Project 3 on Machine Learning
Logistic Regression, Deep Neural Network and Support Vector Machine by comparison

Adele Zaini*

(Dated: December 17, 2021)

ABSTRACT

A comparison between the three Machine Learning techniques of Logistic Regression, Neural Network and Support Vector Machine is presented. Their performance is evaluated in a classification case, using the Climate Models Simulation Crashes dataset from the UCI repository, that explores the failures in running climate models ensembles when perturbating different input parameters. The combination of Machine Learning techniques in climate sciences is rising in importance in the recent decades, due to the stochastic component of the atmosphere that is demanding in modelling using the standard techniques. In this work, a side case of using machine learning techniques to find an apparently non-deterministic pattern in evaluating climate models limitations is here presented. Evaluating the performance of the three techniques, the Neural Network is the one that predicts better when the climate model could fail with an accuracy score of 96.1% and 10 failures correctly predicted out of the 14 expected. The accuracy scores of the Logistic Regression and the Support Vector Machine are respectively of 92.8%, and 92.2%, which seem high, but looking at the failures hit only, they fail in this prediction. The Neural Network performance is also improved when optimizing the hidden activation functions and the optimizers, reaching almost the 98% of accuracy score with the combination of *elu* and *sgd* respectively.

I. INTRODUCTION

Machine Learning techniques are expanding in all the fields of research in the last decades and have led to important improvements. One of this field belongs to the Climate Sciences, where studying the stochastic behaviour of the atmosphere is not possible with only the classical methods. These techniques are also useful to learn about further insights on side topics, such as uncertainty quantification (UQ) simulations, and to improve the Earth System Models themselves.

In this report we will compare three different Machine Learning techniques applied to a classification dataset realized by climate simulations (Section II). After implementing Logistic Regression, Neural Network and Support Vector Machine, their performance will be quantified and compared. The second part of the analysis focuses instead on optimizing the Neural Network, finding the best combination between the activation function of the hidden layers and the optimizer.

The report is organized as follows. After this introduction, an insight of the dataset is presented (*part a*[1]), followed by the theory around the machine learning techniques (*part b*) and the algorithms implemented (*part c*). The following section will present the results of the work (*part d*), while commenting and discussing them. The final section is to summarize the analysis into the conclusions and to have an overview on future perspectives (*part e*).

II. DATA

Since my interest in Climate Modelling, the dataset used in this work is taken from the University of California at Irvine (UCI) repository and deals with the analysis of crashes in Climate Models Ensembles when perturbating different input parameters. The dataset is called *Climate Model Simulation Crashes* and on the UCI website all the information about this dataset and its structure can be found (Ref. [1]).

Motivation This dataset was created to respond to an emergent finding when analysing the effects of ocean model parameter uncertainties on climate simulations: a series of simulation crashes within the Parallel Ocean Program (POP2) component of the Community Climate System Model version 4 (CCSM4). About 8.5% of the CCSM4 simulations failed for numerical reasons at combinations of POP2 parameter values. D.D. Lucas et al. [2] created the present dataset composed by the parameters values and the outcomes (success, failure) for each run, in order to find a pattern in the multidimensional space of parameters using the Support Vector Machine technique and to better understand the reasons underlying the problem.

* GitHub profile: https://github.com/adelezaini/

[1] The project text divides the work in five parts, in this work these parts are sections of the report, following the given order.

Climate Models and UQ ensembles Modern global three-dimensional climate models are extraordinarily complex pieces of science. They contain over a million lines of code and use hundreds to thousands of functions and subroutines to solve equations of state and conservation laws for the flows of matter, energy, and momentum within and between the atmosphere, oceans, land, and other reservoirs of the Earth system. They also use numerous algorithms of biological, chemical, geologic, and anthropogenic processes to simulate the cycles of carbon, nitrogen, sulfur, aerosols, ozone, greenhouse gases and other climate components. Given this enormous complexity, climate models are vulnerable to many types of computational issues. As code verification can be used to find software bugs, emerging tools being developed in the field of uncertainty quantification (UQ) can help pinpoint scientific discrepancies in simulation models, the knowledge of which can be used to guide and improve model development. Primary UQ targets for climate models are schemes containing parameters with adjustable values. This type of simulations are called *Perturbed Parameter Ensemble* (PPE). Small perturbations to the values of the parameters can amplify and lead to large changes in simulation outputs. In some cases, the simulations may fail altogether. The present dataset reports a series of simulation crashes encountered while running perturbed parameter UQ ensembles of the Community Climate System Model Version 4 (CCSM4) (Ref.[2]).

Dataset structure In spite of the complexity underlying these climate model simulations, the authors of the dataset treat the crash problem as a black box where just the values of the input parameters and a binary outcome flag indicating whether the simulations failed or were completed are known. In order to generate this dataset, three separate Latin hypercube ensembles were conducted, each containing 180 ensemble members, given 18 different parameters as input. 46 out of the total 540 simulations failed for numerical reasons at combinations of parameter values. The description of the columns of the dataset is the following:

- *Col 1 - "Study"*: Latin hypercube study ID (study 1 to 3).
- *Col 2 - "Run"*: simulation ID (run 1 to 180)
- *Cols 3-20 - variable names*: values of 18 climate model parameters scaled in the interval [0, 1]
- *Col 21 - "outcome"*: simulation outcome (0 = failure, 1 = success)

For a basic explanation of the Latin hypercube you can check [3] and for further mathematical insights look at Z. Liu et at., 2015 [4].

The *variable names* are listed in the Table 1, for further details look at the complete table in the Appendix VI.

	Parameter	[low, default, high]	Description
1	vconst_corr	$[0.3, 0.6, 1.2] \times 10^{7}$	variable viscosity parameter (vconst_1, vconst_6)
2	vconst_2	[0.25, 0.5, 2.0]	variable viscosity parameter
3	vconst_3	[0.16, 0.16, 0.2]	variable viscosity parameter
4	vconst_4	$[0.5, 2.0, 10.0] \times 10^{-8}$	variable viscosity parameter
5	vconst_5	[2, 3, 5]	variable viscosity parameter
6	vconst_7	[30.0, 45.0, 60.0]	variable viscosity parameter
7	ah_corr	$[2.0, 3.0, 4.0] \times 10^{7}$	diffusion coefficient for Redi mixing (ah) and background horizontal diffusivity within the surface boundary layer
8	ah_bolus	$[2.0, 3.0, 4.0] \times 10^{7}$	diffusion coefficient for bolus mixing
9	slm_corr	[0.05, 0.3, 0.3]	maximum slope for bolus (slm_b) and Redi terms (slm_r)
10	efficiency_factor	[0.05, 0.07, 0.1]	efficiency factor for submesoscale eddies
11	tidal_mix_max	[25.0, 100.0, 200.0]	tidal mixing threshold
12	vertical_decay_scale	$[2.5, 5.0, 20.0] \times 10^{4}$	vertical decay scale for tide induced turbulence
13	convect_corr	$[1.0, 10.0, 50.0] \times 10^{3}$	tracer (convect_diff) and momentum (convect_visc) mixing coefficients in diffusion option
14	bckgrnd_vdc1	[0.032, 0.16, 0.8]	base background vertical diffusivity
15	bckgrnd_vdc_ban	[0.5, 1.0, 1.0]	Banda Sea diffusivity
16	bckgrnd_vdc_eq	[0.01, 0.01, 0.5]	equatorial diffusivity
17	bckgrnd_vdc_psim	[0.1, 0.13, 0.5]	maximum PSI induced diffusivity
18	Prandtl	[4.0, 10.0, 20.0]	ratio of background vertical viscosity and diffusivity

FIG. 1. Description of the 18 features. Note: individual corr parameters (numbers 1, 7, 9, and 13) are used to represent the correlated pair of parameters given in the description. For example, values drawn for vconst corr are assigned to vconst 1 and vconst 6. b Linear and logarithmic scales are used for parameter ranges that have ratios of high/low ¡ 5 and high/low 5, respectively. For the complete table look in the Appendix (Ref. [2])

Note that the dataset parameters are already scaled in the interval [0,1], so they will not be rescaled in the process. In the splitting for training and test data, the studies 1 and 2 are taken into the training dataset, while study 3 is considered to test the model performance.

Dataset analysis

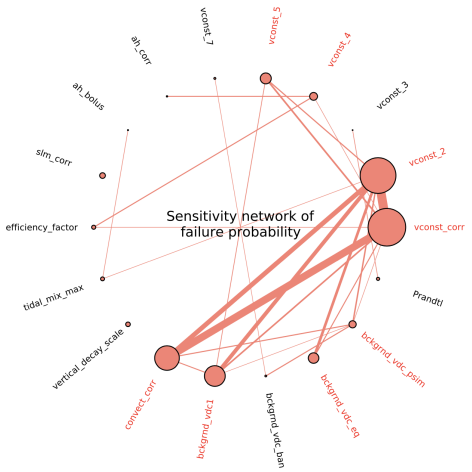

FIG. 2. Sensitivity analysis performed on the 18 perturbed parameters, showing which are the most important relationships with the outcome (Ref. [2]).

Figure 5 shows a very little correlation among the data and . Note that very narrow colorscale is set in order to appreciate the little differences.

Analysing the frequency and the distribution of the success-failure per parameter, most of the parameters do not show any particular behaviour in relation to the simulation crashes. In Figures 3 and 4 just the four most

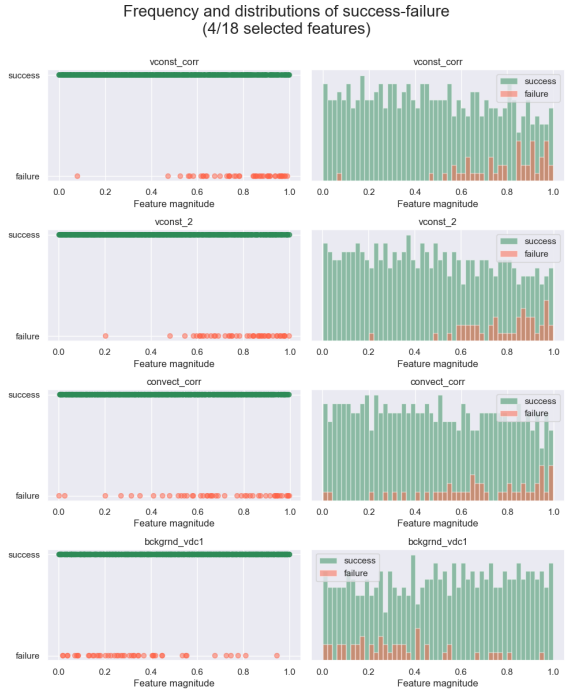

FIG. 3. Frequency and distribution of the outcomes for the 4 most sensible parameters (Figure 2).

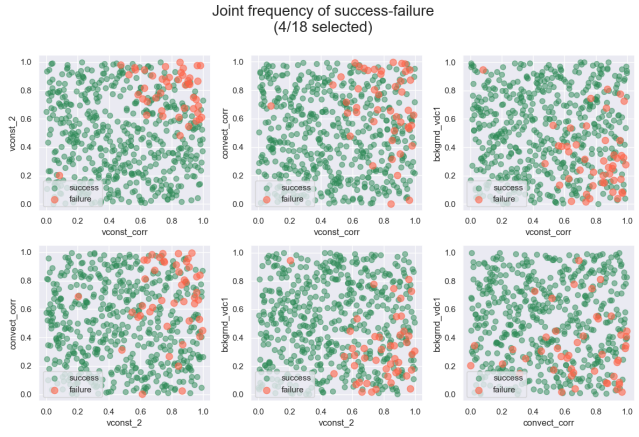

FIG. 4. 2D frequency of the outcomes for combinations of the 4 most sensible parameters (Figure 2).

sensible parameters are selected, based on the study [2] shown in Figure 2. Figure 3 represents the frequency and distribution of success-failure for each of *'vconst_corr', 'vconst_2', 'convect_corr', 'bckgrnd_vdc1'* (refer to Table 1 for variable complete names), while Figure 4 shows the joint frequency taken couples out of these four parameters to better show the relationship to the success of the simulations. We can then notice that the probability that model crashes increase when increasing values of *'vconst_corr', 'vconst_2', 'convect_corr'* and decreasing values of *'bckgrnd_vdc1'*. For further physical interpretation of these behaviours, look at Lucas D. D. et al. [2]. Here it is not reported since it is a discussion that goes further than the objectives of this work. The complete table with all the paramaters analysis can be found in the Appendix (Figure 20). It is not shown here to keep the discussion brief and for pointing out the important relationships.

Since there are 18 features in this dataset, our objective can be finding correlations among the data in order to condense the information in fewer parameters and decrease the dimensionality of the problem. To do so, the selected technique is the Principal Component Analysis (PCA), that will be discussed in details in the next section III.

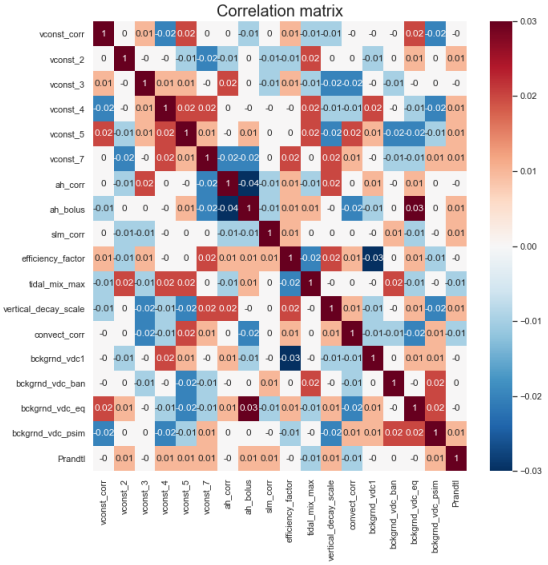

FIG. 5. Correlation matrix for the 18 perturbed parameters. Note that the color scale is set in the narrow range $[-0.03, 0.03]$.

The first step is having an overview on the correlations among data: the more correlations there are, the more the information can be concentrated in fewer variables. The Figure 5 shows though that the data are very little correlated and any significant correlation can be found (the highest is 0.04). Note that the color scale is very narrow to appreciate the small differences. In this specific case, applying the PCA can lead to little improvements in terms of dimensionality reduction. This is indeed confirmed by performing the PCA setting different cumulative variances, instead of choosing *a priori* the number of Principal Components (see Section III). With a cumulative variance of 95%, all the 18 features are kept and even lowering the variance the result doesn't enhance the possibility of reducing the dimensionality. This then lead to the conclusion that in this specific case it is not possible to reduce the number of variables, otherwise too much information would be lost.

III. THEORY

In this section, some background theory is presented to better understand the Machine Learning techniques implemented in this work, together with insights on the algorithms of each technique. Since in this work the code used methods from Scikit-Learn [5] and Tensorflow [6], the algorithms are just sketched theoretically (reason why they are in this section), but we suggest to refer to the documentation of these packages for specific implementations of these algorithms.

A. Principal Component Analysis

Principal component analysis (PCA) is a statistical technique for dimension reduction. In practice, it is used when there are variables related to each other within a dataset and the aim is to reduce the number of data by losing the least amount of information possible. The PCA has the objective of maximizing the variance, calculating the weight to be given to each input variable in order to condense them in one or more new variables (called Principal Components) that will be the linear combination of the original variables. The accurancy score will decrease because we loose little information, but reducing the dimensionality can lead to many advantages, such as computational efficiency and specific model performance (Ref. [7], [8]).

Principal Components The Principal components are new variables that are constructed as linear combinations of the initial variables. These combinations are done in such a way that the new variables are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. So, the idea is p-dimensional data gives you p principal components, but PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on, until having something, as shown in Figure 6.

Organizing information in principal components this way, will allow to reduce dimensionality without losing much information, and this by discarding the components with low information and considering the remaining components as your new variables (Ref. [8]).

An important thing to realize here is that the principal components don't have any real meaning since they are constructed as linear combinations of the initial variables.

Geometrically speaking, principal components represent the directions of the data that explain a maximal amount of variance, that is to say, the lines (or hyperplane in high dimension space) that capture most information of the data (Figure 7, Ref. [9]).

PCA steps The process to find the principal components can be divided into five steps (Ref. [8]):

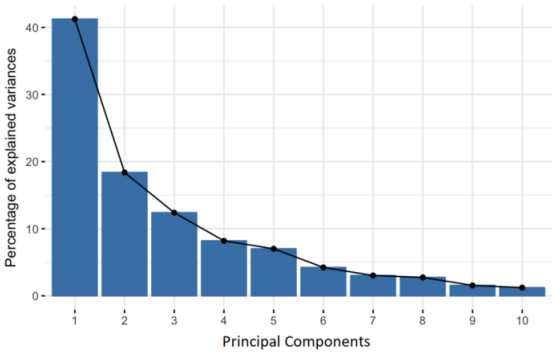

FIG. 6. Percentage of Variance (Information) for each by principal components. The first two are here selected and account for about 60% of the total variance (Ref. **??**).

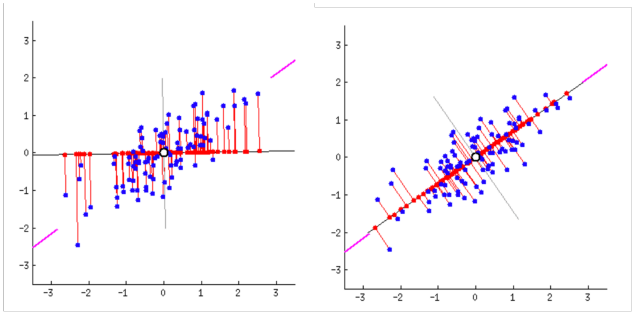

FIG. 7. Geometrical interpretation of the first principal component. It accounts for the largest possible variance in the dataset and it represented by the line that matches the purple marks because it goes through the origin and it's the line in which the projection of the points (red dots) is the most spread out (i.e. maximize the variance) (Ref. [8]).

1. **Standardization:** standardize the range of initial variables
2. **Correlation matrix**: compute the correlation matrix
3. **Eigenvalues and eigenvectors of the correlation matrix:** compute the eigenvalues and eigenvectors of the correlation matrix to identify the principal components
4. **Feature vector:** create a feature vector to decide which principal components to keep
5. **Recasting:** recast the data along the principal components axes

1. Standardization The aim of this step is making all continuous initial variables equally contributed for the analysis, otherwise if variables range on different scales they lead to biased results.

$$x = \frac{x - \bar{x}}{\sigma} \tag{1}$$

where $\bar{x}$ is the mean value and σ is the standard deviation.

2. Correlation matrix In order to reduce the dimensionality of the problem and gather information, a preliminary step is finding correlations among data. The more variables are correlated the more information of the original dataset is redundant. In order to identify these correlations, the correlation matrix needs to be computed (**??**):

$$\boldsymbol{C}[\boldsymbol{x}] = \begin{bmatrix} 1 & \mathrm{corr}[\boldsymbol{x}_0, \boldsymbol{x}_1] & \dots & \dots & \mathrm{corr}[\boldsymbol{x}_0, \boldsymbol{x}_{p-1}] \\ \mathrm{corr}[\boldsymbol{x}_1, \boldsymbol{x}_0] & 1 & \dots & \dots & \mathrm{corr}[\boldsymbol{x}_1, \boldsymbol{x}_{p-1}] \\ \mathrm{corr}[\boldsymbol{x}_2, \boldsymbol{x}_0] & \mathrm{corr}[\boldsymbol{x}_2, \boldsymbol{x}_1] & \dots & \dots & \mathrm{corr}[\boldsymbol{x}_2, \boldsymbol{x}_{p-1}] \\ \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots \\ \mathrm{corr}[\boldsymbol{x}_{p-1}, \boldsymbol{x}_0] & \mathrm{corr}[\boldsymbol{x}_{p-1}, \boldsymbol{x}_1] & \dots & \dots & 1 \end{bmatrix}$$

where x is the matrix containing all the variables by column, p is the number of variables (or fautures) and

$$\mathrm{corr}[\boldsymbol{x}, \boldsymbol{y}] = \frac{\mathrm{cov}[\boldsymbol{x}, \boldsymbol{y}]}{\sqrt{\mathrm{var}[\boldsymbol{x}]\mathrm{var}[\boldsymbol{y}]}}.$$

$$\mathrm{cov}[\boldsymbol{x}, \boldsymbol{y}] = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \overline{x})(y_i - \overline{y}),$$

$$\mathrm{var}[\boldsymbol{x}] = \frac{1}{n}\sum_{i=0}^{n-1}(x_i - \overline{x})^2.$$

Each $\mathrm{corr}[\boldsymbol{x}, \boldsymbol{y}]$ spans in $[-1, 1]$ and mapping $\boldsymbol{C}[\boldsymbol{x}]$ is then possible to appreciate which variables are more or less correlated, checking higher absolute correlation values or 0 respectively.

3. Eigenvalues and eigenvectors of the correlation matrix The eigenvectors of the correlation matrix represent the directions of the axes where there is the most variance(i.e. the "principal components"), while the eigenvalues are the coefficients related to each eigenvector, which give the amount of variance carried. By ranking the eigenvectors in order of their eigenvalues, the principal components are then listed in order of significance.

4. Feature vector The feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep and has as feutures dimension $l < p$ (i.e. dimensionality reduction).

5. Recasting In the previous steps, the selection of the principal components is done, but the input data set remains always in terms of the original axes (i.e, in terms of the initial variables). In order to reorient the data from the original axes to the new ones, it simply needs to multiplying the transpose of the original data X^T set by the transpose of the feature vector V^T:

$$X^* = X^T * V^T \tag{2}$$

An important point to underline is that this method is more effective (i.e. least number of PC) when the data are highly correlated, so the very first principal components gather the far majority of the information (i.e. variance, Figure 6). If this doesn't happen, another way of seeing the problem is choosing the number of the PC according to the total variance is wished to be covered (e.g. first 4 components account for the 95% of the variance) (Ref. [10]).

B. Logistic Regression

Logistic regression is a class of regression analysis where the output is dichotomous (binary) or multicategorical, which cannot be modelled with a linear method (Ref. [11]). Linear regression gives a continuous output, while logistic regression provides a constant output. This is why the continuous function $f(x)$ that models the linear outcome turns into a probability distribution $p(x)$ in logistic regression, as shown in Figure 8. The choice of the probability function is arbitrary, but the most common one is the *sigmoid* function, often called "logistic function" indeed:

$$p(x) = \frac{1}{1 + e^{-x}},$$

The Logistic regression is one of the most simple and commonly used Machine Learning algorithms and its basic fundamental concepts are constructive in deep learning (Ref. [12]).

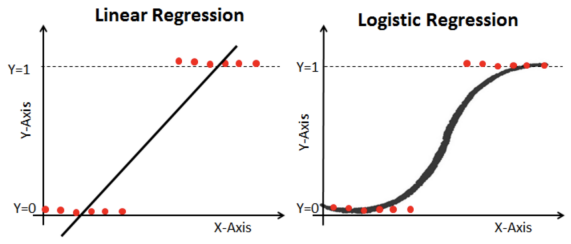

FIG. 8. Limitations of Linear Regression in modelling binary output versus a qualitative sigmoid function fitting properly the data (Ref. [12]).

C. Deep Neural Network

The basic idea of Deep Neural Network lies on building a system of interconnected units, called *neurons*, divided

in multiple layers, that broadcast signals throughout the neural system. The structure is composed of an input layer, one or more hidden layers, with different number of neurons in each of them, and the output layer. The broadcasting of the signal happens thanks to *activation functions* that elaborate the input signals for each neuron into the ouput ones. They are coupled with weights and biases associated to each neuron so that the output is more or less relevant when broadcasted to the next layer (Figure 9). This process starts from the input layer and it runs thoughtout each neuron of each layer, until resulting into an output layer, that is the model prediction (i.e.*Feed Forward Neural Network*). The next step is to *train* the network. Given already the expected *true* output, it is compared to the model prediction thanks to a *cost function* that evaluates the performance of the model. The aim is to minimize this value and using the *backpropagation* and Stochastic Gradient Descent (SGD) algorithms, is possibile to optimize the parameters (i.e. weights and biases) to have the closest result to the expected one. This last step is extremely important because then the network is able to *predict* the results given any other input data. The overall performance of the network is evaluated thanks to the *metrics* on a test dataset (Ref. [9]). For any further information about FFNN, backpropagation algorithm, activation functions, SGD optimizers, cost functions, metrics look at the previous report "Report 2: Feed Forward Neural Network" [13], where information has already been summarized and gathered.

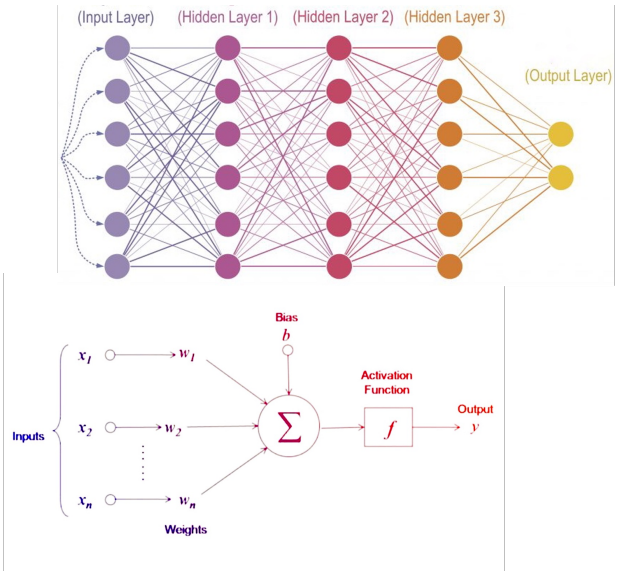

FIG. 9. Examples of Neural Network architecture, with several neurons divided into the hidden layers and with a focus on one single neuron and its schematic operation.

D. Support Vector Machine

The Support Vector Machine (SVM) is another supervised learning technique, used for both regression and classification problems, even though it is mainly known for classification (SMC). Unlike most algorithms, SVM makes use of a hyperplane which acts as a decision boundary between the various classes. The optimal hyperplane is the one that has the maximum distance, called margins, from the closest datapoints from each class, known as Support Vectors [14]. The basic idea is shown in Figure 10.

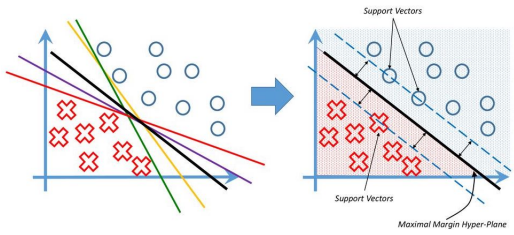

FIG. 10. Schematic idea of the optimal hyperplane in SVM (Ref. [15])

The SVM can also be used for classifying non-linear type of data using kernels, such as Linear, RBF, Sigmoid, Poly. These kernels are used in transforming the non-linear data into an high dimensional feature space, in this way an hyperplane will be created between various classes of non-linear data (Ref. [14], [16]), as shown in Figure 11. For any further information on this topic check Ref. [17].

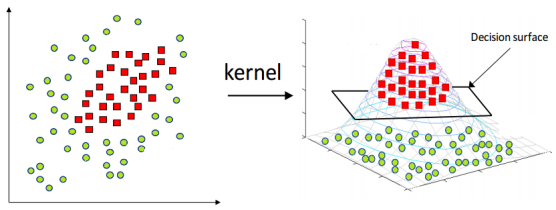

FIG. 11. The kernel trick that transforms the input space into a higher dimensional space, called the feature space, where a linear separation can be performed (Ref. [17]).

IV. METHODS

This work aims to evaluate and compare the performance of three Machine Learning techniques. As a classification problem is here analysed, the accurancy score, the confusion matrix and the ROC metric are employed to quantify the performance. This section will first present the code implementation of each Machine Learning technique together with the metrics used to evaluate the performance of the models.

A. Code implementation

During the previous projects, we implemented our own codes to solve the problems, while in the present work, my intention is to explore and use the Scikit-Learn [5] and Tensorflow [6] packages. The reasons rely on the practical use of the Machine Learning techniques I will have after this course: these packages offer a wide range of ML methods with several functionalities, the code is very versatile and it has been widely used and tested, compared to the one that I made for the previous projects that is made from scratch and the quality is much lower. I take here the opportunity to explore and master these packages better for my future utilization. Another reason is about the conclusions I have taken from the previous projects: the far majority of the time and energy was employed in implementing the code and especially in solving computational problems, while very little time was left for the proper analysis of the performance. I learnt a lot from the previous projects and in spite of my passion for coding in this work I would like to concentrate on something new.

In order to perform the PCA as described in Section III A, the `PCA` method from Scikit-Learn has been applied, but instead of specifying the number of principal components, *n_components* is set to be a float between 0.0 and 1.0, indicating the ratio of variance to preserve. For the ML models `LogisticRegression` and `SVC` has been used from the Scikit-Learn package for the Logistic Regression and Classifier Support Vector Machine respectively, while for building the Neural Network the functionalities in Keras [18] has been applied. For further details, the source code can be found on GitHub in the Machine Learning repository.

B. Metrics

In order to evaluate and compare the performance of the three techniques, the accuracy score, the confusion matrix and the ROC curve are the chosen metrics. Before going into the definition of these metrics, we define some classification parameters (Ref. [19]), common to the three metrics :

- **TP - True Positive**: outcome where the model correctly predicts the positive class;
- **TN - True Negative**: outcome where the model correctly predicts the negative class
- **FP - False Positive**: outcome where the model incorrectly predicts the positive class
- **FN - False Negative**: outcome where the model incorrectly predicts the negative class.
- **TPR - True Positive Rate**:

$$TPR = TP/(TP + FN)$$

- **FPR - False Positive Rate**:

$$FPR = FP/(FP + TN)$$

Accuracy score

$$\text{Accuracy} = \frac{\sum_{i=1}^{n} I(t_i = y_i)}{n},$$

Here t_i represents the target, y_i the outputs of the model, n is the number of targets t_i and I is the indicator function: 1 if $t_i = y_i$ and 0 otherwise (Ref. [9]).

Confusion matrix A confusion matrix is a technique for evaluating the performance of a classification algorithm, where the number of correct and incorrect predictions are summarized. This technique can give more insights than the accuracy score, especially when there is unequal number of observations in each class (Ref. [20]). The Figure 12 shows the structure of the confusion matrix.

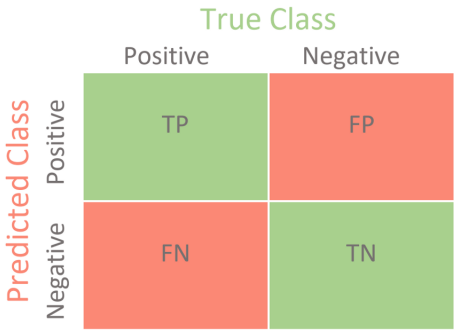

FIG. 12. Typical structure of the confusion matrix on a binary classification problem (Ref. [21])

ROC curve An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots the TPR on the x-axes and the FPR on the y-axes (Ref. [9], [22]). The ideal curve is a sharp 90° angle as shown in Figure 13.

In order to implement these metrics the methods score, confusion_matrix, roc_curve have been used from the Scikit-Learn package. All the source code can be found on GitHub in the Machine Learning repository.

V. RESULTS AND DISCUSSION

As mentioned in Section II, a preliminary analysis has been performed on the original dataset. A dimension

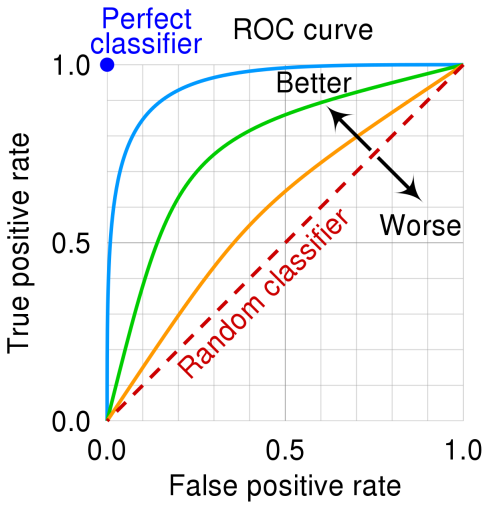

FIG. 13. ROC curves showing performance of different classifiers (Ref. [23])

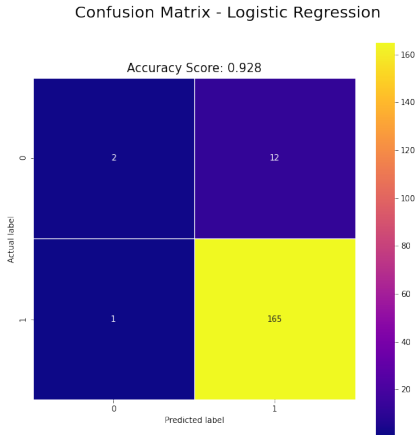

FIG. 14. Confusion matrix of the Logistic Regression. The upper left cell indicates the True Positives (TP) that are the right predicted failures, the upper right represents the False Negatives (FN) that are the missed failures, the bottom left is the False Positive (FP) that stands for the incorrectly predicted failures and the bottom right that are the right predicted success or True Negative (TN).

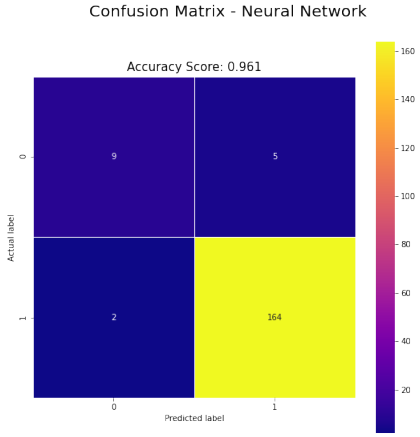

FIG. 15. Confusion matrix of the Neural Network. The upper left cell indicates the True Positives (TP) that are the right predicted failures, the upper right represents the False Negatives (FN) that are the missed failures, the bottom left is the False Positive (FP) that stands for the incorrectly predicted failures and the bottom right that are the right predicted success or True Negative (TN).

reduction was tried, applying the Principal Component Analysis taken as cumulative variance 95%, 70%, 55%, giving respective number of components: 18, 13, 10. This led to the trade off of choosing between reducing the number of features loosing lots of information or keeping information but also the high number of dimensions. Since this is not an extreme case in which we loose so much of the computer efficiency, keeping all the 18 features resulted to be the choice and the following analysis was performed on a training dataset of dimension (360, 18), i.e. Study 1 and 2, and a test dataset of dimension (180, 18), i.e. Study 3.

For the Logistic Regression (LR) and SVM default parameters are chosen, the Neural Network (NN) is composed by three hidden layers with the *ReLU* as activation function and *Adam* as optimizer (look at the previous report for further details on these parameters [13]). In order to compare the performance of the three models, we refer to the accuracy scores and confusion matrices in Figures 14, 15, 16, while the ROC comparison is shown in Figure 17. The accuracy scores for the LR, for the NN and for the SVM are respectively of 92.8%, 96.1%, 92.2%, meaning that all the three models perform very well in the overall performance. Nevertheless, if we have closer look at the confusion matrices, we can actually see that they basically predict very few failures of the climate simulations: 2 for LR, 9 for NN, 1 for SVM, when the real number expected was 14. Note that TP (upper left) stands for actual failures, FN (upper right) stands for missed failures and so on. This behaviour is understandable because the test sample is quite small combined with the fact that there is a very high inequality in the number of failures and successes in the already small sample. We are comparing something that relatively speaking leads to a big error (e.g. for LR $|2-14|/14 = 85\%$), but speaking in absolute terms there is little distance compared to the whole sample (e.g. for LR $|2-14|/180 = 7\%$). This is why the accuracy score in very high in the three models, shown also by the ROC curve, where all the three models are very close to the ideal sharp curve (Figure 13). Going back to the objective of this analysis though, this is using ML techniques to predict the simulations failures.

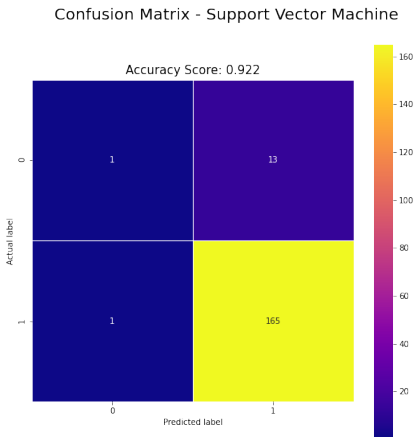

FIG. 16. Confusion matrix of the Support Vector Machine. The upper left cell indicates the True Positives (TP) that are the right predicted failures, the upper right represents the False Negatives (FN) that are the missed failures, the bottom left is the False Positive (FP) that stands for the incorrectly predicted failures and the bottom right that are the right predicted success or True Negative (TN).

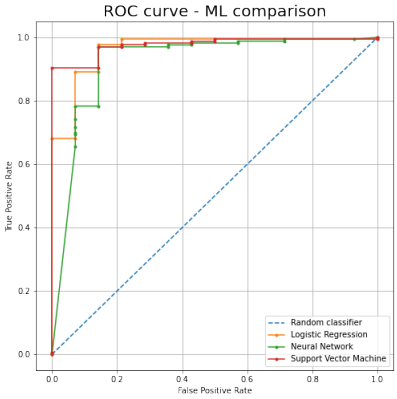

FIG. 17. ROC curves by comparison for the LR (orange), NN (green) and SVM (red).

This then mean that the only model that reaches a close result is the NN, while the other two fail in this goal. The reasons why the NN perfoms better than the LR are that the first is a very more sofisticated techinque than the latter. As SVM is concerned, the result is instead quite surprising, because this technique would be eligible for high dimensional spaces, as it is in the present case (Ref. [24]). Further investigations need to be conducted.

The second step of the analysis involves "optimizing" the NN, i.e. comparing the performance of the NNs setting different activation functions and optimizers. We can see from Table I that all the accuracy scores are very

Activation functions VS Optimizers				
	sgd	*adagrad*	*rmsprop*	*adam*
relu	0.961	0.961	0.967	0.967
sigmoid	0.922	0.978	0.972	0.961
tanh	0.967	0.967	0.950	0.950
elu	0.978	0.956	0.878	0.961

TABLE I. Accuracy scores of Neural Networks with different activation functions and optimizers.

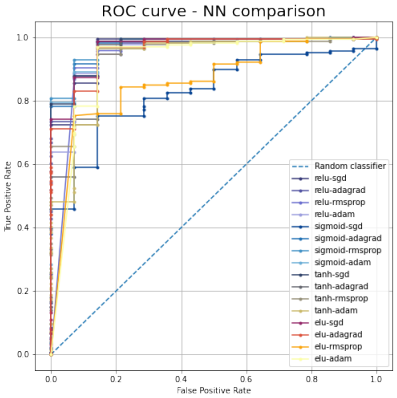

FIG. 18. ROC curves for different NN, changing the activation functions (*'relu', 'sigmoid','tanh','elu'*) and the optimizers *'sgd', 'adagrad','rmsprop','adam'*).

high and so the ROC curves are all sharp (Figure 18). Nevertheless, we can also notice little differences. What in every run of the code is constant is the worst performance of the combinations *'elu-adagrad'* and *'sigmoid-sgd'*, which have respectively 87.8% and 92.2% of accuracy scores and are the ones' curves that evidently perform worst in the ROC plot. It is hard to choose an eligible best combination, because lots of them perform very similarly. Even running the code several times, the best one keep changing, but the *'elu-sgd'* is always on the top of the ranking. The reasons behind these behaviours are not completely clear but this grid analysis was needed to have further insights on the NN optimal parameters perhaps for future research. In the Appendix all the confusion matrices are collected in Figure 21.

VI. CONCLUSIONS AND OUTLOOK

In this report we explored different Machine Learning techniques (Logistic Regression, Deep Neural Network and Support Vector Machine) by comparison in performing a classification problem on the UCI dataset Climate Model Simulation Crashes [1]. The evaluation of the performance was done by evaluating the accuracy score, the confusion matrix and the ROC curve. For the first and

last metrics all the three models seem to have an excellent behaviour but looking into the confusion matrices, it is possible to appreciate that only the Neural Network succeed in predicting when the simulations crashes could happen. An optimization of the Neural Network has been carried out, varying the activation functions and the optimizers. For several combinations, the networks perform very well, except for *'elu-adagrad'* and *'sigmoid-sgd'*.

The work underlying this report was aiming to compare the Machine Learning techniques, but reading the Lucas D.D. et al.'s study [2] other ideas out of the report objectives can be explored in further investigations, such as the sensitivity analysis, reproducing for instance Figure 2). In addition, during this work default parameters of the Sklearn's methods have been considered. exploring the parameters' space would be very interesting in future studies.

APPENDIX

	Parameter[a]	[low, default, high]	Scale[b]	Module	Description
1	vconst_corr	$[0.3, 0.6, 1.2] \times 10^{7}$	lin	hmix_aniso	variable viscosity parameter (vconst_1, vconst_6)
2	vconst_2	[0.25, 0.5, 2.0]	log	hmix_aniso	variable viscosity parameter
3	vconst_3	[0.16, 0.16, 0.2]	lin	hmix_aniso	variable viscosity parameter
4	vconst_4	$[0.5, 2.0, 10.0] \times 10^{-8}$	log	hmix_aniso	variable viscosity parameter
5	vconst_5	[2, 3, 5]	lin	hmix_aniso	variable viscosity parameter
6	vconst_7	[30.0, 45.0, 60.0]	lin	hmix_aniso	variable viscosity parameter
7	ah_corr	$[2.0, 3.0, 4.0] \times 10^{7}$	lin	hmix_gm	diffusion coefficient for Redi mixing (ah) and background horizontal diffusivity within the surface boundary layer (ah_bkg_srfbl)
8	ah_bolus	$[2.0, 3.0, 4.0] \times 10^{7}$	lin	hmix_gm	diffusion coefficient for bolus mixing
9	slm_corr	[0.05, 0.3, 0.3]	log	hmix_gm	maximum slope for bolus (slm_b) and Redi terms (slm_r)
10	efficiency_factor	[0.05, 0.07, 0.1]	lin	mix_submeso	efficiency factor for submesoscale eddies
11	tidal_mix_max	[25.0, 100.0, 200.0]	log	tidal	tidal mixing threshold
12	vertical_decay_scale	$[2.5, 5.0, 20.0] \times 10^{4}$	log	tidal	vertical decay scale for tide induced turbulence
13	convect_corr	$[1.0, 10.0, 50.0] \times 10^{3}$	log	vertical_mix	tracer (convect_diff) and momentum (convect_visc) mixing coefficients in diffusion option
14	bckgrnd_vdc1	[0.032, 0.16, 0.8]	log	vmix_kpp	base background vertical diffusivity
15	bckgrnd_vdc_ban	[0.5, 1.0, 1.0]	lin	vmix_kpp	Banda Sea diffusivity
16	bckgrnd_vdc_eq	[0.01, 0.01, 0.5]	log	vmix_kpp	equatorial diffusivity
17	bckgrnd_vdc_psim	[0.1, 0.13, 0.5]	log	vmix_kpp	maximum PSI induced diffusivity
18	Prandtl	[4.0, 10.0, 20.0]	log	vmix_kpp	ratio of background vertical viscosity and diffusivity

[a] Individual `_corr` parameters (numbers 1, 7, 9, and 13) are used to represent the correlated pair of parameters given in the description. For example, values drawn for `vconst_corr` are assigned to `vconst_1` and `vconst_6`. [b] Linear and logarithmic scales are used for parameter ranges that have ratios of high/low < 5 and high/low ≥ 5, respectively.

FIG. 19. Complete description of the 18 features (Ref. [2])

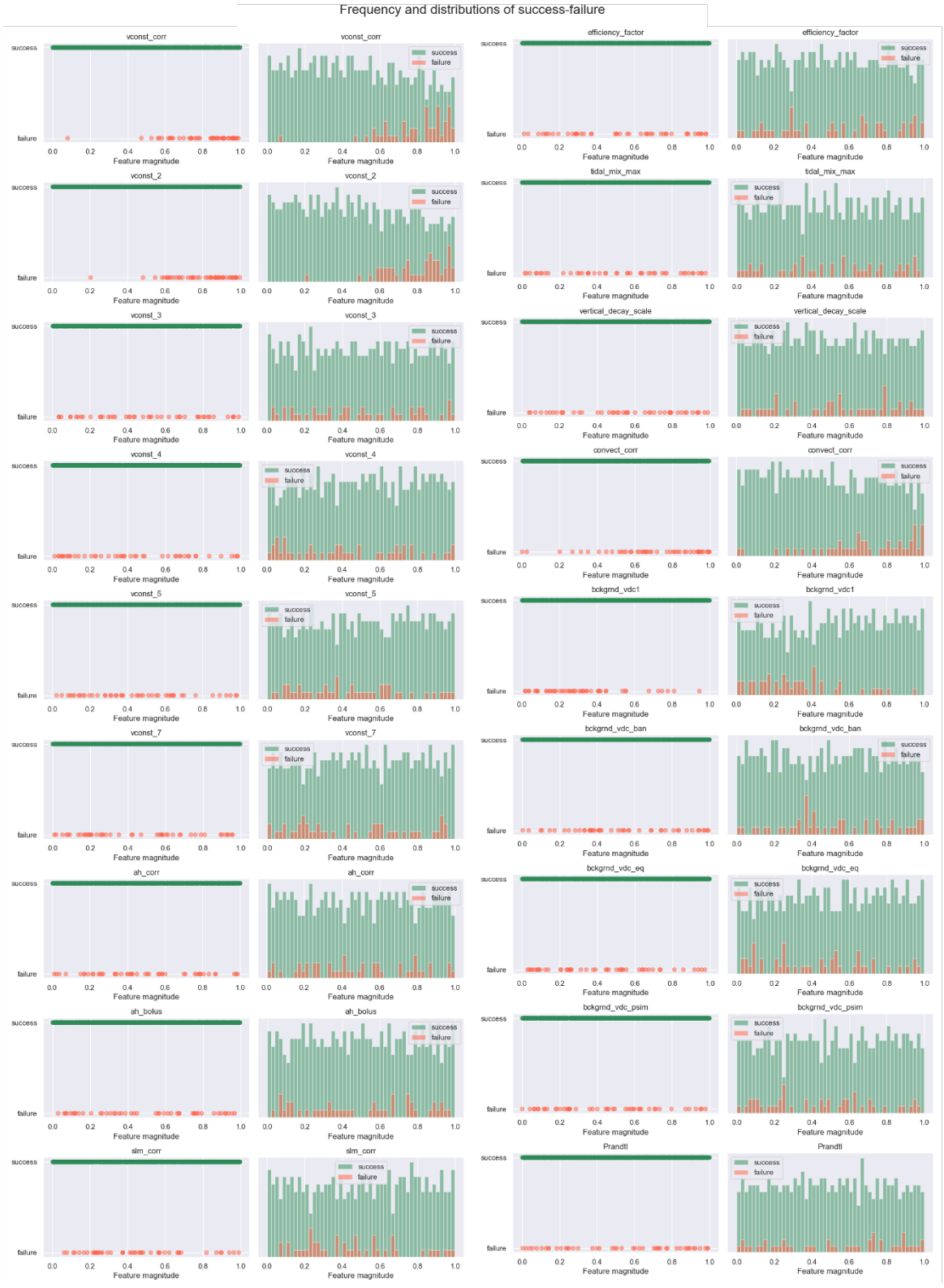

FIG. 20. Total plot of the outcomes frequency and distribution for all 18 parameters.

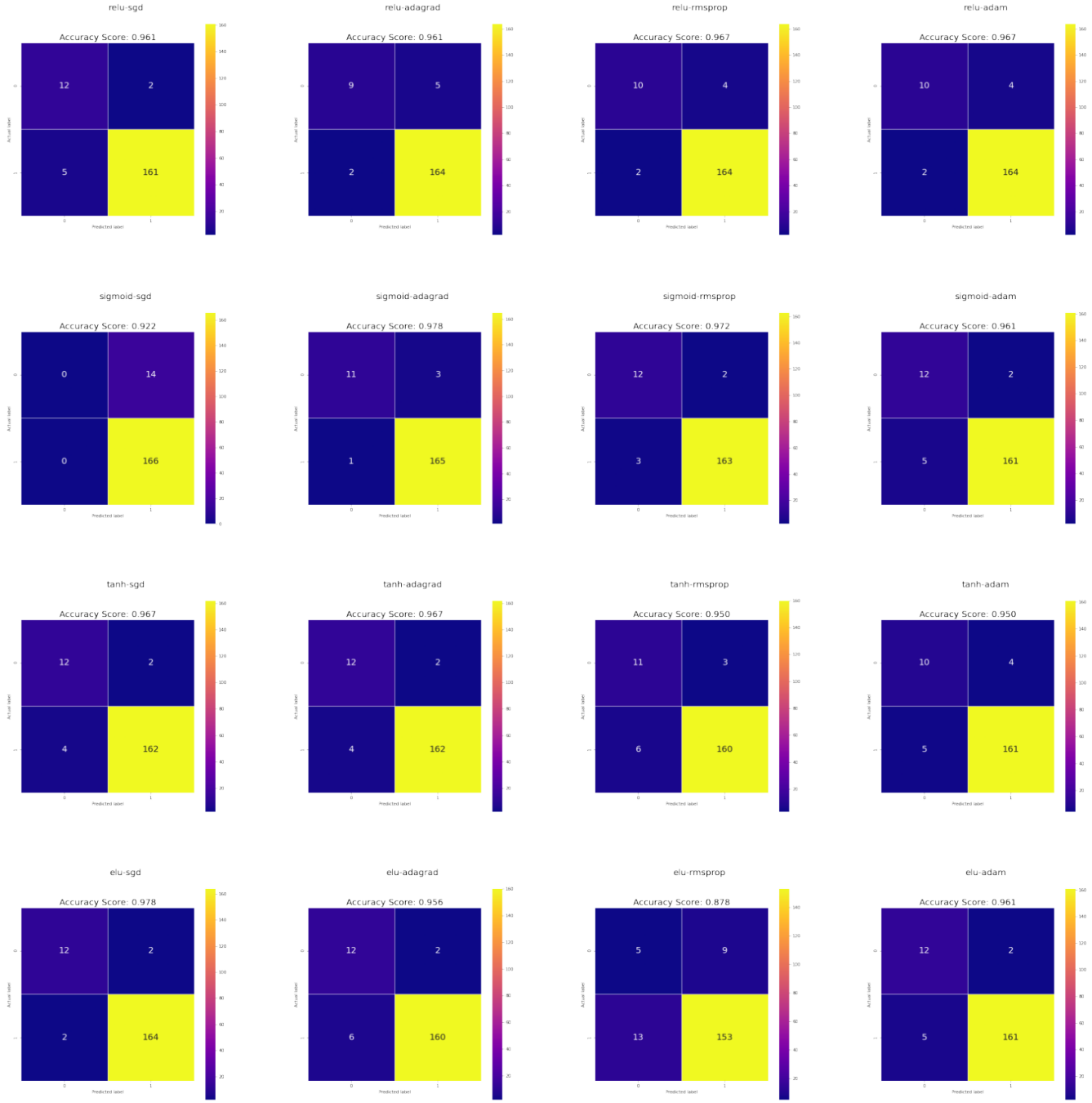

FIG. 21. Collection of all the 16 confusion matrix for the combination of the 4 activation functions (*'relu', 'sigmoid','tanh','elu'*) and the 4 optimizers *'sgd', 'adagrad','rmsprop','adam'*).

REFERENCES

[1] UCI Machine Learning Repository. Climate model simulation crashes data set. https://archive.ics.uci.edu/ml/datasets/Climate+Model+Simulation+Crashes, Last accessed on 2021-12-16.

[2] D. D. Lucas, R. Klein, J. Tannahill, D. Ivanova, S. Brandon, D. Domyancic, and Y. Zhang. Failure analysis of parameter-induced simulation crashes in climate models. *Geoscientific Model Development*, 6(4):1157–1171, 2013.

[3] Wikipedia. Latin hypercube sampling. https://en.wikipedia.org/wiki/Latin_hypercube_sampling. Last accessed on 2021-12-16.

[4] Jose J. Muñoz, Zhi-zhao Liu, Wei Li, and Ming Yang. Two general extension algorithms of latin hypercube sampling. *Mathematical Problems in Engineering*, 2015:450492, 2015.

[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*, 12:2825–2830, 2011.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[7] Somia B. Mohammed , Ahmed Khalid, Saife Eldin F. Osman , Rasha Gaffer . M. Helali. 2016.

[8] Zakaria Jaadi. A step-by-step explanation of principal component analysis (pca), 2021. https://builtin.com/data-science/step-step-explanation-principal-component-analysis. Last accessed on 2021-12-16.

[9] Morten Hjorth-Jensen. Fys-stk4155 - applied data analysis and machine learning course notes), 2021. https://compphysics.github.io/MachineLearning/doc/web/course.html.

[10] Paola Pozzolo. Analisi delle componenti principali, 2020. https://paolapozzolo.it/analisi-delle-componenti-principali-criteri/. Last accessed on 2021-12-16.

[11] Complete Dissetion by statistics solutions. Logistic regression. https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/logistic-regression/. Last accessed on 2021-12-16.

[12] Avinash Navlani. Understanding logistic regression in python, 2019. http://www.who.int/mediacentre/factsheets/fs282/fr/. Last accessed on 2021-12-16.

[13] Adele Zaini. Report 2: Feed forward neural network, 2021. https://github.com/adelezaini/MachineLearning/blob/master/Projects/Reports/Report2/Report2_Adele_Zaini.pdf.

[14] Vaibhav Paliwal. Comparing machine learning algorithms on a single dataset(classification), 2020. https://medium.com/@vaibhavpaliwal/comparing-machine-learning-algorithms-on-a-single-dataset-classification-46ffc5d3f278. Last accessed on 2021-12-16.

[15] Anis Marshall. Support vector machine(s.v.m) — classifiers and kernels. https://slideplayer.com/slide/16227619/. Last accessed on 2021-12-16.

[16] William S Noble. What is a support vector machine?, 2006. https://www.ifi.uzh.ch/dam/jcr:00000000-7f84-9c3b-ffff-ffffc550ec57/what_is_a_support_vector_machine.pdf. Last accessed on 2021-12-16.

[17] Apurv Jain. Support vector machine(s.v.m) — classifiers and kernels, 2020. https://medium.com/@apurvjain37/support-vector-machine-s-v-m-classifiers-and-kernels-9e13176c9396. Last accessed on 2021-12-16.

[18] Francois Chollet et al. Keras, 2015.

[19] Machine Learning Crash Course. Classification: True vs. false and positive vs. negative, 2020. https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative. Last accessed on 2021-12-16.

[20] Jason Brownlee. What is a confusion matrix in machine learning, 2020. https://developers.google.com/machine-learning/crash-course/classification/true-false-positive-negative. Last accessed on 2021-12-16.

[21] Joydwip Mohajon. Confusion matrix for your multi-class machine learning model, 2020. https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826. Last accessed on 2021-12-16.

[22] Machine Learning Crash Course. Classification: Roc curve and auc, 2021. https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc. Last accessed on 2021-12-16.

[23] Wikipedia. Receiver operating characteristic, 2021. https://en.wikipedia.org/wiki/Receiver_operating_characteristic. Last accessed on 2021-12-16.

[24] Dhiraj K. Top 4 advantages and disadvantages of support vector machine or svm, 2019. https://dhirajkumarblog.medium.com/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107. Last accessed on 2021-12-17.