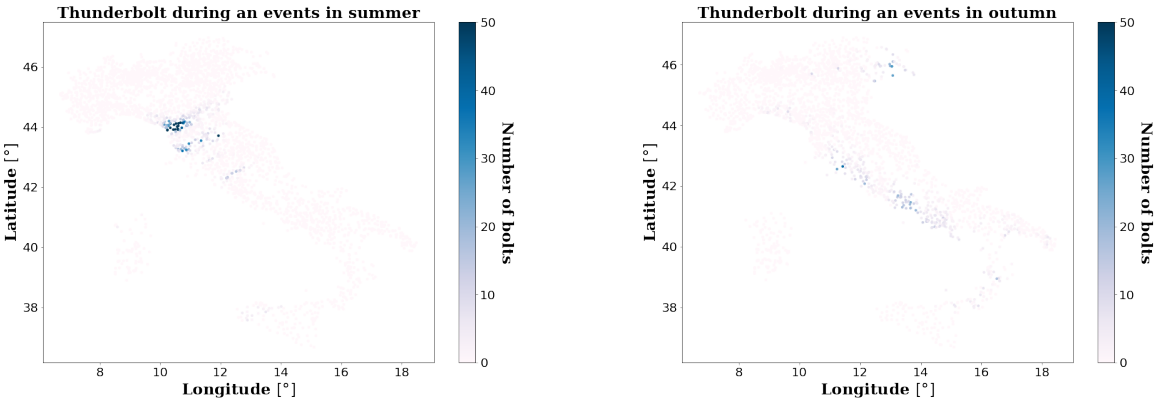


Figure 2.28: Schematic representation of the construction of the thunderbolt dataset. For the green point under examination we will report a number of thunderbolts in the 3×3 area of 3 and a value of 8 for the 5×5 area.

Despite the main idea that we will follow in the construction of the grid, the multi source merging, we will use a neural network and inputs other than the sources we want to merge can be passed. As already mentioned, terrain variables are used but a new variable highly related to precipitation is based on thunderbolts. Since from a meteorological point of view is more difficult to identify very tight thunderstorms, especially in summer, this variable can give good hints. The introduction of this variable is related to the fact that may happen that the radar sees a point of high precipitation but in the surrounding area there are no stations available for measuring that event and the nearby station report a significantly lower value due to the small area affected by the storm. In these situations , that arises especially during summer, the model may fail the prediction and it is hard to assign a real value of precipitation to the unknown point. For these reasons it is useful to introduce information on thunderbolts. Since Radarmateo exploits a national network of thunderbolts recognition systems we have the raw data as scattered points in a map. We manipulate this information by extracting two datasets that respectively incorporate the number on bolt inside an area surrounding the unknown points covering a 3×3 portion of the grid or a 5×5 portion as in the scheme of figure 2.28. Figure 2.29 shows instead the spatial distribution of bolts during a rainy event in summer and autumn respectively. The same days of figure 2.22 is presented to make the comparison. As we can see summer precipitations are often associated with a high number of lightning strikes while autumn and winter rainy events are not.



(a) (b)

Figure 2.29: Typical situation of thunderbolt during summer (a) and during autumn (b).

With this we conclude the chapter dedicated to the data and the manipulation of the information to pass as input to our algorithms. There will be other occasions when we will return to this topic in the course of the document for example talking about the preprocessing of data or about the algorithm to clear the rain dataset.

Chapter 3

Analysis

In this chapter we are going to present the second part of this work. Once that the datasets are ready, we can use the information to train different algorithms and here we will delve into the technical details of these training. To follow the workflow we will split this chapter in three sections like for the data presentation that correspond to the different variables to analyze. For each variable then different type of algorithms was tested and as we will see, for the temperatures we are going to present multi linear regressors, boost decision trees and neural networks with more attention to the latter. For the rain instead we drop the first two models and develop three different approaches with neural networks.

3.1 Minimum temperature

3.1.1 Neighbors

We want to start the discussion of the techniques used for the training that regards minimum temperature with the concept of distance. As emerged from the previous chapters, our analysis (independently from the variable under consideration) exploits information retrieved by the neighbors of a given point. In particular as suggests [2] we choose six neighbors with relative information on temperatures, rain and other variables which we will specify in each application. The problem that arises here is a computational problem. Since there is the necessity to produce grids of a given area in a time window that can be large, we have to require to our algorithms to be fast as well as accurate. During the elaboration emerges that the bottleneck of the process (once that the algorithms were trained) is the evaluation of such neighbors. The reason is that in meteorology, like other fields, the natural unit of measure for the position of the points is the system latitude/longitude that we already use to produce our plots in chapter 2 and which are expressed in degrees. There are different ways to calculate the distance between two points in lat/lon coordinates but a proper way is to evaluate it in the surface of the heart (if we consider a sphere) that takes the name of spherical distance. If we call the two point under study $(P_1 = X_1, Y_1)$ and $(P_2 = X_2, Y_2)$ where the coordinates is given in degrees, X_i corresponds to the latitude, Y_i is the longitude and we call $La_1 = \frac{X_1\,\pi}{180}$, $Lo_1 = \frac{Y_1\,\pi}{180}$ (and the analogous for the second point), the spherical distance in kilometers is given by:

$$d = r \cdot \arccos[\cos(La_1 - La_2) - \cos(La_1) \cdot \cos(La_2) \cdot (1 - \cos(Lo_1 - Lo_2))] \tag{3.1}$$

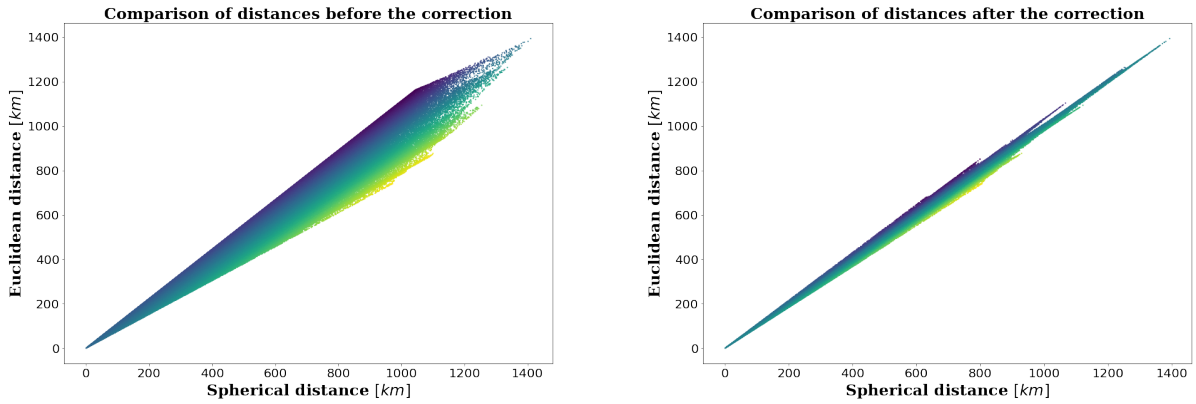
with $r = 6371\ km$ the radius of the earth. The goal is to find an approximation that exploits the euclidean distance to be able to bypass the calculation of the trigonometric functions that are present in the general formula which constitutes a huge computational weight.
The idea is to use a distance like:

$$d_{approx} = 100 \cdot \sqrt{(\Delta X \cdot a)^2 + (\Delta Y \cdot b)^2} \tag{3.2}$$

with a and b empirical constant that minimize the difference with the spherical distance and the 100 is needed to have the distance in kilometers. To evaluate these parameters we use a brute force approach divided into steps:

- we generate 10 000 random points with latitude and longitude in a range that cover Italy.
- From these points we generate all the possible combinations of two elements.
- We evaluate the distances starting from a and b in a given range and we compare the results.
- we iterate the last step with a smaller range of the parameters until convergence.

From this algorithm we find that the optimal values for the parameters are: $a = 1.111900$ and $b = 0.828400$. Without this correction we have that the MAE of the euclidean distance is about 52 km while using the correction we have a MAE of about 6 km. At first glance it results high but we are looking at points taken randomly over Italy. In our application, the distribution of the distances of the neighbors from our points is of the order of the km (only the sixth neighbor is able to reach 10 km) and so the relative error that we are introducing is negligible. Quantitatively before the correction we have an absolute relative error of almost 10% while after the correction we reach 1%. Figure 3.1 shows the visual relation between distances before and after the correction. We will explore distance highly under 100 km (see figure 3.2) where the correction has a smaller error.



(a) (b)

Figure 3.1: Scatter plots of the euclidean distance as function of the spherical distance before (a) and after (b) the correction.

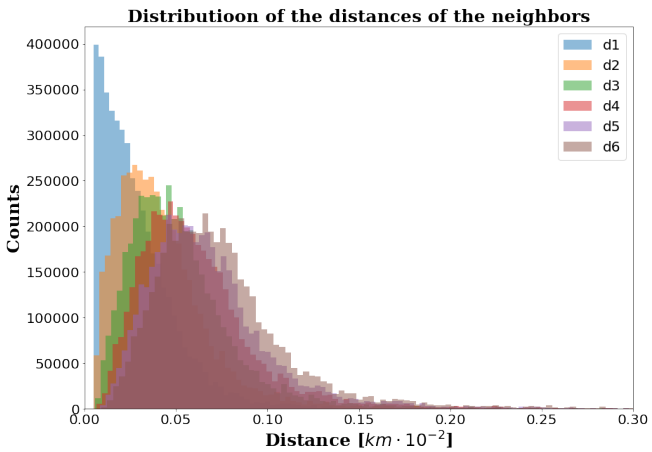


Figure 3.2: Distribution of the distances of the neighbors.

3.1.2 Multi linear regressor

The first algorithm that we implement is a multi linear regressor. Since this is a very basic algorithm, no particular preprocessing is needed and we can use the data once that we clear our dataset based on the minimum temperature from the model. We implement the regressor in Python and in particular with the already mentioned library "scikit-learn". Since we want to use this model as a comparison and due to the low complexity of the model itself, we do not have to tune any parameter and we can pass the inputs for the training. The variables that we select to pass are divided into three categories:

- from the target station we use geographical information (latitude and longitude), the altitude, the slope and the difference of DEMs.
- The value of the minimum temperature from the model.
- Information from neighbors of minimum temperatures, distance, altitude and model.

We choose these inputs based on the performances. In fact passing the information of the aspect and other information on neighbors such as land cover and so on, we do not decrease the MAE of the predictor meaning that the algorithm is not able to use these additional variables. The minimum temperature reported by the station is passed as target value for the training.
With this algorithm we reach on a test set an error based on the MAE metric of 1.2 $C°$. The Advantage of this simple model is that we can print the weights given to each input to make some comparisons. The highest weights are given to the temperature of neighbors, the model and then the other variables meaning that in some way the model makes a mean of this information on temperature based also on some regression with geographic variables (for example altitude).

3.1.3 Boosted decision tree

The second and more complex algorithm that we use to learn the dependence of the minimum temperature of an unknown point with terrain variables and information from neighbors station is the boosted decision tree. In this case we can exploit the complexity of the model to extract more information from our inputs. In this case in fact the optimal model has as inputs the following variables:

- for what regards the station under study we pass geographic information (latitude, longitude), altitude, slope, ΔDEM and ground type.
- For each one of the six neighbors we use as inputs the minimum temperature, the distance, altitude, ΔDEM, the minimum temperature from the model and the land cover.
- In addition we use also the value of the minimum temperature from the model for the considered point and the month.

The characteristics of the boosted decision tree allow us to use as inputs different type of variables without any preprocessing. In particular the XGBoost (name from the library used) developed receives month and ground types that correspond to categorical variables but we can pass this information as continuous. We will see that the neural network is not able to use these variables without a proper transformation.
The XGBoost is the first model that needs to be tuned from a machine learning point of view. With this we mean that the model has many hyper parameters to select and this selection has an empirical base. The first two parameters that in this case represent also the two most important ones, regard the structure of the trees. We have to choose in fact how many estimators we want to use for the training and the maximum depth of each of these trees. The selection of these parameters follows a grid search approach taken by step: we give a prior number of estimator (starting from low unity number) and for each models we train it with some different number of trees. The next step consists in start with a higher number of estimators if the trend of the error is to decrease or try different number of trees. The next parameter to select is the learning rate. The default learning rate is 0.3 so we perform a random search of this parameter using a simple structure (because of the computational cost of the training of a complex forest) looking for values of this parameter near the default value and

then tuned near the value that gives the best performance. A learning rate of 0.2 gives us the best performance independently from the structure used. For the loss instead we make a regression with a combination between squared loss and an L2 normalization on the weight ($\lambda = 1$ on equation 1.5). Other computational parameters are left as default due to the computational cost of the training. As we can see from figure 3.3 that reports the grid search results from a point of view of the MAE evaluated on a test set, the steps of the grid drive us to select a max depth of 15 that outperforms all the other models. For what regards the number of estimators, the trend for all the depths evaluated is to get a lower value of the MAE as we increase the complexity (meaning that we add more estimators). The last three green points in the right part of the plots correspond respectively to a model with 1500, 2000 and 3000 estimators. The MAE reached from these models is respectively 1.076858, 1.076404 and 1.075972 (all the MAE are reported in $C°$). So the gain we have in performance is not comparable with the computational time and the computational power that we need to train bigger models. For this reason we select as best XGBoost the one with 2000 estimators. The final MAE on a test set is so 1.076. As we will see in the results section it is a very good results. The performance of these models in fact is comparable with the one obtained with a neural network but this is not a big surprise since for this type of application, the XGBoost is an algorithm largely applied.

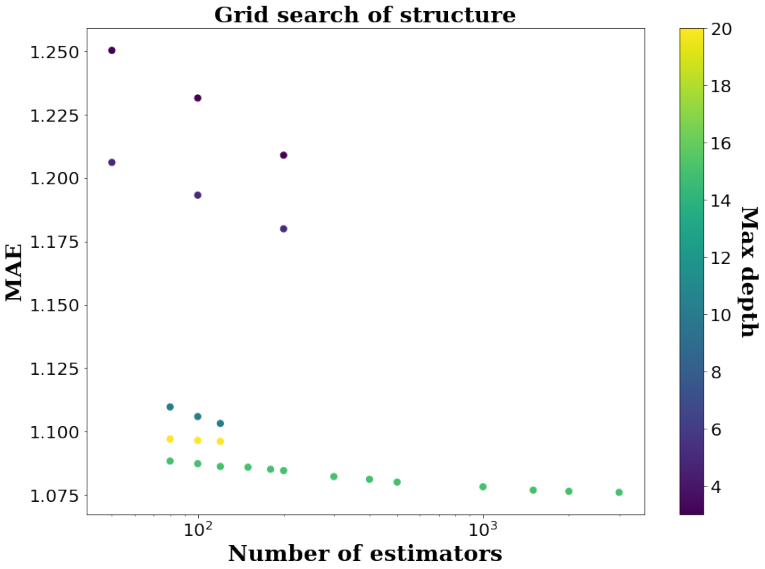


Figure 3.3: Grid search of the structure of the XGBoost.

3.1.4 Neural Network

We will now present the main model that we base on: neural network. Before start to talking about the tuning of the hyper parameters that govern the functioning of these models, we have to make a brief explanation on how to pass the variables to the predictor. First of all we present the variables that we use as inputs for the model; as we will see we use the same inputs of the XGBoost. The categories are the following:

- geographical and morphological variables: latitude, longitude, altitude, slope, ΔDEM and land cover.
- From neighbors: minimum temperatures, distances, altitude, difference of DEMs, minimum temperature from the model and land cover.
- In addition we give information on the minimum temperature from the model in the considered point and the month of the event.

As for the XGBoost, we have some continuous and some discrete variables. The difference of the

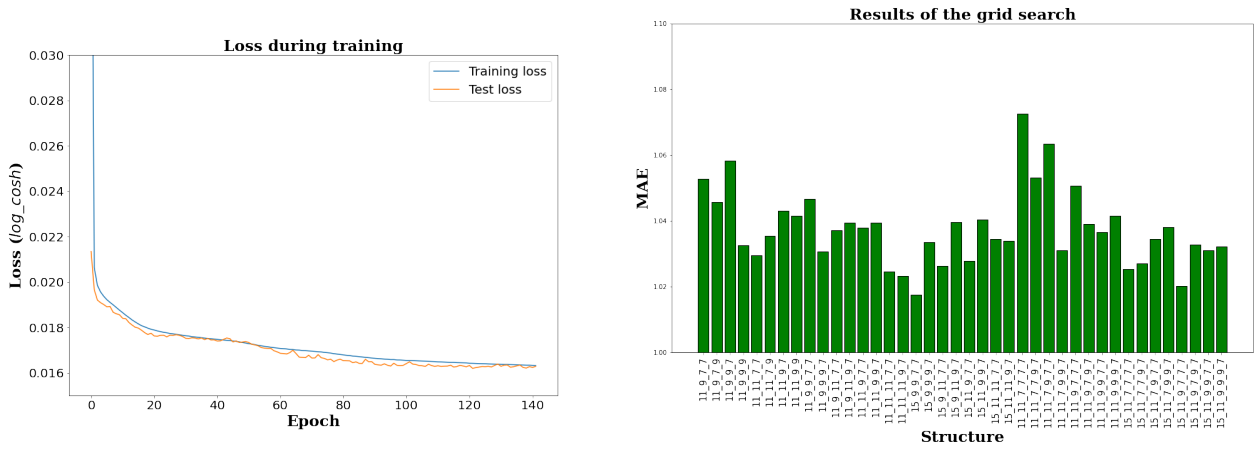
neural network with the previous model is that to reach a better performance we need to process again the dataset and both the category of variables will be modified. In principle we have that from an empirical point of view, neural networks work better if the inputs and the targets are small values, close to the units. For this reason we manipulate the continuous variables in this way:

- for the variables distributed as a Gaussian we make a standardization of the values, meaning that the raw information x_i become: $\hat{x_i} = \frac{x_i - \bar{x}}{\sigma}$ where $\bar{x}$ is the mean of the distribution and σ is the variance.
- For the variables that have a distribution completely different from a Gaussian we make a normalization of the values between zero and one. In the normalization we care about out layers that can appear in the grid evaluation (an example in the altitude, we normalize that variable between zero and the highest value over Italy).

For what regards information on the month and land cover (the categorical variables) the technique of one hot encoding is exploited since we can't pass unprocessed discrete information to neural networks. The one hot encoding consists in build a vector of dimension n, where n is the number of possible values that the variable can assume. Then we associate each possibility to a vector with a 1 in a given position and all 0 in the others. We end up with a vector associated with each possible value of the discrete variables. Once that the dataset is ready we can move the tuning topic.
All the neural networks are developed using "Keras", a library for machine learning for Python. This library gives us the freedom of choose each parameter and each strategy of learning. Also in this case the learning is divided in different sections. To approach the problem we make a first learning to get an idea of the order of magnitude of the values to give to hyper parameters. We start from the optimizer. Some random architectures are tested and we evaluate different types of optimizers, all the optimizers can be grouped into the Adam and SGD categories with the Adam that outperform the SGD in both computational time and performance. Once that the optimizer is selected we use the same approach to determine the loss function to use. An MSE, a MAE, a *tanh* and a *log_cosh* was tested [1]. The final metric that we use is the MAE of the minimum temperature and surprisingly using the *log_cosh* we get the best performances in terms of MAE of the test set. The activation function that performs better is evaluated between the *Relu* function, the *Sigmoid* function and the *tanh* function. *Relu* outperform the others in all the test. In addition we let the output neuron with a linear activation to be able to obtain also negative values. A random search was performed to get the best learning rate and we set this parameter to 0.0005. Adam optimizer then intrinsically perform and adaptive learning based on momentum so the learning procedure change during the epochs. The default values for the decay of the first and the second momentum are used. Once that these parameters are chosen we can move to the architecture of the networks. In this case a grid search over the number of layers and number of neurons in each layers are performed. The grid was performed from 3 layers (input, hidden, output) up to 8 layers. For each training we let the algorithm run for 300 epochs but an early stopping was implemented. In particular if the test loss does not decrease for 20 epoch, we stop our algorithm to avoid overfitting. Figure 3.4b show the performance reached from the grid search. The best model has 5 hidden layers with the following number of neurons $(input\ len, 15, 9, 9, 7, 7, 1)$ which reach a MAE of $1.017C°$. We want to give a comment also on figure 3.4a: as we can see the loss function of both training and test set decrease during the training; this means that the algorithm is learning without overfitting. More complex structure tested in the first approach present a similar behavior for the training loss while the test loss has a significantly higher value, index of overfitting. We also try to add some regularization such as dropout layers and L2 terms on weights but since our models seem to not overfit, we just obtain lower performance with this type of regularization.

The last comment that we want to make before move to maximum temperature regards figure 3.4b. Looking at the y-axes one can make a consideration that the variability is not that high and in practice it is possible to take each model and obtain the same results. As we will see in the results section, this is wrong. In fact if we just look at the MAE we find similar values and similar mean behavior but since our main goal is to construct grids and in particular for the minimum temperature we are

[1] see https://keras.io/api/losses/ for the details of the losses



(a) (b)

Figure 3.4: Loss function during the training for the test and the training set (a); grid search results on structure (b).

interested in the phenomenon of the inversion, a model with a MAE of 1.1 for example is not able to recognize the inversion as well as the best model even if the general behavior remains good.

3.2 Maximum Temperature

3.2.1 Multi linear regressor

The training phase for what regards the maximum temperature follows the same structure of the minimum temperature. In this case we expect a general better performance because maximum temperature does not have strange behavior such as thermal inversion. The input variables that we pass to the algorithm are the same of the minimum temperature with the addition of the aspect because we think that the correlation of this variable with the solar exposition can be useful. Namely the inputs are:

- longitude, latitude, altitude, slope, difference of DEMs, aspect.
- From the neighbors: maximum temperature, distance, altitude and maximum temperature from the model.
- Maximum temperature from model for the unknown point.

With this inputs we get a MAE on the test set of $0.895C°$. As we expect this error is lower than the correspondent for the minimum temperature an it is an index of a less complex task.

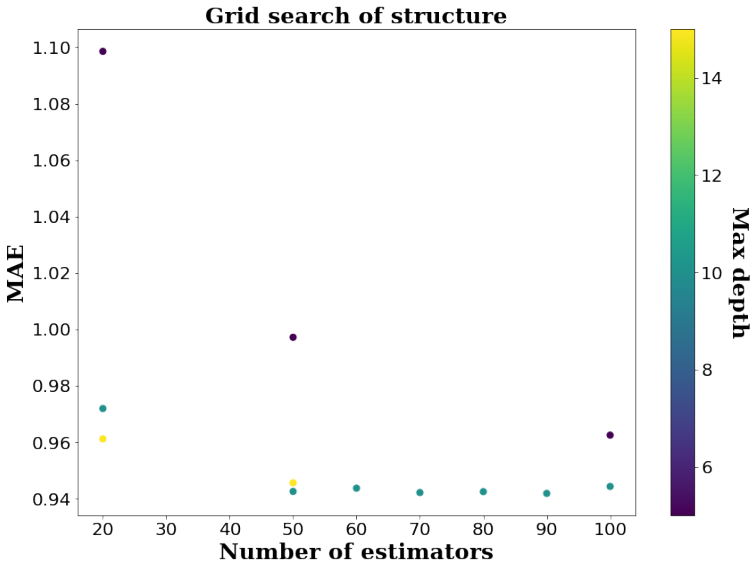
3.2.2 Boosted decision tree

As for the MLR, the XGBoost trained for the maximum temperature exploits the same structure of the one trained for the minimum temperature. The input variables without preprocessing can be passed and we have that the parameters are selected based on a grid search. The inputs are:

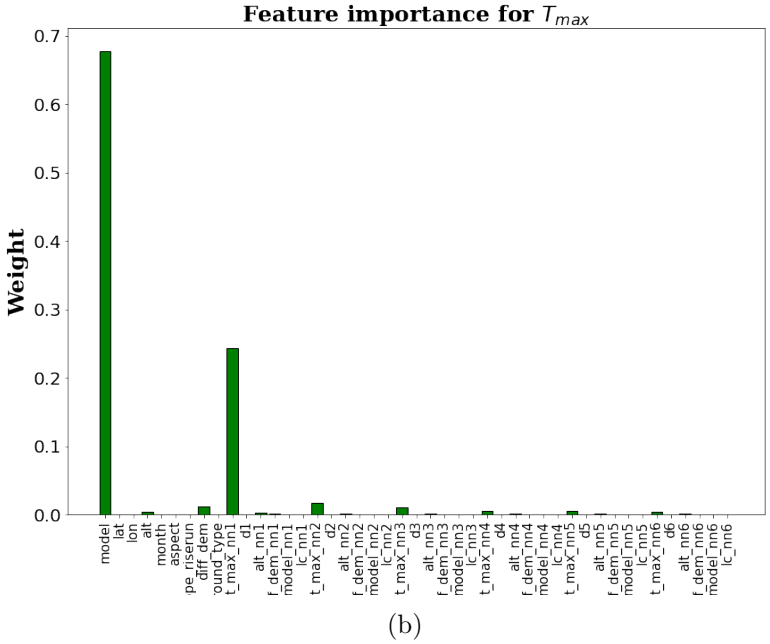
- longitude, latitude, altitude, slope, difference of DEMs, aspect and ground type.
- From the neighbors: maximum temperature, distance, altitude, maximum temperature from the model, difference of DEMs and land cover.
- Maximum temperature from model for the unknown point and the month.

Also in this case a combination of a random search with a grid search are exploited to reach the best structure for our algorithm. As we can see from figure 3.5a the grid search do not require to investigate over large range of parameter to find the optimal solution. A model with 50 trees with a depth of 10

is able to generalize the information without overfit the data. With this model we reach a MAE on the test set of $0.95C°$. This time we have that the MAE of this model is higher of the error reached by the MLR. This can mean that a complex algorithm such as an XGBoost is an overkill for this task (despite this we will see that the neural network remains the best solution).



(a)



(b)

Figure 3.5: Grid search results on structure (a) and feature importance for the evaluation of the maximum temperature (b).

An important consideration that we can make using the decision tree is the information showed in figure 3.5b. XGBoost in fact allows us to get the features importance meaning that we can see the "weights" that the algorithm gives to the inputs. As we can see information on neighbors and on model are largely the most used from the predictor.

3.2.3 Neural network

The neural network for the evaluation of the maximum temperature has only one relative difference on the inputs with respect to the neural network for the minimum temperature. In fact in this case we add the information on the aspect. For the model this variable is a bit confusing because if on one hand we have that the aspect is represented by a continuous value between 0 and 360, on the other hand we have that the extremes of this interval represent the same geographical condition. For this reason we transform this variable in a discrete variable. In practice we divide the aspect in four ranges and we associate the four cardinal points to each event. Once that this new variable is ready we can pass it to a one hot encoder to get the vectors. The same techniques for the normalization and standardization of the minimum temperature are performed. Just for completeness, we present the inputs that are formally the same as the XGBoost:

- longitude, latitude, altitude, slope, difference of DEMs, aspect and ground type.
- From the neighbors: maximum temperature, distance, altitude, maximum temperature from the model, difference of DEMs and land cover.
- Maximum temperature from model for the unknown point and the month.

For the setting of the hyper parameters we can exactly retrace what we did for the minimum temperature and we do not repeat and is redundant to present again the process. We just remember the results:

- The Adam optimizer is again the best choice.
- The initial learning rate is set to 0.0005.
- The activation function of the neurons in the layers is a Relu function but in the output layer we have a linear function to permit negative values of temperature.
- The loss function is the *log_cosh* but the final metric that we evaluate is the MAE.

With these parameters the grid search for the best structure ends with the best architecture with 5 hidden layers and a number of neurons in each layer of $(input\ len, 11, 11, 9, 7, 7, 1)$. The MAE of this model is $0.854C°$. Looking at figure 3.6, which shows a portion of the grid search where the best performance are reached, we can see that all the models in the figure have more or less the same MAE. In this case, in opposition with the behavior of the minimum temperature, we can select any structure and the result will be almost the same. This fact is related to the absence of strange phenomena such as inversions and a general task that is less complex. Nevertheless we still prefer to choose the best model since the computational cost of the training is almost the same and once that the algorithm is trained the computational cost for the evaluation is negligible.

3.3 Rain

We will now see how to handle the problem of the spatialization of precipitation. This field is completely different from the evaluation of the temperature and so we have to change strategy to retrieve important information and to be able to draw a homogeneous map containing this variable. As already mentioned we will treat the problem as a multi source merging one. The main sources of information on this field are the model, the radar and the stations. Before the model description, we want to expose the metric that we use for the evaluation of the algorithms. The mean absolute error is of great use in the field of temperatures but working on precipitations we find different metrics. The most common are based of the consistency matrix (a $2x2$ matrix that reports the relation of observed variables and forecasts in the overcoming of some predetermined thresholds). It is possible to build different types of scalar attributes that represent the score of a predictor. One scalar that alone contains information on the prediction takes the name of Peirce Skill Score (PSS) [25]. Given a consistency matrix as table 3.1 the PSS is defined as:

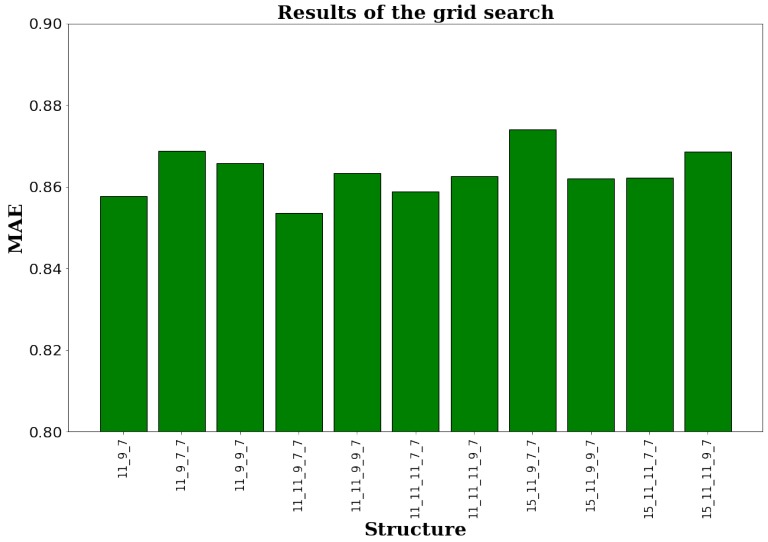


Figure 3.6: Grid search of the structure of the neural network for the maximum temperature.

$$PSS = \frac{ad - bc}{(a + c)(b + d)} \tag{3.3}$$

Since the PSS gives qualitative information on the predictor (in fact a value of 1 of this metric is equivalent to the perfect predictor while a 0 in equivalent to a random one), we still want to evaluate a quantitative error. For this reason we maintain the MAE for each split of the dataset. The split of the dataset is introduced due to the necessity of imposing some threshold to evaluate the PSS. Actually five threshold was selected: $1mm$, $5mm$, $10mm$, $20mm$, $100mm$. So for each predictor we have five PSS and we evaluate the MAE in each range defined by the thresholds. The final metric that we decide to implement to have a single value that takes in consideration both qualitative and quantitative aspect is the sum of the PSS minus the total MAE of the predictor in the dataset. The highest value will recognize the best model.

	Observed over	Observed less
Forecast over	a	b
Forecast less	c	d

Table 3.1: Consistency matrix

3.3.1 Cleaning of the dataset

We want to present now the strategy that we adopt to clear our dataset. Since we have now two sources of information to compare the rain that came from the stations, we implement a simple neural network to perform the cleaning of the dataset. The inputs of this model are:

- latitude, longitude, altitude and aspect of the point.
- Month of the event.
- Rain from the model and rain from radar.
- Maximum temperature of the considered point and information on the thunderbolts.

The idea is to not use the information from the station in this part of the work as input but we use this information as target variable. Like the other network we perform a grid search for the parameters and we choose the following values:

- 0.0005 is the initial learning rate with an Adam optimizer
- Relu function for the hidden layers and a Sigmoid function for the output layer to do not have a negative value of precipitation.
- A structure with six hidden layer with the following number of neurons $(input\ len, 15, 15, 9, 9, 5, 5, 1)$.

Once that the model is trained we can select the condition to eliminate corrupted data. Based on some "manual" observations we choose to select as bad data all the raingauges that report a value of precipitation greater than three times the prediction of the network or that is lower than the 15% if the prediction. Then if the difference between the raingauges and the network is greater than $10mm$ we confirm the hypothesis of bad data and we eliminate these events from the database.

Once that the dataset is ready we can present the implementation of the model for the evaluation of the precipitation over a grid. In this field we decide to not implement multi linear regressor and boosted decision tree but we still try three different methods based on neural networks. As we will see the grid research of the best parameters do not change too much between the various models but what intrinsically changes is the concept used for the training and the data we pass to the algorithms.

3.3.2 Double neural network

The first algorithm that we want to present is based on the application of two consecutive neural networks. The idea comes from a common tool used in the meteorological field to perform a spatialization over a grid of a given variable: the Kriging of the data. Without delving into the detail of the various Kriging techniques, we take it as a 2D regression. The first part of one of the algorithms exploited by Radarmeteo to spatialize information is based on the evaluation of the Gauge factor of the radar estimation and then a Kriging is used to generalize information where we do not have stations. The entire process can be summarized in:

- Evaluation of the precipitation of the radar in the given area.
- Comparison between the value reported by the radar and the stations in the station points.
- Evaluation of the Gauge factor of the radar: $Gauge = station/radar$.
- 2D regression with a Kriging approach to spatialize the Gauge factor.
- Spatialization of the precipitation given by $rain = Gauge \cdot radar$.

We decide so to redesign this algorithm based on machine learning tools. In Particular both the first evaluation of the precipitation and then the spatialization of the Gauge factor is performed exploiting a deep neural network. The first neural network is almost the same used for the cleaning of the dataset, the inputs in fact are:

- latitude, longitude, altitude, aspect.
- Rain from model, rain from radar.
- Month and information on thunderbolts.

The target of this first model is the rain measured by stations. The best model from the grid search has five hidden layers and a structure of $(input\ len, 9, 9, 5, 3, 3, 1)$ where all the activation functions are Relu except the last neuron that has a Sigmoid function. The other hyper parameters remain the same of the cleaning procedure. The score of the network is shown in table 3.2. The information on the MAE should be read as follow: the MAE of $2.127mm$ associated to $1mm$ meaning that we evaluate the MAE on a range of precipitation $1 - 5\ mm$. The information on the PSS instead means that the 0.729 is the value that we find evaluating the consistency matrix with a threshold of $1mm$.

The second network then has to incorporate information on stations and on neighbors. In particular since we are interested in the spatialization of the Gauge factor, we evaluate for each station on the training set that parameter and then we associate to each station also the information on the Gauge

	MAE $[mm]$	PSS
$1mm$	2.127	0.729
$5mm$	4.271	0.750
$10mm$	5.959	0.744
$20mm$	12.539	0.708
$100mm$	62.094	0.747

Table 3.2: Performance of the first neural network

factor of the six neighbors. Then we manipulate the value of the Gauge factor in order not to encounter problems of mathematical origin like a division by a null value. First of all we set a fixed range of variability of the Gauge factor between 0.1 and 10 imposing values out the range to be equal to the extremes. Then for the events with precipitation reported by the station and the radar lower than 5 mm we set the gauge factor to 1. The inputs of the second model are:

- latitude, longitude, altitude.
- Information thunderbolts, month.
- Gauge factor of the neighbors, the distance of the neighbors and their altitude.

The target variable of the network is the gauge factor of the station. In this case the best structure is a network with four hidden layers and a number of neurons given by $(input\ len, 9, 7, 7, 5)$ that has a MAE on the prediction of the Gauge factor of 0.165. To evaluate the whole performance of the algorithm associate to all the events in the test set a new prediction given by : $Prediction_2 = Prediction_1 \cdot Gauge\ factor$. The performances of the model are presented in table 3.3

	MAE $[mm]$	PSS
$1mm$	1.733	0.686
$5mm$	3.103	0.813
$10mm$	4.132	0.807
$20mm$	8.276	0.779
$100mm$	33.864	0.698

Table 3.3: Performance of the entire model.

We have a general lower error in each range with respect to the first network, the values of the PSS that is smaller for a threshold of $1mm$ or $100mm$ is probably related to the correction imposed to the Gauge factor.

3.3.3 Single neural network

The second approach is a simpler model with a single neural network that takes as inputs all the information on the rain and tries to predict the behavior of the raingauges. Namely the inputs are:

- latitude, longitude, altitude,aspect, slope.
- Month, information on thunderbolts.
- Rain from radar, rain from model, maximum temperature.
- Rain from the six neighbors, distances of these neighbors, model and radar for the neighbors.

The target of our algorithm is the rain reported by the station. The hyper parameter of the best structure is the same of the previous models with the only difference given by the structure. The best neural network is obtained with: an Adam optimizer with an initial learning rate of 0.0005, Relu function for the hidden layers and Sigmoid for the output, log_cosh as loss function and a structure

	MAE $[mm]$	PSS
$1mm$	1.414	0.851
$5mm$	2.703	0.861
$10mm$	3.975	0.863
$20mm$	8.328	0.851
$100mm$	32.966	0.771

Table 3.4: Performance of the single neural network with all the variables as input.

with six hidden layers with $(input\ len, 11, 15, 11, 9, 7, 7, 1)$ neurons in each layer. The Performances of this model are presented in table 3.4

As we can see we have a better result in general with respect to the first model (as we will see we have in fact that this is the best algorithm to use). We want to present and comment also the behavior of the loss function for the train and test set during the training procedure. Figure 3.7 shows this behavior. We can conclude that the model is not overfitting since we have a decreasing value for the loss for both the train and the test set but this time (if we compare that result with the one obtained for the minimum temperature in figure 3.4a) we have a more instability for what concerns the test set. For this reason we try to implement some dropout layers and some regularization on layers but the result was just a general loss of performances without a smooth loss curve. For this reason we can conclude that our model probably is not overfitting and the fluctuations in the test loss are related to the complexity and the variability of the variable itself.

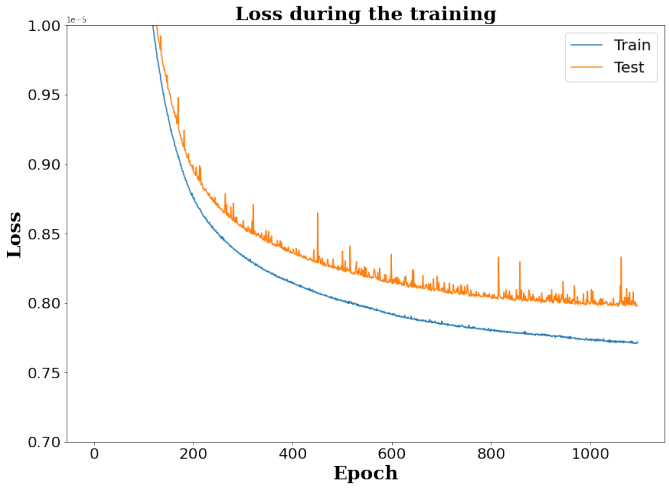


Figure 3.7: Loss function during the training.

3.3.4 Data augmentation

The last model that we want to present is not much different from the neural network above. In fact the model remains more or less the same (the grid search for the parameters was performed in any case) but we introduce a fundamental difference in the data. Remembering the graph shown in figure 2.21b we have a great unbalanced distribution. Since the more interesting phenomena are the more intense ones, we can accept a loss in terms of accuracy for low value of precipitation if we are able to gain more accuracy for intense events. For this reasons, after the train test split, we decide to manipulate the train set in order to obtain more information from events with high values of precipitation. Guided by an idea that is often applied in the field of computer vision, we try to perform a data augmentation on our dataset. Data augmentation is the concept of generating new, fake data starting from the original one (in computer vision a common technique of data augmentation is to rotate, stretch or blur the image). To perform this type of technique on a meteorological dataset

we define our custom function for generating fake data. First of all we set the final dimension of our dataset (to 5 000 000 in our case). Then we select some range of rain to fill with values, in our application we select $([0-5],[5-10],[10-20],[20-50],[50-100],[100-300],[300-700])$. For each range we select a percentage of the total dataset that we want to have in that bin. Then if we have enough data in the original dataset to cover the entire request we sample the required number of event, otherwise the generation of fake data is used. To generate that data we start from original events and for each variable we add a random number based on the variable we are augmenting. For what regards the rain we add a random number between $-i$ and i where i depends from the range (respectively we have $i = (1, 2, 4, 5, 15, 25, 50)$) with the constraint of having positive values. For the latitude, longitude and the distances of the neighbors we add a random number in the range $[-0.02, 0.02]$; the altitude in the range $[-15, 15]$. The T_{max} is multiplied by a random number between 0.9 and 1.1 and the information on thunderbolts are obtained multiplying the original value for a number in the range $[0.2, 2]$. With this setup we have a final distribution given by figure 3.8. As we can see for precipitation lower that 300 mm we have the original distribution in each bin of the division. The strange behavior of the precipitations above $300mm$ is related to the small number on events that we have with these intensities.

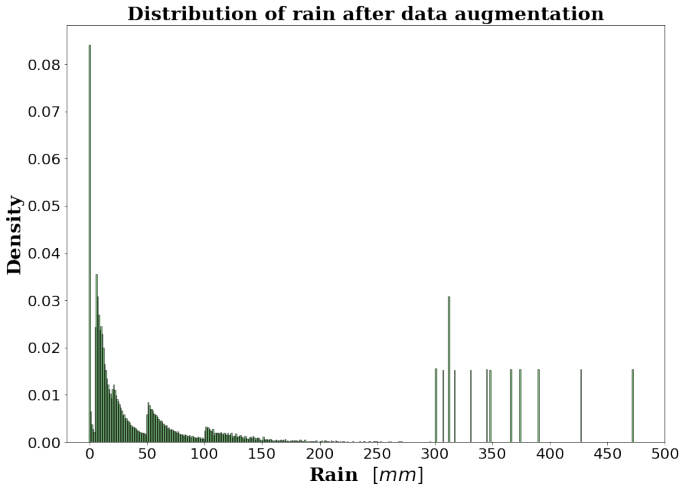


Figure 3.8: Distribution of the rain after the data augmentation.

Once that the dataset is ready we can perform a grid search with the same setup of the previous model. The only hyper parameter that changes with respect to the model with all in input is the structure that this time is a five hidden layers network with $(input\ len, 15, 9, 9, 5, 5, 1)$ as number of neurons in the respective layers. The performances of this model are summarized in table 3.5

	MAE $[mm]$	PSS
$1mm$	3.464	0.650
$5mm$	3.168	0.587
$10mm$	3.948	0.710
$20mm$	8.793	0.747
$100mm$	26.875	0.580

Table 3.5: Performance of the single neural network with all the variables as input and the data augmented.

As we can see the only advantage in this model is the MAE obtained for events with a precipitation greater than 100 mm. As we will comment in the next chapter this advantage does not give us sufficient condition to accept the model as better than the others so we still prefer the second algorithm with the original data. This behavior can be commented on because in some way we are corrupting the original distribution of the data and the network is not able to recognize patterns and makes good

predictions. In the next chapter we will return on the results presented here and we exploit these results to reach our final goal.

Chapter 4

Results and conclusion

With the data augmentation for the precipitation and the relative training, we conclude the part that regards the exploration and the training of the various algorithms to achieve some useful knowledge that allows us to perform a spatialization on a grid of variables of our interest. The chapter is divided into three sections:

- in the first section we want to present some comparisons between the prediction of our algorithm with respect to the data used as a black box, namely the model for the temperatures and model and radar for the precipitation.
- The second section regards instead the presentation of our final products: the grids.
- In the last section we want to conclude the work with an analysis of the grids, some comments on the entire pipeline and we propose also some future developments based on machine learning tools.

4.1 T_{min}

To follow the workflow of the document we want to start with the minimum temperature. As explained in section 3.1.4 we have that a predictor with a structure of $(input\ len, 15, 9, 9, 7, 7, 1)$ is the best model for the task and it is able to reach a MAE of $1.017C^\circ$ on a test set. If we compare the predictions of this model with the values of minimum temperature reported by the station in a given day (the images showed below report the value of the $2020 - February - 09$) we get a result similar to the one presented in figure 4.1. Two main comments can be done looking at this figure:

- we can not recognize a clear pattern in the residuals. One way to describe that behavior is to associate a randomly distributed error to the predictor that means that all the useful information is used by the network to make the prediction remains an unpredictable random noise.
- We have that all the scattered points have a low intrinsic value in accord with the MAE of $\approx 1\ C^\circ$. In the light of this we have another signal of goodness of the network.

Another comparison that we want to present and comment on is the scatter plot of a given day for what concerns the difference between the model and the predictor. Figure 4.2a shows a typical behavior of the residuals during winter(we actually have that also during summer the situation is more or less the same with the difference that the residuals tend to be more generally negative). The focus here is that in fact we can recognize some patterns, for example looking at the norther part of Italy (the Alpi) we find only negative residuals. This may mean that with respect to the station, here we have some intrinsic information that the model is not able to exploit and we do no just have random fluctuations. To support the hypothesis that the neural network performs better than the model we can look at figure 4.2b. This figure represents the spatial distribution over Italy on a given day of the residuals

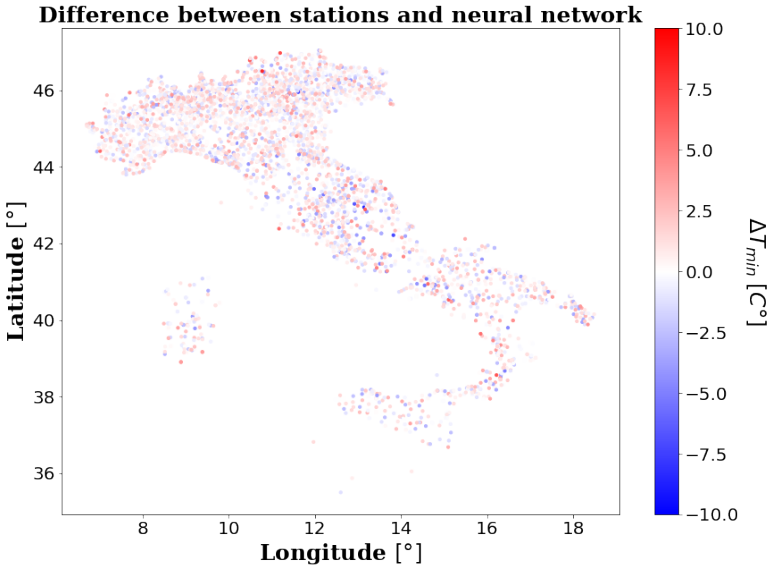
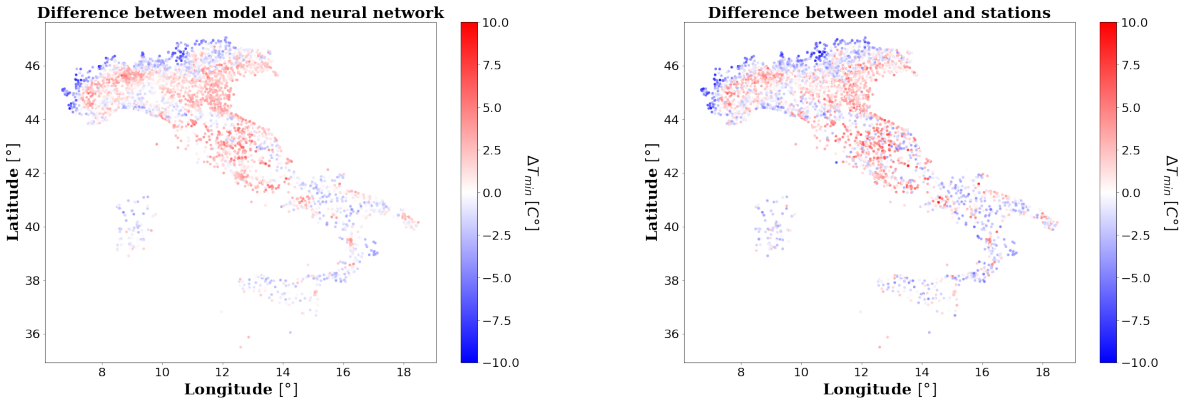


Figure 4.1: Residual of the minimum temperature reported by the station and predicted by the neural network in date $2020 - February - 09$.

between model and station. We can clearly recognize the same pattern of figure 4.2a so in fact we have that the neural network and the station have the same behavior.
Another information that we can find looking at the comparison model-neural network is that again in the Alpi we can see that inside the blue dots we find some red one dot. Red dots mean that the neural network predicts a value of minimum temperature for this point lower than the model. These areas correspond to depression of the mountain where (during winter for the most) we find inversion that the model is not able to recognize.

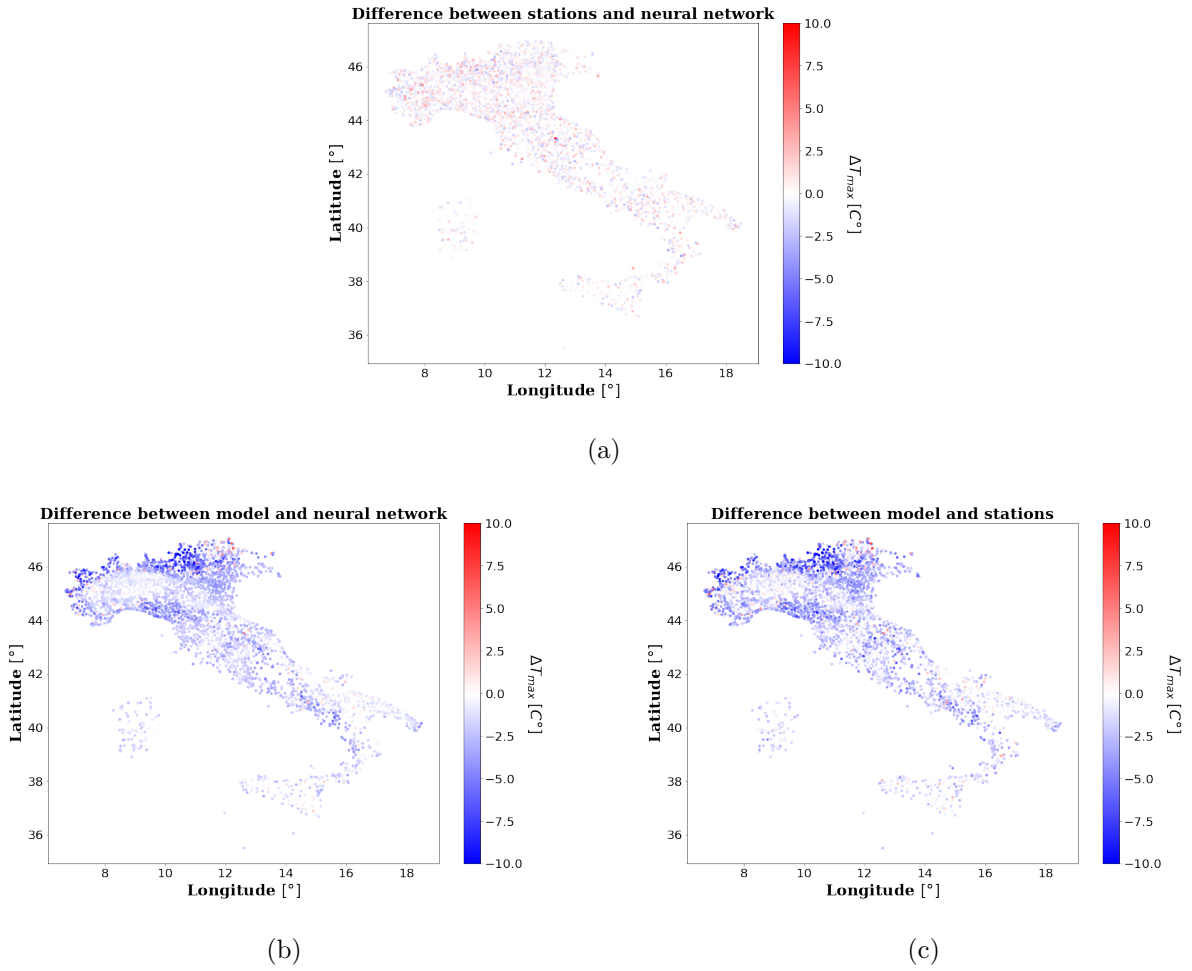


(a) (b)

Figure 4.2: Spatial representation of the residual between the meteorological model and the neural network (a) and between the model and the stations (b).

4.2 T_{max}

For the maximum temperature the grid search presented in the previous chapter gives us the best model with a structure $(input\ len, 11, 11, 9, 7, 7, 1)$ and the MAE of this model is $0.854C°$. Also for this variable we have the possibility to make some comparison between the predictor, the values reported by the station and the information by the meteorological model.



(a)

(b)

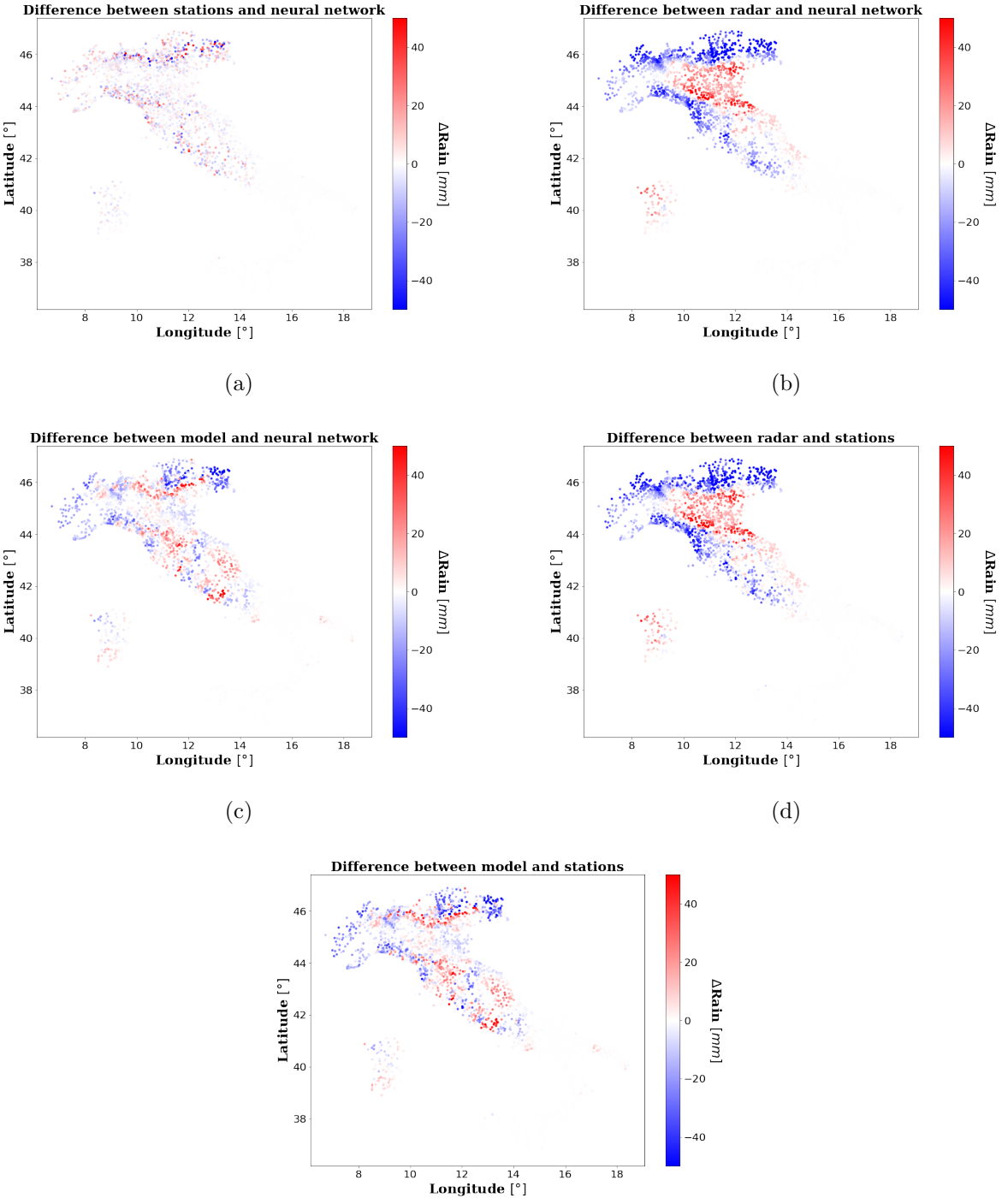
(c)

Figure 4.3: Spatial representation of the residual between the neural network and the station (a), the meteorological model and the neural network (b) and between the model and the stations (c).

Starting from the residuals between stations and neural network we can see from figure 4.3a that we find more or less the same general behavior of the residuals between T_{min} and station. Also in this case we can see a lot of confusion and the residual seems to be randomly distributed around the zero without any particular region with a predominant behavior. Same conclusion of the minimum temperature can be done and we can treat this error as random error. For what concerns figures 4.3b and 4.3c, we can see that the model seems to underestimate the maximum temperature in general. The plots regard a day during summer but also in winter the situation is the same. We can make a relation between this behaviour (especially in mountains) and the DEM upon which the model itself is built. We have in fact that this DEM has a resolution of 4 km and so the peaks are understated and since we have a strong relation between altitude and T_{max} the result is an underestimation also of the temperatures.

4.3 Rain

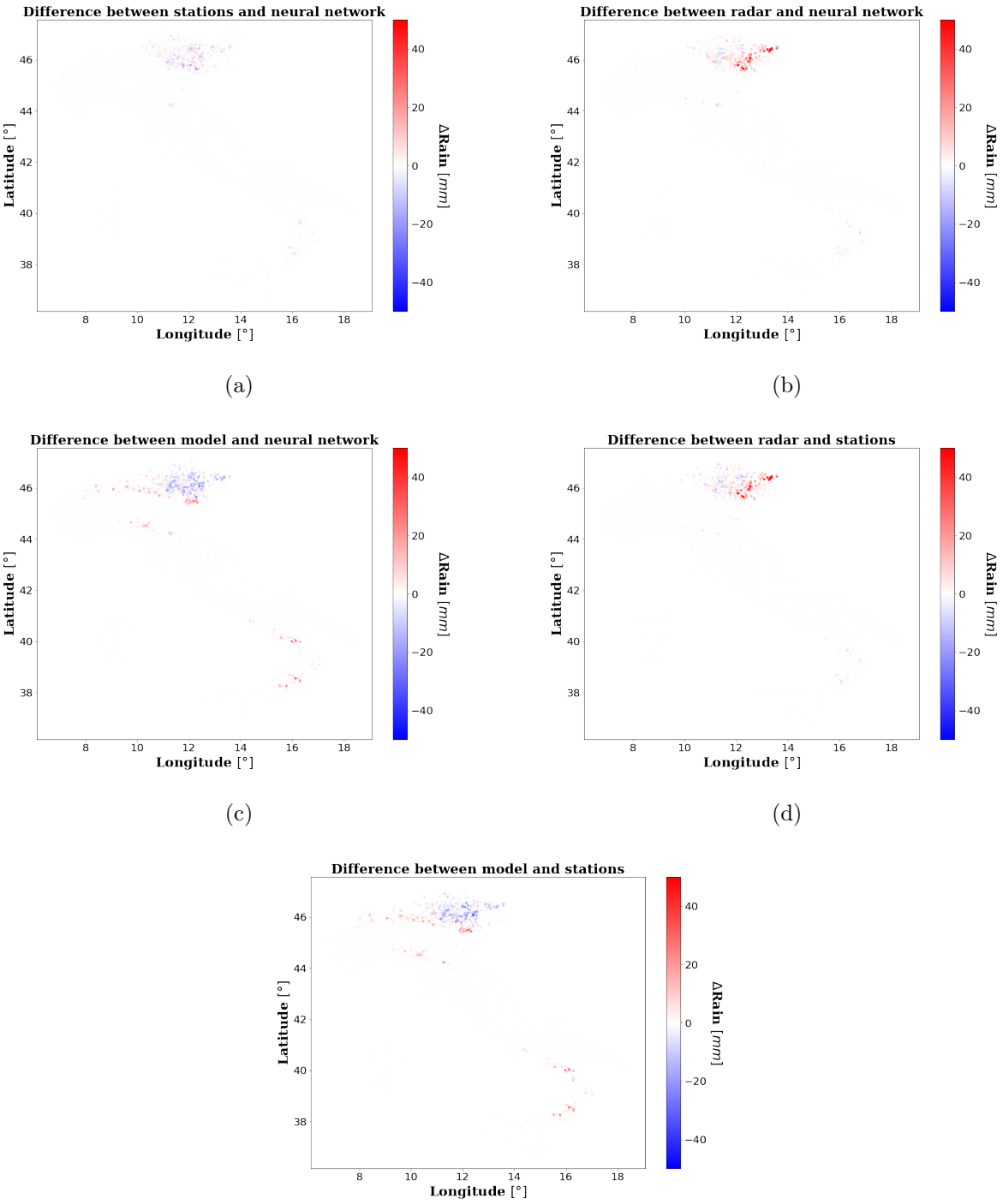
For what regards the precipitation we want to present two different comparisons. Since from the analysis emerges that the behaviors of the radar and of the meteorological model change based on the type of precipitation, we want to show two typical situations, the first one with an event of heavy rain during autumn and one with a convective event in August. We start with autumn in figure 4.4. From 4.4a we can make the same consideration of the temperatures. Since the residuals between our predictor and the station appear as randomly distributed around zero we probably extract all the useful information without a systematic error.



(a) (b) (c) (d) (e)

Figure 4.4: Comparison of the various models during a typical event in autumn

From the other plots instead we can assert that the comparison model/radar with respectively station and neural network gives us almost the same results but in some way we recognize that the radar tends to overestimate precipitations in plains and underestimate precipitations in mountains while in autumn the residuals of the model appear more randomly distributed and more close to one (the colors on the plots that regards radar are brighter). On the other hand, as already mentioned, during summer is more probable to have isolated convective phenomena and as we can see from figure 4.5c and 4.5e the model tends to spread the precipitation in a larger area than that recognized by the network and radar. This time so the network is able to recognize the right location and more weight will be given to the radar.



(a) (b) (c) (d) (e)

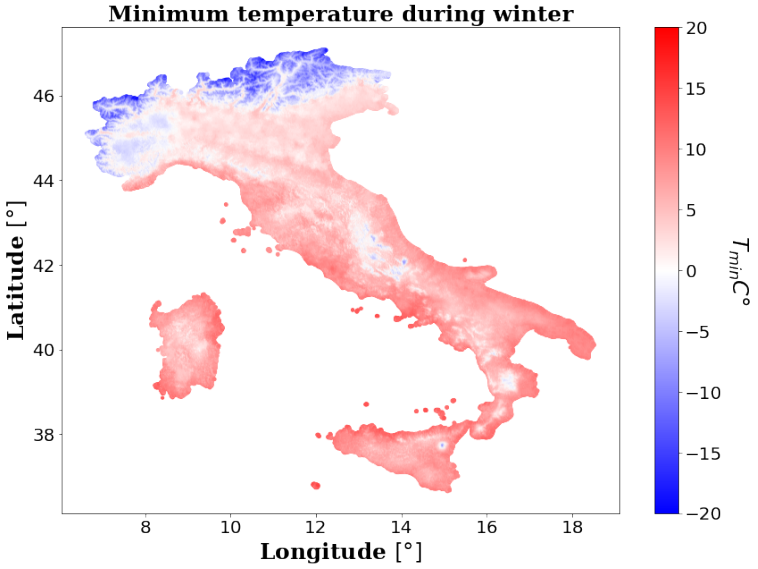
Figure 4.5: Comparison of the various models during a typical event in summer.

4.4 Conclusion

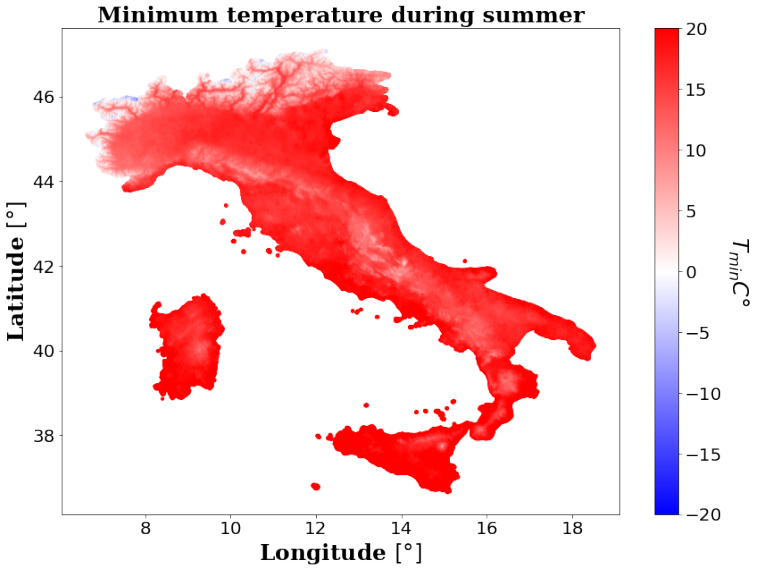
After the analysis and the presentation of the results we can present the final product of the entire work: the grids. The grids are made using 331267 points uniformly distributed over Italy, each point is the center of a 1 km x 1 km square

Grids

As always let us begin with the minimum temperature. We can see from figure 4.6 examples of a spatialization of minimum temperature during a typical day in winter and summer respectively. The



(a)



(b)

Figure 4.6: Grid over Italy for the minimum temperature in a typical day during winter (a) and summer (b)

first comment regards the phenomenon of inversion. First of all we can see that during summer we see more homogeneous colors meaning that the inversion is not present or at least it is really rare. During winter instead we have more complex behavior. First of all we can clearly distinguish the valleys creeping into the mountains (in the norter area) with higher temperatures at the mouth of these lowland valleys, and then, for example looking at the Pianura Padana, we have a more speckled red. Looking in detail we have that in some case the more intense red (higher temperature) is associated with a hilly area (for example Colli Euganei in Veneto) where in winter the thermal inversion is very

common, in other cases we have the information of the ground type that which reveals large urban centers. Another important concept that emerges here is the higher temperature of the coast. In literature, it is common to find works with the purpose of us that use distance from the coast as an input variable for the algorithms. We decide to not use this information since Italy is a very varied territory exposed to water and we enclose this information in the land cover. Nevertheless we are able to recognize higher temperatures near the sea.

The maximum temperature in the field of the generation of grids covers a fundamental role. First of all it is important for having a uniform distribution of this variable over a territory but it is fundamental because we pass as input to the neural network that predicts the precipitation information on the maximum temperature of the considered point. In the training procedure we retrieve this information directly from the station but here we have to associate a T_{max} to each point in the grid and then we can evaluate the precipitation. For this reason we present in figure 4.7 two couples of plots that are example of T_{max} and Rain evaluated in the same days during autumn and during spring respectively. Again we find a typical muontuous heavy precipitation during autumn (badly seen from the radar but good from the model and the neural network) and a "local" precipitation during spring seen well by the radar and the neural network and worse from the model.

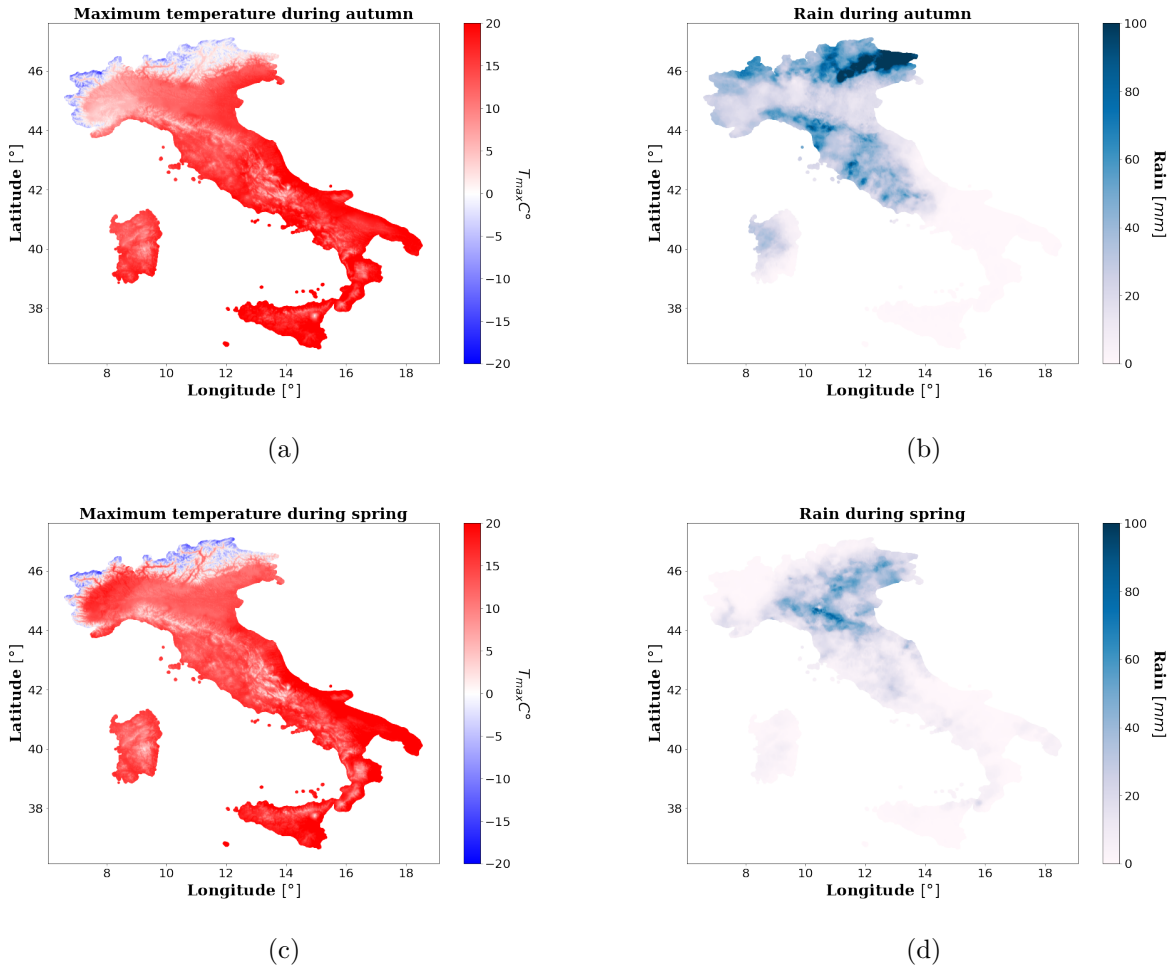


Figure 4.7: Grid over Italy for the maximum temperature and rain in a typical day during autumn (a,b) and spring (c,d)

Comments and Future developments

With these grids, we conclude the entire work regarding machine learning methods to perform a spatialization of a variable uniformly over a territory. We have that the three main variables have substantial differences. For what regards the minimum temperature it emerged that the most critical part is the search for relationships with thermal inversions. In this field we have that a complex structure such as a neural network, combined with a large preliminary study on the variables, gives us good results in terms of both the ability to find thermal inversions and also in a general case with a low value of the MAE. In this field we have also the possibility to make some relations with other models. In particular the comparison with a linear model (the MLR) give us information about the advantage to exploit non-linear relation between the inputs. Using a neural network in fact we not only have a MAE better than a 20% with respect to the MLR but producing some examples with this latter model we also can't recognize any inversion. For the maximum temperature we still have a better performance using a neural network instead of a linear model but this time we find a MAE better of 5%. This is related to the fact that as already mentioned the evaluation of T_{max} is a less complex task with respect of T_{min}. For the rain instead we do not test any linear approach since the multi source merging of different models and variables drive us to directly select a neural network approach. As emerged from the analysis we are able to exploit the information from the model and from the radar in their areas and time periods of best functioning and exploiting also the information on neighbors. The criticality of the rain determination process, which is one open problems of our work and that can be proposed as future development, is that can happen that the radar for example, in plains during summer (best situation to use the radar) see large amounts of precipitation in a restricted area where we do not have nearby stations. If in addition the model fails in the prediction since probably we have a very restricted convective phenomena, may happen that the network will give more weight to the radar but will also deliver less intensity of precipitation due to the values reported by the six closer stations and model.
Despite all neural networks offer a method for the estimation of these variables without some drawbacks of more classical methods such as the linearity assumptions of regression largely used in tasks like this. For sure we have also other predictors that can fit into the task; as we saw, XGBoost is a great alternative for the evaluation of the temperatures. We still prefer neural networks since we can train them more easily and the evaluation is faster. For this reason the pipeline developed with neural network is of great interest also from a computational time point of view and Radarmeteo is actually implementing these models for various uses. First of all the accuracy of the temperatures and the ability to predict inversions is really high and then we can also use this model to act another validation procedure as we saw that outlayers of the distribution of the residuals between stations and NN are actually bad data that pass the validation check.

Of course the field of the interpolation over a grid is not limited to our work and we only explored the tip of the iceberg. Some other implementations can be done both exploiting the existing system presented here or implementing new concepts based on a machine learning approach. The first future proposal regards the data. Since we have a relatively short period from when the validation is implemented, in the future we can treat more "good" data and intrinsically we can achieve a better knowledge of the entire field of application. Other variables can be explored in order to get even more information on thermal inversion, rain and temperatures in general such as cloud cover, information on fog, solar radiation and so on. Other related works are exploiting a different infrastructure from a fully connected network. For example it is possible to map information in a 2D scale and pass these 2D variables to a convolutional network. In this way we are probably able to get more information from the surrounding area and the combination of the variables that are transformed into "channels" of an image can give us more precise predictions. Even more complex models can be based on information of a time windows. For example to perform the spatialization of minimum temperatures, it is possible to exploit models such as Long Short Term Memory to make some relations also with information from previous days. Taking this concept further to an analysis of the present rather than the past, it is possible to find applications for the future forecast of meteorological variables.

These are just some of the possible applications of machine learning in the field of meteorology and the trend shows us that machine learning techniques and neural networks will be increasingly present and increasingly able to offer satisfactory results.

Bibliography

[1] S. Scher and G. Messori, "Weather and climate forecasting with neural networks: using general circulation models (gcms) with different complexity as a study ground," *Geoscientific Model Development*, vol. 12, no. 7, pp. 2797–2809, 2019. [Online]. Available: https://gmd.copernicus.org/articles/12/2797/2019/

[2] J. P. Rigol, C. H. Jarvis, and N. Stuart, "Artificial neural networks as a tool for spatial interpolation," *International Journal of Geographical Information Science*, vol. 15, no. 4, pp. 323–343, 2001. [Online]. Available: https://doi.org/10.1080/13658810110038951

[3] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, "A high-bias, low-variance introduction to machine learning for physicists," *Physics Reports*, vol. 810, p. 1–124, May 2019. [Online]. Available: http://dx.doi.org/10.1016/j.physrep.2019.03.001

[4] S. Shalev-Shwartz and S. Cen-David, *Understanding machine learning, from theory to algorithms.* 32 Avenue of the Americas, New York, NY 10013-2473, USA: Cambridge University press, 2014.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[6] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[7] J. H. Friedman, "Stochastic gradient boosting," *Computational Statistics and Data Analysis*, vol. 38, no. 4, pp. 367–378, 2002, nonlinear Methods and Data Mining. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167947301000652

[8] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, p. 5–32, Oct. 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[9] J. Friedman and B. Popescu, "Importance sampled learning ensembles," 10 2003.

[10] G. S. M. Thakur, R. Bhattacharyya, and S. S. Mondal, "Artificial neural network based model for forecasting of inflation in india," *Fuzzy Information and Engineering*, vol. 8, no. 1, pp. 87–100, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1616865816300061

[11] F. F. Ting, Y. J. Tan, and K. S. Sim, "Convolutional neural network improvement for breast cancer classification," *Expert Systems with Applications*, vol. 120, pp. 103–115, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417418307280

[12] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 303–314. [Online]. Available: https://doi.org/10.1145/3180155.3180220

[13] D. Hebb, "Organization of behavior. new york: Wiley," *J. Clin. Psychol*, vol. 6, no. 3, pp. 335–307, 1949.

[14] P. Werbos, "Beyond regression:" new tools for prediction and analysis in the behavioral sciences," *Ph. D. dissertation, Harvard University*, 1974.

[15] P. Dayan and L. F. Abbott, *Theoretical neuroscience: computational and mathematical modeling of neural systems.* Computational Neuroscience Series, 2001.

[16] M. A. Nielsen, *Neural networks and deep learning.* Determination press San Francisco, CA, 2015, vol. 25.

[17] "Copernicus land monotoring service," https://land.copernicus.eu/.

[18] R. Barnes, *RichDEM: Terrain Analysis Software*, 2016. [Online]. Available: http://github.com/r-barnes/richdem

[19] T. T. Warner, *Numerical weather and climate prediction.* cambridge university press, 2010.

[20] S. Michaelides, F. Tymvios, and T. Michaelidou, "Spatial and temporal characteristics of the annual rainfall frequency distribution in cyprus," *Atmospheric Research*, vol. 94, no. 4, pp. 606–615, 2009.

[21] Q. Hu, Z. Li, L. Wang, Y. Huang, Y. Wang, and L. Li, "Rainfall spatial estimations: a review from spatial interpolation to multi-source data merging," *Water*, vol. 11, no. 3, p. 579, 2019.

[22] Q. Sun, C. Miao, Q. Duan, H. Ashouri, S. Sorooshian, and K.-L. Hsu, "A review of global precipitation data sets: Data sources, estimation, and intercomparisons," *Reviews of Geophysics*, vol. 56, no. 1, pp. 79–107, 2018.

[23] E. Goudenhoofdt and L. Delobbe, "Evaluation of radar-gauge merging methods for quantitative precipitation estimates," *Hydrology and Earth System Sciences*, vol. 13, no. 2, pp. 195–203, 2009.

[24] K. A. Browning, "Uses of radar in meteorology," *Contemporary Physics*, vol. 27, no. 6, pp. 499–517, 1986. [Online]. Available: https://doi.org/10.1080/00107518608211028

[25] D. S. Wilks, *Statistical methods in the atmospheric sciences*. Academic press, 2011, vol. 100.