Journal of open research software

Elnesr, M N and Alazba, A A 2017 NeuroCharter: A Neural Networks Software to Visually Discover the Effects and Contributions between Interrelated Features. *Journal of Open Research Software*, 5: 19, DOI: https://doi.org/10.5334/jors.135

SOFTWARE METAPAPER

NeuroCharter: A Neural Networks Software to Visually Discover the Effects and Contributions between Interrelated Features

Mohammad N. Elnesr and A. A. Alazba

Alamoudi chair for water research, King Saud University, SA

Corresponding author: Mohammad N. Elnesr, Assistant Professor (drnesr@gmail.com, melnesr@ksu.edu.sa)

NeuroCharter is an open-source software that helps in prediction problems in scientific research through artificial neural networks. The program is designed mainly for researchers who focus on details of the neural-network's parameters, in addition to easy reuse of the trained network. The program outputs almost all the necessary graphs regarding the network and features contributions and relative outputs for both numeric and categorical features. The program was implemented in Python 2.7.11 and is open sourced for reuse and future development. The program consists of four main classes, one for the neural networks calculation, one for data manipulation, one for plotting the neural network, and the main class that manages and links the other classes. The source code and some experimental data are freely available at the GitHub code repository http://j.mp/NeuroCharter.

Keywords: Artificial Neural Networks; Python; Backpropagation; predictions

Funding Statement: The project was financially supported by King Saud University, Vice Deanship of Research Chairs.

(1) Overview

Introduction

Artificial Neural Networks (ANNs) have proven their reliability in solving complex systems for pattern recognition and multivariate regression [1, 2]. Although the backpropagation technique was developed 40 years back [3], the interest in ANNs increased dramatically in the last decade because of the advances in hardware and the needs of artificial intelligence applications in handheld devices. Most of the uses of ANNs nowadays are for pattern recognition, either for handwriting recognition, image recognition, or more advanced applications like Self-driving cars [4]. However, the regression analysis of ANNs plays the most important role in scientific research especially for economic, biological, and environmental research (e.g. [5–7]). The primary function of the ANN software packages is to perform training, validating, and testing the network, then use the trained network in prediction/recognition of missing features/patterns. Most of the commercial packages dealt with the ANNs as a black box, not allowing the user to analyze/modify the resulted weights and biases, or to make some studies on the effects of some input features on output features, especially when the feature is categorical. In scientific research, after finding the suitable ANN, the researchers need to analyze the inputs/outputs features and their relations to each other and to plot these in order to discuss them in their publications, however, most of the existing packages just perform the training and prediction roles, leaving the analysis and studies for the researcher.

The aim of this work was to develop an open sourced ANN package that can be used easily by researchers to train the ANN, find its optimal structure, reuse the trained ANN in prediction, and plots all the possible relationships between inputs and outputs in a publication quality charts.

Implementation and architecture

The program contains four main classes; the Study Class, the NeuralNetwork Class, Data Class, and PlotNet Class, **Figure 1**. The main class is the Study Class where we can select one of five running modes;

1. To perform a *full run*, where the whole data is used for training the network (no partitioning).
2. To perform a *cross validation*, where the given data will be partitioned into three partitions; about 70% for training, 20% for validation, and 10% for testing. The mentioned partitioning ratios are the defaults, but they can be specified by the user. In this mode, we start by the training data set, but at the end of each epoch, we check the errors of the validation

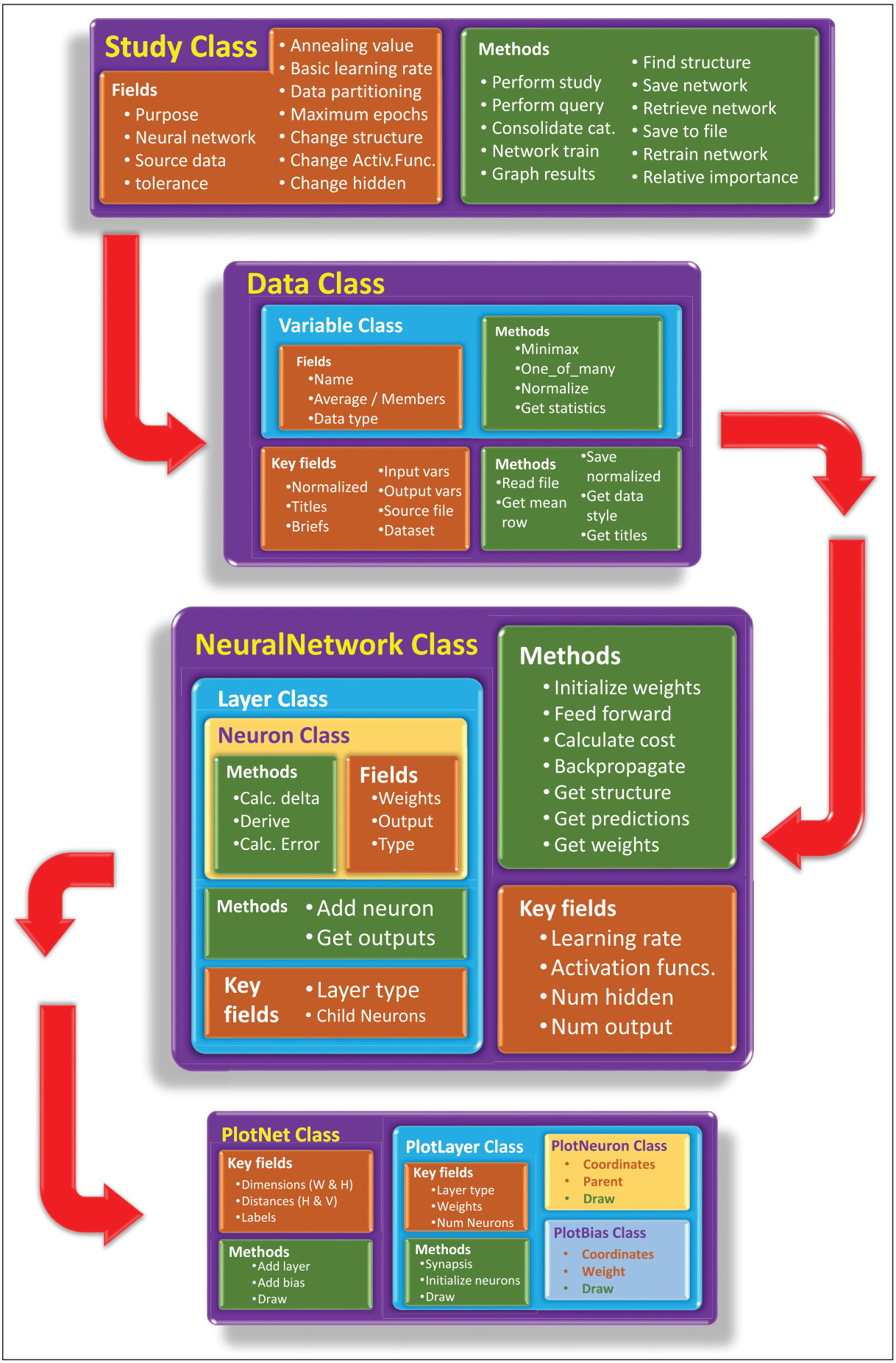

Figure 1: The architecture and workflow of the program.

dataset, if the error of the latter starts to increase instead of decrease, the training will stop as it will be considered an evidence of overfitting. This is the default purpose.

3. To perform a *progressive validation*, where the given data will be partitioned as specified before. In this mode, we start by the validation dataset, from which we specify the stopping epoch, then we launch the training dataset for maximum epochs equals to double the validation epochs. This mode was proved to eliminate overfitting.
4. To perform an *optimization run*, which is similar to the validation run, but involves searching for the best ANN structure and best activation functions before starts training.
5. To perform a *query*, i.e. to predict output features depending on saved ANNs.

In the initiation of the Study class, the Data Class is called to manipulate the data and to specify the suitable ANN structure; then the NeuralNetwork Class is called to construct the ANN accordingly.

The Data Class automatically analyses the given data, determines the type of each variable (either numeric or categorical), then it normalizes the data to be ready for analysis by the NeuralNetwork Class. Additionally, in the case of querying predictions of features, the Data Class is used to check the suitability of each data line, and to normalize or denormalizes outputs. This class contains a sub class Variable, which uses mini-max normalization for numeric features, and 1-of-C dummy encoding for categorical features [8]. Additionally, it provides basic statistics of each feature/variable depending on its type; for numeric features, it calculates minimum, maximum, average, variance, and standard deviation, while for the categorical features, it provides, the members of the category and the percent each one as it appears in the given data. The program allows optional variables' titles as they are placed at the data headers in one or two lines; the first line is for full-captions and the second for brief-captions which are used in graphs.

The NeuralNetwork Class creates and solves neural networks. While creation, the structure of the network is specified according to the normalized data. To ensure fast learning rate, the ANN weights are initialized randomly within the range of $\pm\sqrt{1/\text{Number of inputs to the neuron}}$ as suggested by [9, 10]. The class uses sequential (online) learning mode to manipulate the data as it is more likely to escape from local-minima than the batch mode [11, 12]. This means that the errors are updated after manipulating each data line, not after manipulating the whole dataset as in batch mode. The NeuralNetwork Class consists of the Layer sub-class which specified the layer type (input, hidden, or output), and manages the child neurons by their sub class. The Neuron Class initiate neurons, each with its specific weight, bias, and activation function. It also calculates the deltas and the cost function of each neuron. This helps in customizing neurons according to variables types. As we described earlier, the training is triggered from the Study Class depending on the mode of the class. In the full run, validation run, and optimization run the ANN training starts by feed forward operation, where the inputs of each neuron are calculated as the summation of the product of weights and outputs of the previous layer, while the outputs of the neurons are calculated by smashing the inputs using the activation function. The default activation function is the sigmoid function. However, the program supports twelve different activation functions, **Table 1**. After reaching the output layer, the cost function for each neuron is calculated as $0.5(\text{Neuron target-Neuron output})^2$ then we sum the cost of all output neurons to find the cost that will be used in the backpropagation process.

One of the features of the program is to plot the ANN in an informative way; showing the weights and biases with line thicknesses that reflect their values and sign, **Figure 2**, The PlotNet Class uses the Matplotlib library to draw the network through three sub classes. The core algorithm of ANN plotting is based on Milo et al. [13]. The main class is responsible for specifying the outlines of the network, and the locations of the child components like layers and neurons. The PlotLayer Class draws the synapsis (lines) with thicknesses and colors that reflect the magnitude and sign of the weight where the heavier thickness of the lines imitates higher magnitudes of weights, and the line colors reflect the weight's sign (blue and red for positive and negative). PlotLayer Class also initiates Neurons and biases, sending their information to PlotNeuron and PlotBias classes that are responsible for drawing either. The main difference between the two child classes that the latter is in charge of drawing the bias and its synapses lines, while the former is in charge of drawing the neuron only, while the weights are drawn by the PlotLayer Class as mentioned.

Program outputs

The main output of NeuroCharter is a set of hi-res charts representing almost all that researchers need in publishing their papers. However, NeuroCharter outputs detailed text outputs in CSV formats to help researchers to analyze the results their way. Furthermore, while training, a step by step progress of the program is printed to the console along with execution times which helps to diagnose errors if any. Some of the outputs of NeuroCharter are the following, detailed outputs description in the tutorial file listed at the 'Quality control' section of this paper:

1. `'NrCh_NormData_????.csv'` a list of inputs and output data in normalized form., where the???? is an encrypted timestamp of the execution time.
2. `'NrCh_Weights_????.csv'` a list of weights and biases of the network after training.
3. `'NrCh_Outputs_????.txt'` a continuous file handling all the studied ANNs information including structure, weights, activation functions, and other variables.
4. `'NrCh_StoredANN_????.nsr'` A Neural Structured Repository (*.nsr) file (encrypted)

Table 1: Different activation functions available in the NeuroCharter program.

Name	Formula	Derivative
Sigmoid	$f(x)=1/1+e^{-x}$	$f'(x)=f(x)\times(1-f(x))$
Softmax	$s=e^{x-\max(x_i)_{i=1}^{n}}$ $f(x)=s/\sum_{i=1}^{n}s_i$	$f'(x)=f(x)\times(1-f(x))$
Binary	$f(x)=\begin{cases}0 & x<0\\ 1 & x\geq 0\end{cases}$	$f'(x)=\begin{cases}0 & x\neq 0\\ \infty & x=0\end{cases}$
Soft sign	$f(x)=x/(1+\lvert x\rvert)$	$f'(x)=1/(1+\lvert x\rvert)^2$
Bent identity	$f(x)=x+0.5(\sqrt{x^2+1}-1)$	$f'(x)=1+0.5x/\sqrt{x^2+1}$
Gaussian	$f(x)=e^{-x^2}$	$f'(x)=-2xe^{-x^2}$
Tanh	$f(x)=2/1+e^{-2x}-1$	$f'(x)=1-f(x)^2$
Linear	$f(x)=x$	$f'(x)=1$
Arctan	$f(x)=\mathrm{Atn}(x)$	$f'(x)=1/(1+x^2)$
Sinusoid	$f(x)=\sin(x)$	$f'(x)=\cos(x)$
Soft plus	$f(x)=\ln(1+e^x)$	$f'(x)=1/(1+e^x)$
Sinc	$f(x)=\begin{cases}\sin(x)/x & x\neq 0\\ 1 & x=0\end{cases}$	$f'(x)=\begin{cases}(\cos(x)-\sin(x)/x)/x & x\neq 0\\ 0 & x=0\end{cases}$

where the whole study is saved including the ANN structure, weights, and data limits, thus it can be recalled by NeuroCharter for later predictions.

5. `'NrCh_Clouds_????.csv'` a set of given vs. predicted data by the ANN.
6. `'DataFile_Output.txt'` a list of predicted values of the output features in de-normalized format, where 'DataFile' is the name of the original data file name.
7. `'NrCh_OutputCharts_????.pdf'` all the output charts from NeuroCharter. where the '?' characters are replaced by the current date and time where the study performed. The pdf file basically consists of 6 pages, in addition to 1 page per output variable. The basic pages are:

    a. The cost function development during different stages (training, validation, and testing) **Figure 3**
    b. The full ANN structure diagram, **Figure 2**.
    c. A brief ANN structure where the categorical neurons of each variable are consolidated to one neuron for a better understanding of variables contribution, **Figure 4**.
    d. The relative importance of inputs to outputs, one bar chart with +ve and –ve contributions of each variable, and one pie chart for each output variable, **Figure 5**.
    e. Prediction function and data cloud, the predicted curve vs. the original data for each output variable, **Figure 6**.
    f. Real vs. predicted data, plotting given data vs. predictions on 45° line curve, **Figure 7.**
    g. Effect of each input feature on all output features, one page per output feature. Within each page, there is one chart per output feature. Each chart contains three curves, at 25, 50, 75% of data. **Figure 8** and **Figure 9** for numeric and categorical input features respectively.

Quality control

To understand how the program works, we provide some examples below. However, we provided full docstrings for all functions and classes in addition to informative comments before the key routines.

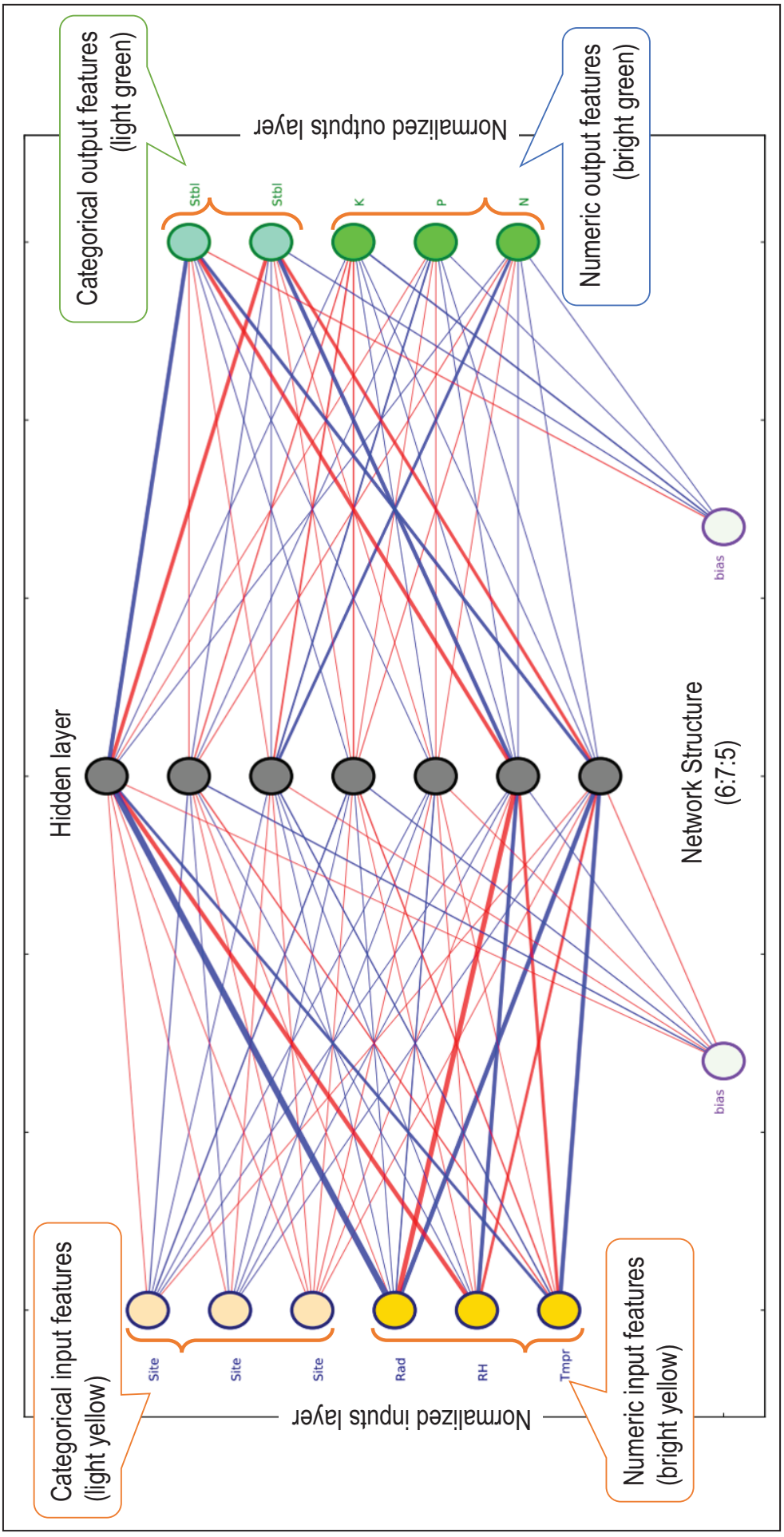

Figure 2: A full ANN diagram showing weights and biases. The heavier thickness of the lines reflects higher magnitudes of weights; the line colors reflect the weight's sign (blue and red for positive and negative). Categorical features' neurons are lighter in color for normalized layout.

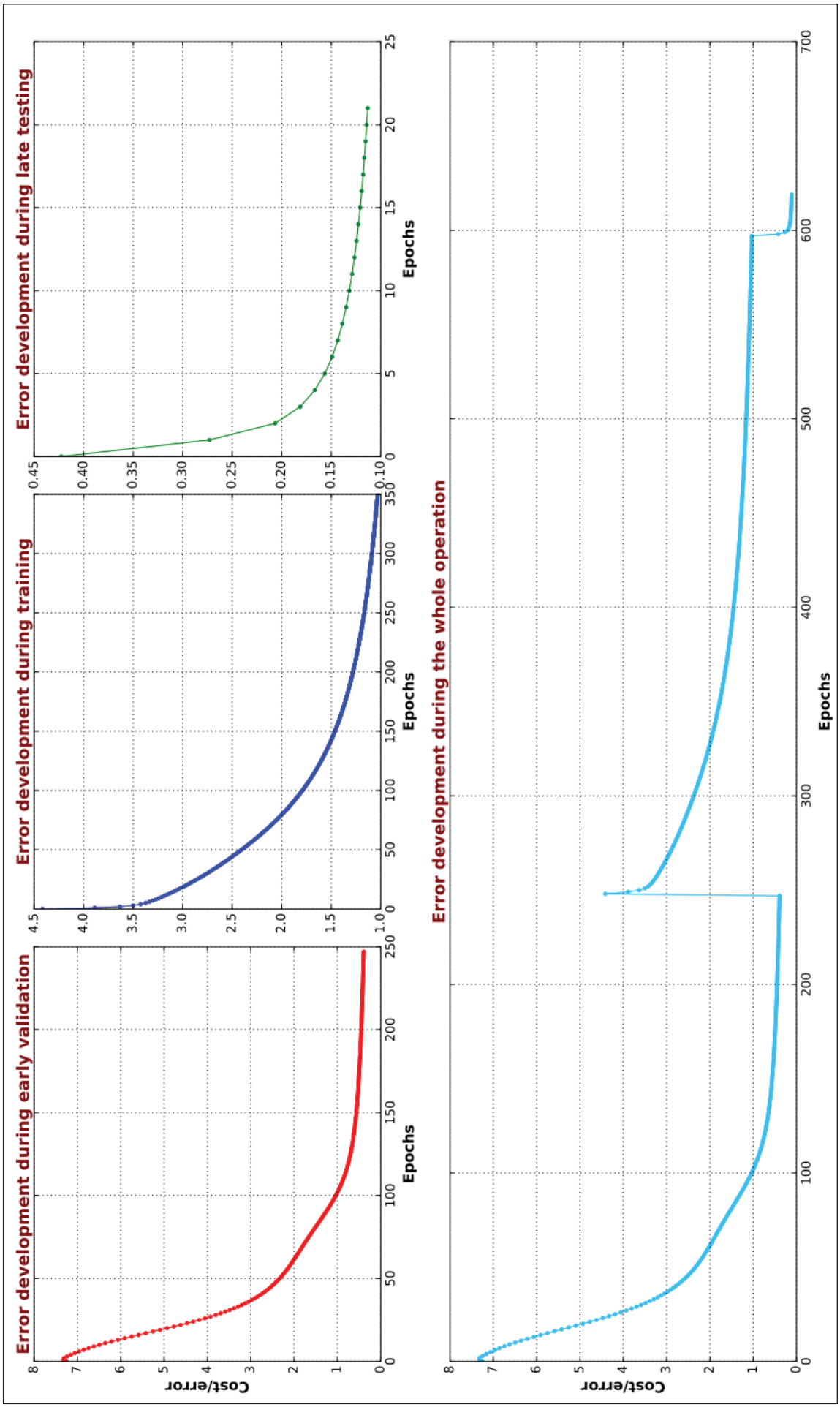

Figure 3: Cost development through deferent stages.

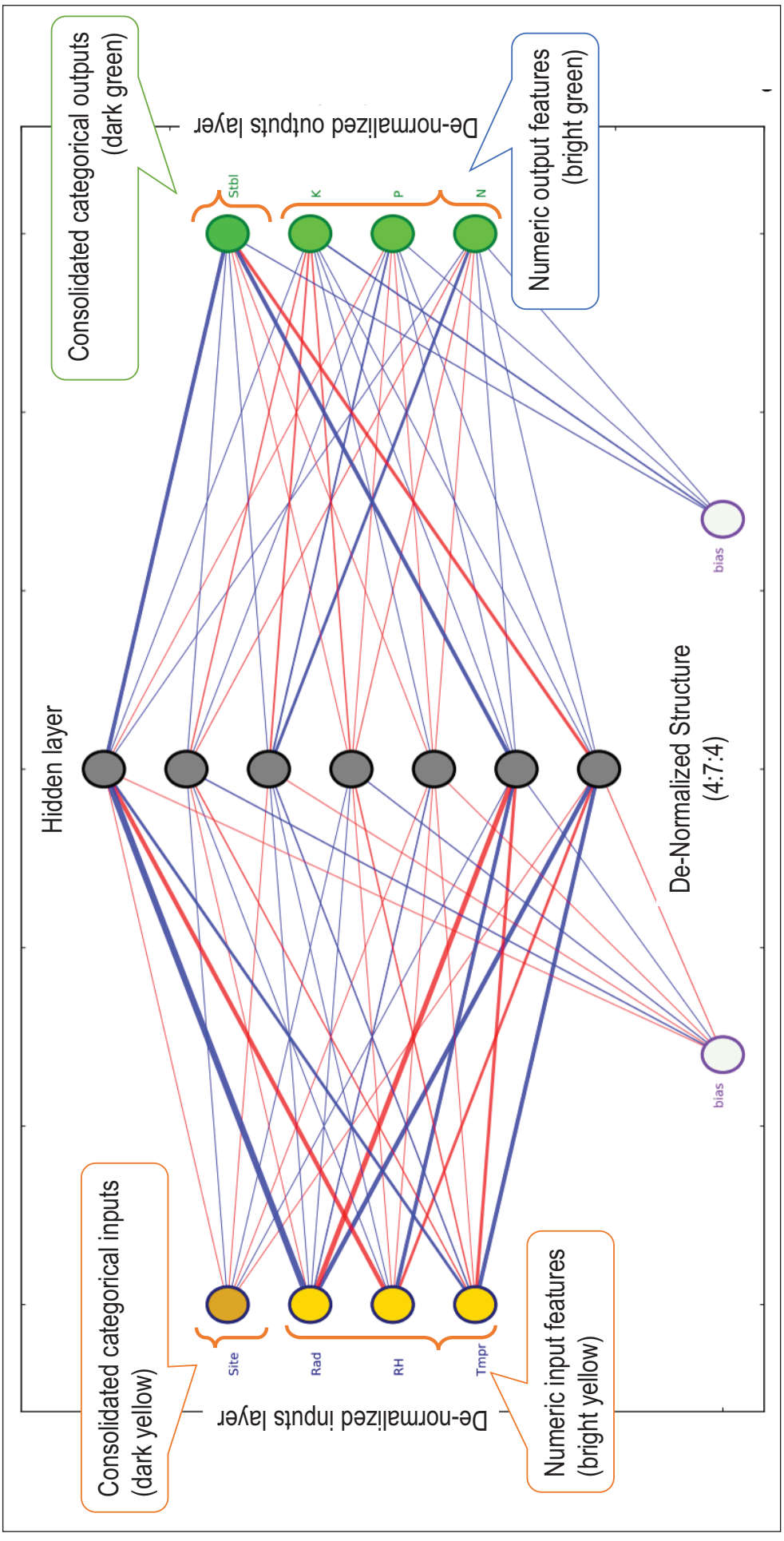

Figure 4: Brief ANN diagram showing weights and biases. Lines and colors are similar to the full ANN diagram, except that categorical features' neurons are heavier in color to reflect consolidation.

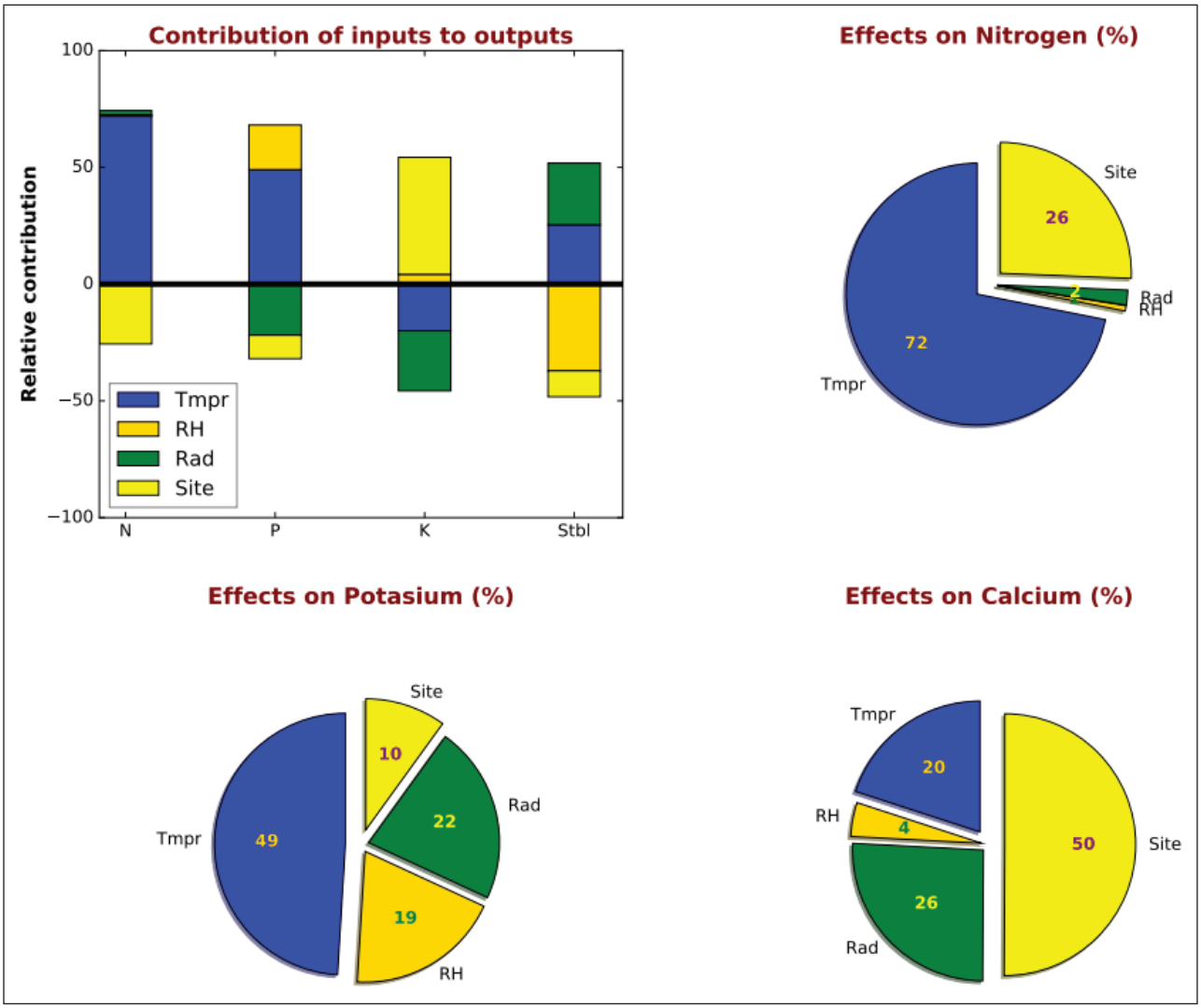

Figure 5: Sample relative importance charts.

Program tutorial

A simple tutorial is located at the GIT page, here: https://goo.gl/TGop2p. Please read it before proceeding to the next examples.

Example for creating and training an ANN

If you have data in CSV file format (say its name is dataNT.csv), to start a Study that involves normalizing data, building an ANN, train it, and visualize results in pdf format, so please consider adding the following code at the end of the program, or you can import the program as is, and add this code after the import statement (*the program and the testing file should be in the same folder, please refer to the instructions at the simple tutorial above*):

```
from NeuroCharter import Study
```

The basic line code that can do this job with the default parameters is:

```
Study('dataNT.csv', num_outputs=4)
```

While for more control of the study's parameters, the user can specify more parameters' values such as the following:

```
Study('dataNT.csv', 'cross validation',
      num_outputs=4, data_partition=(65, 15),
      tolerance=0.001, learning_rate=0.4,
      maximum_epochs=3000, adapt_learn-
      ing_rate=False, annealing_value=2000,
      display_graph_windows=False, display_
      graph_pdf=True, data_file_has_titles=True,
      data_file_has_brief_titles=True)
```

Example for using trained ANN for prediction

If you have data in csv file format (say its name is QueryN.csv), to start a Study that involves normalizing data, retrieving the ANN, and predict the output features, so please consider adding the following code at the end of the program, or you can import the program as is, and add this code after the import statement: either to simple query:

```
Study('QueryN.csv', 'query', previous_study_
      data_file='NrChNet.nsr')
```

Or, alternatively with advanced query:

```
Study(purpose= "advanced query", Previous_study_
      data_file= 'NrChNet.nsr',
      start_time=time.time(),input_parameters_
      suggested_values= ([10, 45, 5], 7.5, (0.5,
      0.8), ('A', 'C')))
```

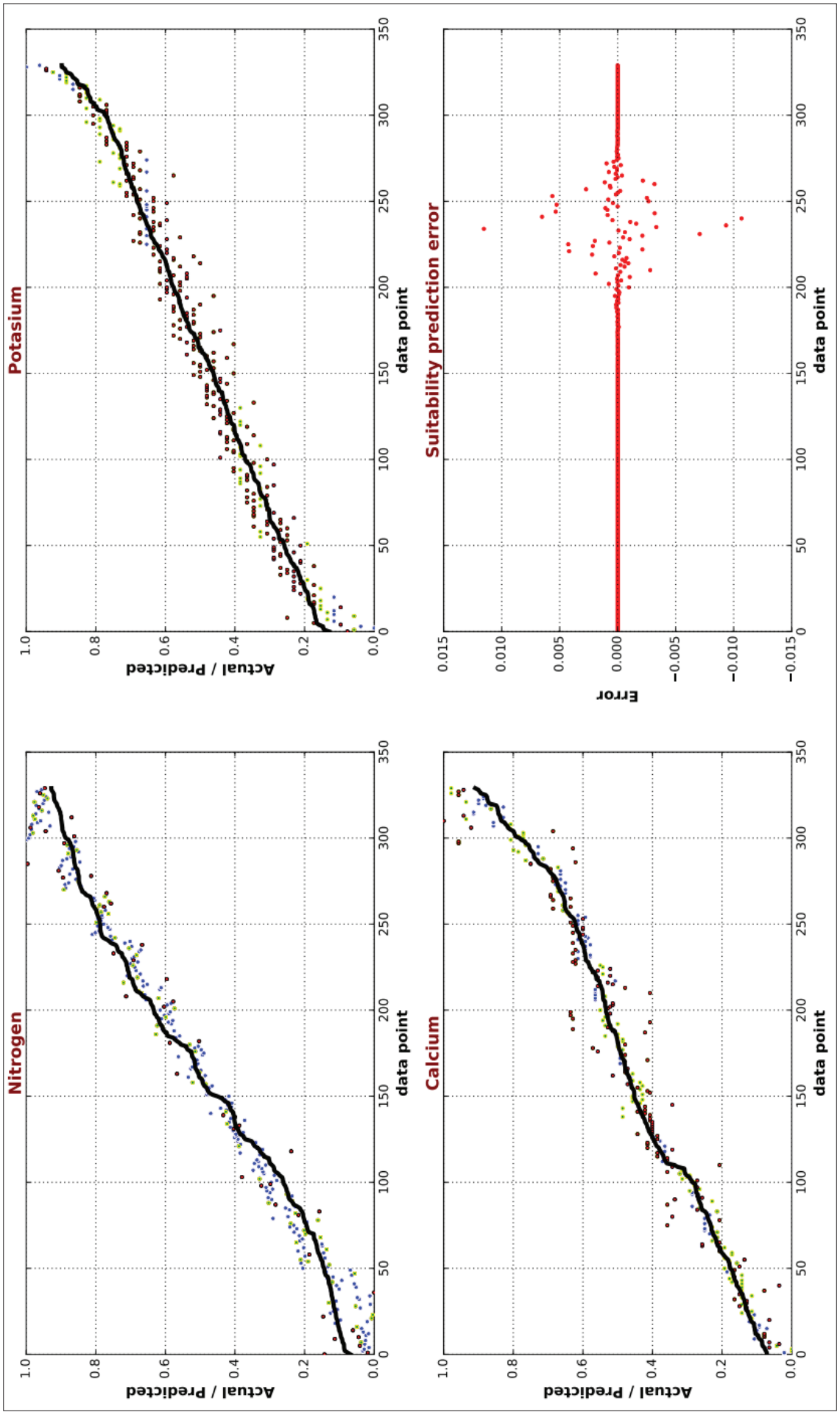

Figure 6: Prediction function and data cloud.

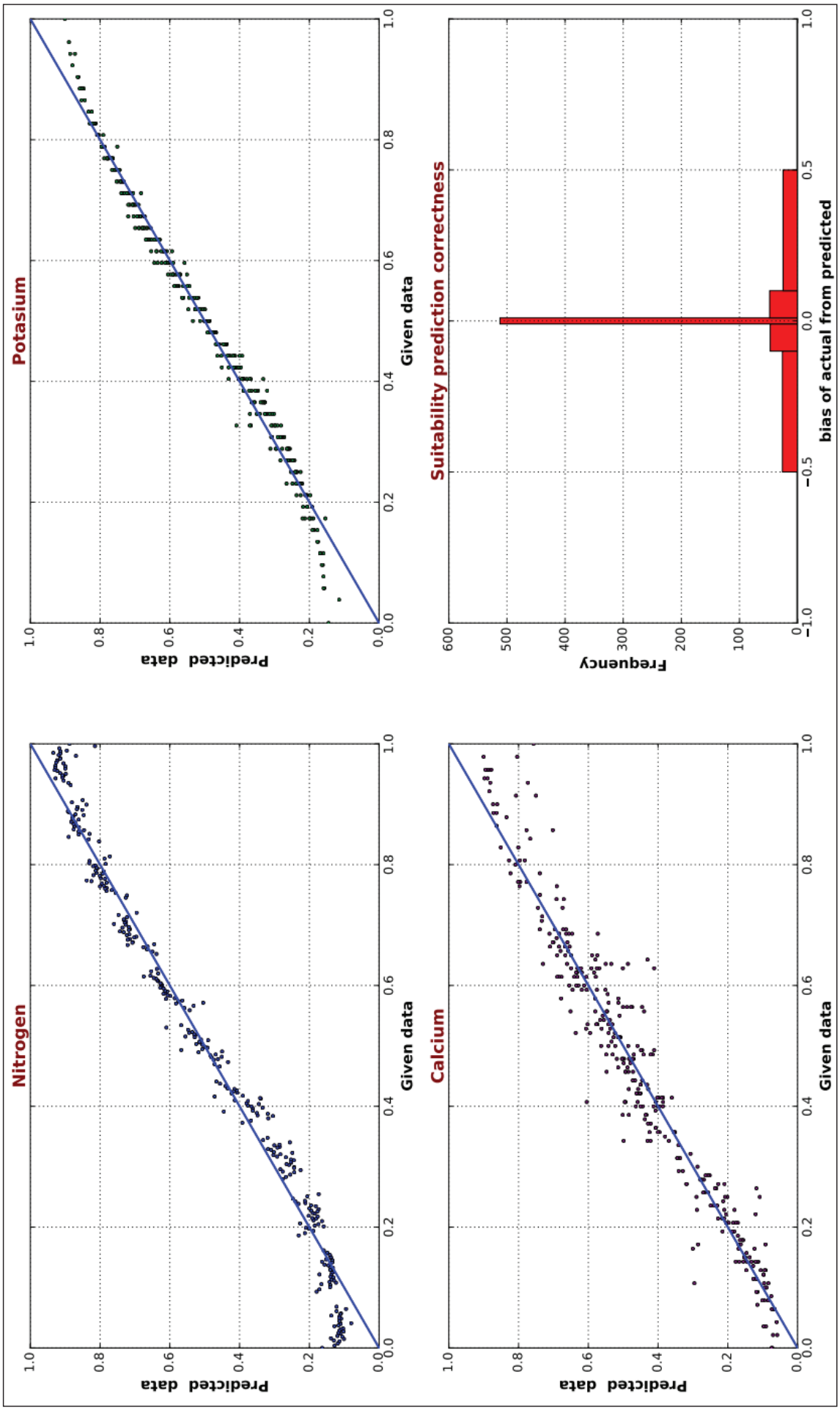

Figure 7: Given vs. predicted data on 45-degree line.

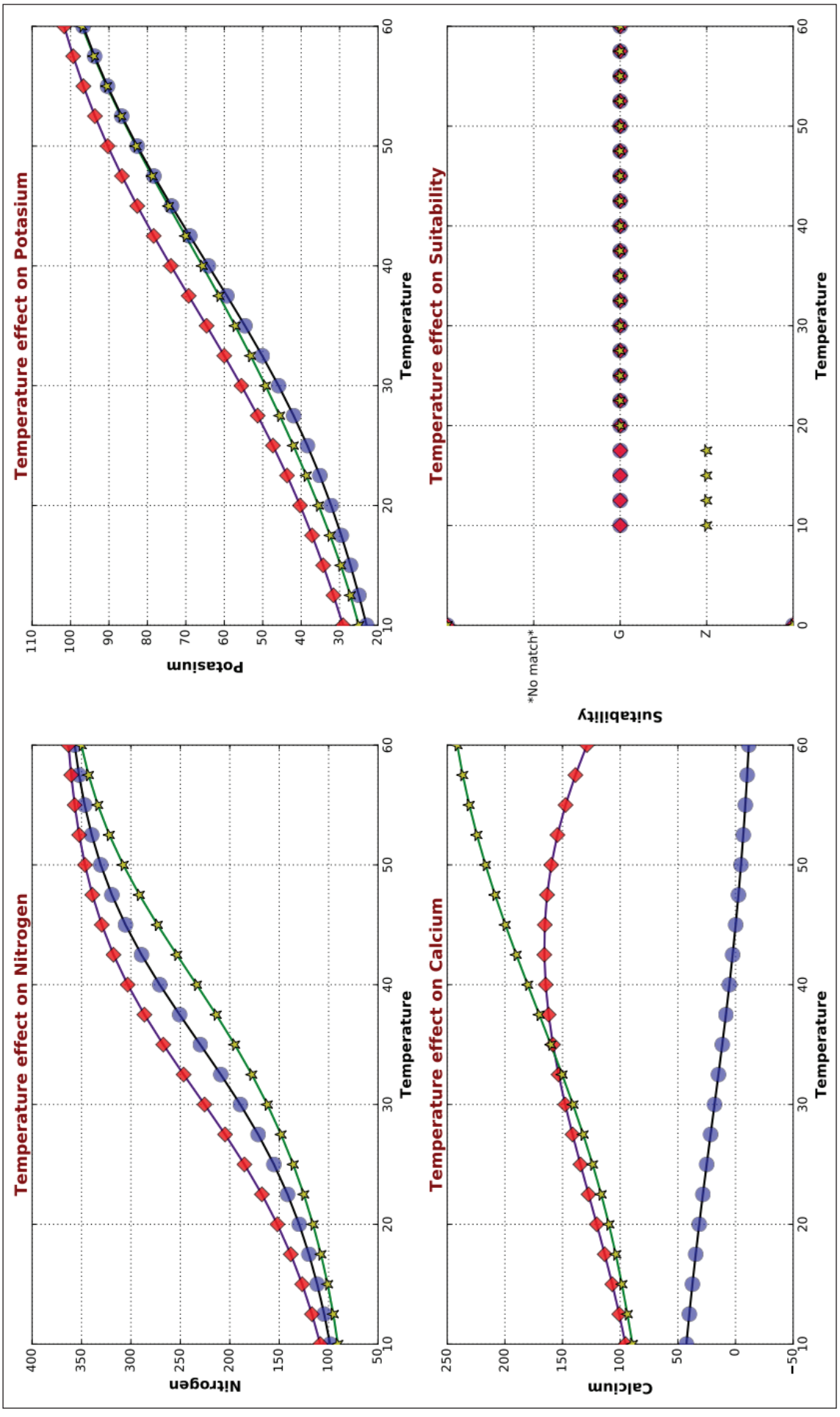

Figure 8: Sample relational charts of numeric input feature.

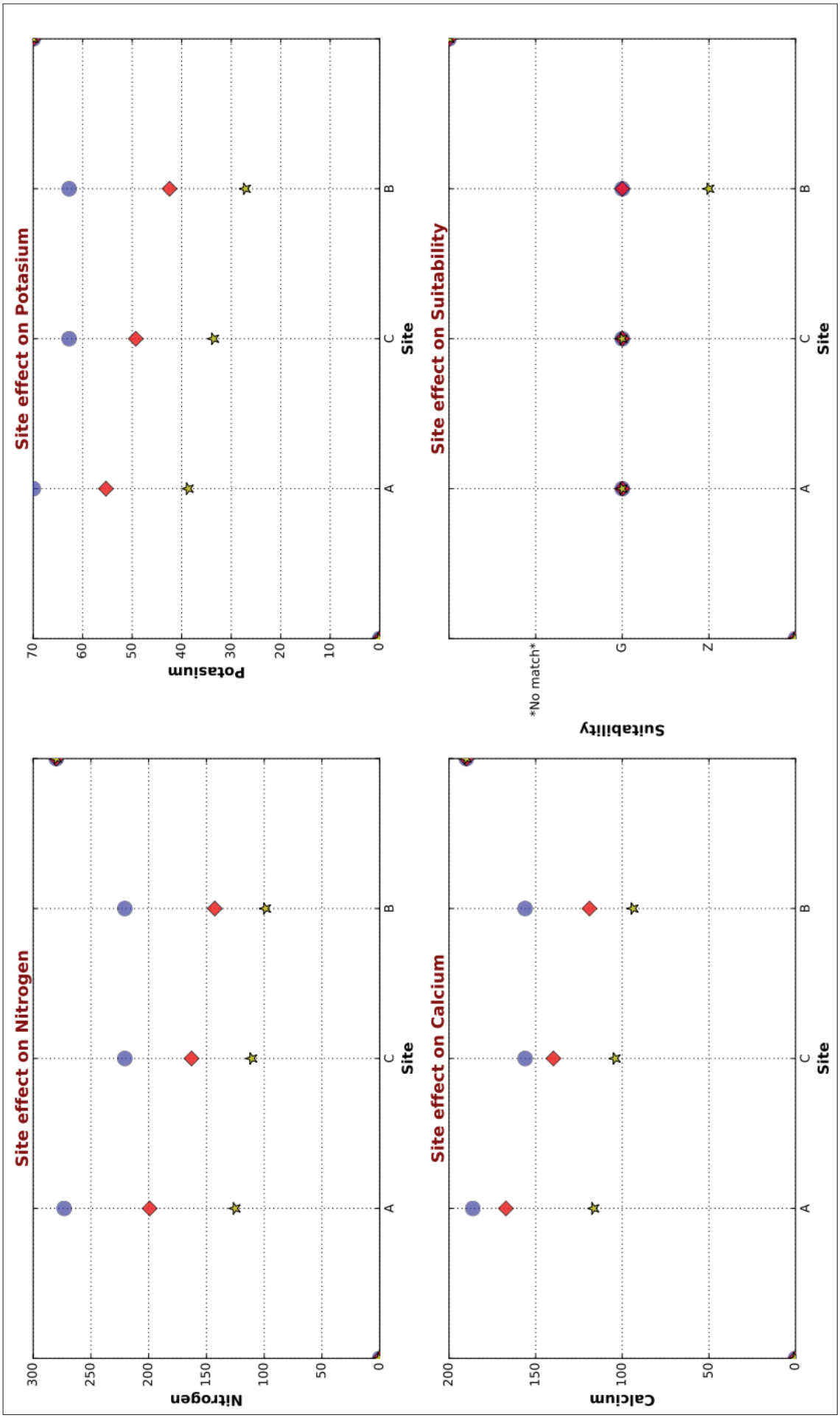

Figure 9: Sample relational charts of categorical input feature.

Validation of results

The results of the program were validated by comparing to the results of Statistica v13 academic edition running the same problems. All of the tested cases match very well with the results of NeuroCharter. However, Statistica deals only with one type of outputs at a time (either numeric or categorical through the regression and classification modules respectively), while the proposed software can deal with both types at the same time. The results of Statistica show faster performance and a small increase in the accuracy measures like the correlation coefficient, which may be attributed to that they use more advanced algorithms in training than that we employed in this software. However, the sensitivity analysis of the variables and the predictions match very well with our software.

(2) Availability

Operating system

Windows, (Tested only on Windows 10 64bits).

Programming language

The program was programmed in Python 2.7.12. It is not suitable to run with Python 3.* without modifications.

Additional system requirements

No special requirements of hardware, the faster the better.

Dependencies

The program requires Microsoft Windows OS and Adobe Acrobat to be installed. Additionally, as the program has no graphical user interface in its current version, it requires Jet Brains PyCharm and Anaconda (for python 2.7.*) to be preinstalled as well. All the dependencies and the requirements installation steps are listed in the attached tutorial (https://goo.gl/TGop2p). The program requires some of the built-in libraries (`collections`, `copy`, `CSV`, `datetime`, `math`, `itertools`, `random`, `shelve`, `subprocess`, and `time`). Additionally, it requires the following external libraries to be installed: `matplotlib 1.5.1`, `prettytable 0.72`, `reportlab 2.7`, `scipy 0.19`, and `numpy 1.11.0`. (Please refer to the installation tutorial for details.)

List of contributors

Authors only.

Software location

Archive: FigShare

Name: NeuroCharter, software for artificial neural networks

Persistent identifier: DOI: 10.6084/m9.figshare.5293453

License: MIT

Publisher: Mohammad Elnesr

Version published: 1.0.79

Date published: 08-08-17

Code repository: GitHub

Name: NeuroCharter

Identifier: https://github.com/drnesr/NeuroCharter

Licence: MIT

Date published: 08/08/17

Language

English.

(3) Reuse potential

NeuroCharter can be used to perform regression analysis for systems in many fields including but not limited to environmental systems, bio systems, soil and water systems.

If a researcher has any system of independent inputs, and their corresponding outputs, this software can normalize the data, build an artificial neural network, train it, and save it for future use in querying regression data. The researcher can benefit from the publication-quality graphs the program provides, in addition to the detailed reports about the trained network. Through the saved network, the program can easily predict the output features of any query dataset. The program is free-to-use according to the MIT license agreement. Contributions to add more features to NeuroCharter are most welcomed. Technical support to the software is provided in a limited scope, by direct email to the corresponding author, or to the GIT repository.

Important: To reuse this software easily, please follow the instructions at the tutorial file mentioned above.

Competing Interests

The authors have no competing interests to declare.

References

1. **Leon, M A** and **Keller, J** 1997 Toward implementation of artificial neural networks that "really work". *Proc AMIA Annu Fall Symp.*: 183–7.
2. **Ragsdale, C T** and **Zobel, C W** 2010 (Jan 1) A Simple Approach to Implementing and Training Neural Networks in Excel. *Decis Sci J Innov Educ.*, 8(1): 143–9. DOI: https://doi.org/10.1111/j.1540-4609.2009.00249.x
3. **Werbos, P J** 1975 Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University (Dept. of Defense contract).
4. **Bourzac, K** 2016 Bringing Big Neural Networks to Self-Driving Cars, Smartphones, and Drones (Internet). IEEE Spectrum: Technology, Engineering, and Science News. Available from: http://spectrum.ieee.org/computing/embedded-systems/bringing-big-neural-networks-to-selfdriving-cars-smartphones-and-drones (cited 2016 Jun 2).
5. **Moustris, K P, Nastos, P T, Larissi, I K, Paliatsos, A G, Moustris, K P, Nastos, P T,** et al. 2012 (Jul 18) Application of Multiple Linear Regression Models and Artificial Neural Networks on the Surface Ozone Forecast in the Greater Athens Area, Greece, Application of Multiple Linear Regression Models and Artificial Neural Networks on the Surface Ozone Forecast in the Greater Athens Area, Greece. *Adv Meteorol Adv Meteorol.*: e894714.
6. **Jaimes, F, Farbiarz, J, Alvarez, D** and **Martínez, C** 2005 Comparison between logistic regression and neural networks to predict death in patients with sus-

pected sepsis in the emergency room. *Crit Care*, 9(2): R150–6. DOI: https://doi.org/10.1186/cc3054
7. **Izadifar, Z** and **Elshorbagy, A** 2010 (Nov 15) Prediction of hourly actual evapotranspiration using neural networks, genetic programming, and statistical models. *Hydrol Process*, 24(23): 3413–25. DOI: https://doi.org/10.1002/hyp.7771
8. **McCaffrey, B J** 2013 (Jul 22) Neural Network Data Normalization and Encoding. *Visual Studio Magazine* (Internet), 2013(7). Available from: https://visualstudiomagazine.com/articles/2013/07/01/neural-network-data-normalization-and-encoding.aspx (cited 2016 Jun 5).
9. **Orr, G B** and **Müller, K-R** 2003 Neural networks: tricks of the trade (Internet). Springer. Available from: http://j.mp/1TPUfhV (cited 2016 Jun 5).
10. **Bengio, Y** 2012 (Jun 24) Practical recommendations for gradient-based training of deep architectures. ArXiv12065533 Cs (Internet). Available from: http://arxiv.org/abs/1206.5533 (cited 2016 Jun 5).
11. **Bishop, C M** 1995 Neural networks for pattern recognition (Internet). 1st ed. Oxford university press (Advanced Texts in Econometrics; vol. 1). Available from: https://books.google.com/books?hl=en&lr=&id=T0S0BgAAQBAJ&oi=fnd&pg=PP1&dq=Bishop,+C.M.,+1995a.+Neural+Networks+for+Pattern+Recognition&ots=jL8YrE8Cne&sig=9tUiUpDH0HkEERhWj0DedEQbKTI (cited 2016 Jun 5).
12. **Leverington, D** 2009 A Basic Introduction to Feedforward Backpropagation Neural Networks (Internet). Texas Tech University. Available from: http://www.webpages.ttu.edu/dleverin/neural_network/neural_networks.html (cited 2016 Jun 5).
13. **Milo, B O** and **Denis, F** 2015 python - How to visualize a neural network (Internet). Stack Overflow. Available from: http://j.mp/219xGJ9 (cited 2016 Jun 9).

How to cite this article: Elnesr, M N and Alazba, A A 2017 NeuroCharter: A Neural Networks Software to Visually Discover the Effects and Contributions between Interrelated Features. *Journal of Open Research Software,* 5: 19, DOI: https://doi.org/10.5334/jors.135

Submitted: 11 June 2016 **Aceepted:** 11 August 2017 **Published:** 04 September 2017

]u[*Journal of Open Research Software* is a peer-reviewed open access journal published by Ubiquity Press

OPEN ACCESS