**Hyperparameter Estimation for Neural Network Architectures in Agriculture-based Structured Data Sets for Yield Prediction**

**Computers and Electronics in Agriculture**

**https://www.sciencedirect.com/journal/computers-and-electronics-in-agriculture**

**Authors:** Thanos Gentimis, Yinan Yang, Phillip Lanza, Tri Setiyono, Maria Bampasidou

**Abstract:** In this paper we will be exploring the performance of various Neural Network architectures, for predicting yield on datasets coming from smart operations in Agriculture. We will analyze the accuracy, measured through the Mean Square Error on test sets and its behavior over the space of hyperparameters, describe a technique to improve performance by choosing ever increasing architectures and justify it mathematically. We will discuss a statistically driven choice of removing outliers for yield and discuss a spatial train test split for validation of the model. This is intended as an introductory methods paper to various statistical and mathematical aspects of Machine Learning techniques in agriculture.

**Keywords:** Hyperparameter tuning, Yield Threshold, Neural Network Architecture, Yield Prediction, Digital Agriculture, Repetition number, Semi-groups, Outlier Removal, Spatial Train Test Split

**1.Introduction:**

Agriculture, just like many other fields of scientific study, is entering an era of big data driven decision support algorithms. Although widespread adoption is still lagging, more and more researchers especially in industry are turning to Artificial Intelligence applications and the trend seems to be irreversible. One of the biproducts of the successful application of precision agriculture techniques these decades, was the creation of vast quantities of data, which although possibly actionable, have gone beyond the traditional understanding of scientistic in the area. Digital Agriculture, and specifically analytics based on Artificial Intelligence seems to be the answer to this problem of data over saturation. The main analytics tools in AI are Machine Learning based techniques with Neural Networks, Random Forests and Support Vector Machines being the top three go to options.

One of the most important questions in Machine Learning applications for practically all fields of study is the hyperparameter tuning, especially when neural networks based on GUI’s or pre-constructed Neural Networks of various open sources packages are employed. Commonly used Machine Learning techniques implemented in R or Python, come with pre-defined architectures and default hyperparameters, which in general don’t guarantee high prediction accuracies. Thus, the need for specific hyperparameter ranges is paramount, especially when the application of the ML prediction is more important than the theoretical exploration of it.

There is a plethora or scientific literature on the subject most of the time connected to specific architectures or datasets, with random search approaches (Bergstra 2012), automated “brute force” algorithms (Thornton 2013) and multiple heuristic and area specific approaches. The problem of Hyperparameter Optimization is indeed a well understood area of statistical optimization, that predates the current Machine Learning renaissance, with examples as early as 1990 (Feurer 2019).

Wu et al (Wu et al 2019) have connected the hyperparameter tuning with a Baysian Optimization approach. Their methodology does not need explicit expressions of the optimization function and it works very well for Random Forests and Convolutional Neural Networks, but it is not directly applicable to structured dataset, and it is more well suited to datasets where images are the primary input. In the paper “Hyperparameter tuning and performance assessment of statistical and machine-learning algorithms using spatial data” (Scratz et al 2019) the authors discuss the potential shortcomings of various hyperparameter tuning techniques on spatial data, like the ones involved in yield prediction and propose the idea of block cross-validation (Brenning 2005), or spatial cross validation (Ponjankukka 2017).

We thus understand that although lots of heuristic solutions, automated hyperparameter tuning software, packages, and functions exist, a systematic analysis of accuracies over the hyperparameter space is lacking. This is due to the natural variability of the underlying datasets, so endeavors like that fail in their generality. Furthermore, although finetuning exists in other fields employing machine learning, such results don’t exist as far as we know in the bibliography for structured data coming from agriculture, especially those that don’t employ images from drones or satellites and instead rely on geolocated datasets with spatial autocorrelations.

In this paper we give a mathematically rigorous systematic analysis on various aspects of the process, including a concrete methodology to identify optimal Neural Network architectures, a statistically validated outlier detection technique, a train-test split algorithm that takes into account the spatial correlations

The structure of the paper is as follows. After the introduction and motivation above, we give a thorough dataset description, and discuss the structure of similar datasets in section 2. In section 3, we explore theoretically the RMSE computation as a function of the various hyperparameters and establish all the relevant terminology, we discuss the corresponding hyperparameters. Our results are presented in section 4, followed by a discussion/conclusion section and the corresponding bibliography. Various graphs, summary statistics and others are added in the document, or delegated to the appendix based on importance.

**2. Dataset Description:**

This section details the provided data variables. Each variable is described, and their data type is given. Variables are described in the order in which they appear in the data files and is courtesy of Ag Analytics.

Create a table with all variables here, split it into varying and fixed (2 columns for varying one for fixed)

**2.1 Varying Variables in Dataset**

The fields x and y are used to hold the geographic coordinates of a given record. X holds the data point’s longitude coordinate and Y holds the point’s latitude coordinate. Data are in the form us unprojected, decimal coordinates using the WGS-84 coordinate reference system (CRS). Data are numeric with 8 decimals of precision. This translates to an approximate location precision of 1.11 x 10 ^ -6 m (which is beyond the precision of the RTK-enabled GNSS receiver, so location accuracy, which is more likely accurate to +/- 4 cm).

VRYieldVol is used to store volumetric yield. This is the value corresponding to the yield volume recorded by the harvester’s yield monitor at a given data point. Data are numeric with up to 6 decimals of precision. Units are user-defined and are usually either kg/ha, lbs/ac, or bushels/ac. The data value’s unit is stored in metadata file that is associated with the original harvest data file in John Deere’s data store. It may be possible to retrieve these by accessing the source data in Ag Analytics’ cloud database.

Row and Col store the row and column of the data point’s location within Ag Analytics’ 10m x 10m data point sample grid for a given field. Row and col identifiers are 0-indexed (i.e., column numbering starts with 0 instead of 1). These grid points are aggregations of data falling within the 10m x 10m area represented by a given row and column. Values are calculated from raw data using proprietary algorithms. Data are of the integer data type and should be treated as being categorical in most cases.

Relative\_Elevation1 is a decimal value representing the standardized elevation value (z-score) of a given record relative to the mean elevation of the field. This is useful in determining areas of higher and lower elevation present within the field. Relative elevation can affect the water and nutrient status of a given area due to how water and nutrients flow to and from a given area due to elevation differences. Values are positive or negative numeric with 9 decimals of precision.

Slope1 holds the maximum slope value present in the 10m x 10m cell represented by a given data point. Values are positive numeric values having 9 decimals of precision. Slope can affect water and nutrient status of a given point due to how water and nutrients flow over that area due to the degree of slope.

TR1 holds the terrain ruggedness index value. This represents the amount of hilliness and slope amount present within a given cell. This value is calculated by a proprietary algorithm. Values are positive numeric with 9 decimals of precision. Terrain ruggedness can affect water and nutrient status of an area by affecting how water and nutrients flow and percolate through a given area.

Elevation1 holds the absolute elevation value of a given point. This is generally meters above sea level (ASL) but is user-configurable on the GNSS sensor. Values are numeric with 8 decimal points of precision.

Application\_<#>\_<N\_Rate, ID, and date> – these fields hold the application rates of nitrogen (N) applications for each record, the application operations associated internal ids and their application dates. There are 10 fields for each of these for a total of 30 total fields. Nitrogen application rate, in conjunction with the percentage of N applied is very strongly correlated to plant growth and yield in most grain crops (soybeans and other leguminous crops being the exception). The N% values can be retrieved from the AgAnalytics database using the associated ID record values from the ApplicationOperations table. N\_Rate values are numeric 8 decimals of precision. Units are user-defined and are generally gallons/ac, lbs/ac, or kg/ha. Application\_ID records are categorical alphanumeric text values and correspond to AnAnalytics’s internal nitrogen application operation record ids present in their database. Application\_Date records are integer values corresponding to the day of the year in which the nitrogen was applied. For example, a value of 2 corresponds to January 2nd of a given year and 365 (or 366 if a leap year) corresponds to 12/31 of a given year. These values can be compared with the PlantingDay1 values (covered later) to determine how many days after planting (DAP) each N application was applied.

Ph\_mean\_30\_60 stores the mean soil pH value present between 30cm and 60 cm for a given 10m x 10m cell and are numeric with 9 decimals of precision. pH is the -log of hydrogen ion activity present in a sample. Values below 7 are acidic and values above 7 are alkaline. Minimum pH is usually 0, with only extremely acidic compounds having pH values below 0. Maximum pH is usually 14, with only extremely alkaline compounds having pH values exceeding 14. Soil pH is generally between 4 and 10, except in extreme cases. Soil pH affects nutrient availability to plants. Optimum ranges are between 5.5 and 6.5 for most agronomic crops. Ranges outside of these values can have nutrient deficiencies or toxicities.

Clay\_mean\_30, 60, silt\_mean\_30\_60, and sand\_mean\_30\_60 denote the percentages of clay, silt, and sand, respectively, present in the soil between 30cm and 60cm for a given 10m x 10m cell record. The relative percentages of each affect the soil’s texture, nutrient holding capacity, and other chemical and physical soil properties.

Ksat\_mean\_30\_60 denotes the mean saturated hydraulic conductivity of soil between 30cm and 60cm for a given 10m x 10m cell record. These are numeric values having 9 decimals of precision. This value indicates how easily water can percolate through soil once the soil is fully saturated. Higher values indicate greater flow rates, meaning that the soil allows water to flow through it more freely than areas of soil having lower ksat values. Soil’s hydraulic conductivity can influence water and nutrient availability to plants by influencing how long water and nutrients are present in each area and how quickly mobile nutrients can leach away. Soils having higher percentages of clay generally have lower ksat values whereas soils having higher percentages of sand generally have higher ksat values.

Om\_Mean\_30\_60 holds the mean organic matter (OM) percentage of the soil for a given 10m x 10m cell record between 30cm and 60cm. Values are numeric with 9 decimal places of precision. OM generally improves soil texture and soil nutrient and water holding capacity, and nutrient availability to plants. OM can also slowly decompose (mineralize) to release nutrients to the soil solution.

The rest of the variables in the dataset are common for all our datapoints. A complete description of them can be found on Table A in the appendix.

**2.2 Structured data set Characteristics**

The dataset described above is a typical example of an agronomic dataset that is tied to smart devices on a field. Multiple geolocated layers are gathered and saved in raster files, which then can be turned into point-based datasets like the one we describe here. While collapsing spatial information to the centers of various polygons leads to some loss of information, this is the best approach available, and it is the standard when predicting yield outputs is the end goal.

This average over polygon approach is also essential since various agronomic practices are done on different levels, by different machines therefore some form of standardization is needed for all input to be tied to the corresponding output.

We note that the common variables identified above can be safely removed from our predictive model without loss of accuracy. Still, a more general model, for example one that incorporates different crops, locations, years etc., will need these extra variables. It is not unheard of, for machine learning models that give accurate results even when applied to different crops, provided that the crop type is an input variable and multiple training datasets are available. Similarly, we have observed that by introducing multiple years of structured data, our predictive capabilities increase even though the environmental conditions may change drastically.

**3. Methods:**

**3.1 Description of the MSE function**

We will be exploring the MSE function where E is the number of epochs, b is the batch size, OT is an optimization technique, LF is a loss function, NNA is the specific Neural Network architecture and X is the dataset used.

* Add a description of the hyperparameter space, exploring its “shape”.
* Compute the distribution of the expected value of f given some hyperparameters.

Definition of RMSE with the formula (with a train test split idea)

Root means square error is the standard deviation of the residuals (prediction errors). RMSE is one of the most used measures for evaluating the quality of predictions. It shows how far predictions fall from measured true values using the Euclidean distance. To compute RMSE, calculate the square of the residual (difference between prediction and truth) for each data point in the training set, then calculate their and take the square root of the result. RMSE is commonly used in supervised learning applications, as RMSE uses and needs true measurements at each predicted data point

Where is the number of data points in the testing set, is the true i-th measurement, and is its corresponding prediction. In our train-test split procedure, we use Train dataset x\_train to fit the model, and then we use this model along with x\_test to predict the test dataset y\_test and get prediction of y\_test. So, in our case, is y\_test and is prediction of y\_test calculated from the model.

It is obvious that a key component that drives the variability of this function is X, i.e., the specific dataset being used, including its subdivision in train-test data. We will assume that the datasets we are analyzing come for a certain population, whose variables fall on similar, although probably unknown distributions.

**3.2 Description of hyperparameters**

Our implementation and subsequent analysis were done using the package Keras (Chollet 2015), which is a package written in Python, building on the very popular Machine Learning library TensorFlow. Keras was chosen among other options since it offers simple APIs, minimizing the number of user actions required and streamlining the process of fine-tuning traditional NN’s. Keras has an extensive support base that has created detailed documentation and guides.

**Batch[[1]](#footnote-1)**: a set of N samples. The samples in a batch are processed independently, in parallel. If training, a batch results in only one update to the model. A batch generally approximates the distribution of the input data better than a single input. The larger the batch, the better the approximation; however, it is also true that the batch will take longer to process and will still result in only one update. For inference (evaluate/predict), it is recommended to pick a batch size that is as large as you can afford without going out of memory (since larger batches will usually result in faster evaluation/prediction).

**Epoch[[2]](#footnote-2)**: an arbitrary cutoff, generally defined as "one pass over the entire dataset", used to separate training into distinct phases, which is useful for logging and periodic evaluation. When using validation data or validation split with the fit method of Keras models, evaluation will be run at the end of every epoch. Within Keras, there is the ability to add callbacks specifically designed to be run at the end of an epoch. Examples of these are learning rate changes and model checkpointing (saving).

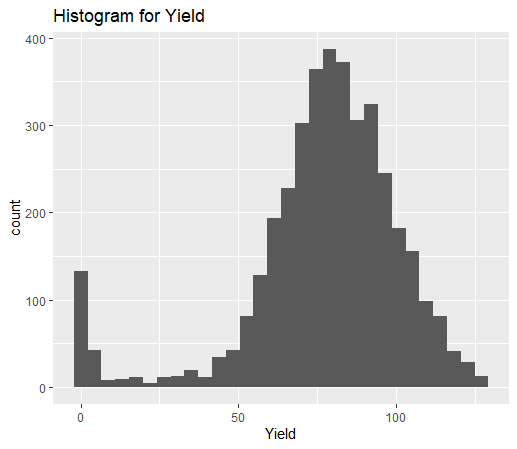
The batch size is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. Think of a batch as a for-loop iterating over one or more samples and making predictions. At the end of the batch, the predictions are compared to the expected output variables and an error is calculated. From this error, the update algorithm is used to improve the model, e.g., move down along the error gradient.

The number of epochs is a hyperparameter of gradient descent that controls the number of complete passes through the training dataset. One epoch means that each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch is comprised of one or more batches. For example, an epoch that has one batch is called the batch gradient descent learning algorithm. We can think of a for-loop over the number of epochs where each loop proceeds over the training dataset. Within this for-loop is another nested for-loop that iterates over each batch of samples, where one batch has the specified “batch size” number of samples.

**3.3 Outlier removal**

The key output variable for these types of datasets is the Yield, which is typically measured with yield monitors, representing fixed areas on the field. A typical histogram of the yield values reveals that the measurements of yield follow a normal distribution, but there are many zero or low values, that are coming from edge effects, uncovered areas (roads, draining ditches etc.) and other field anomalies.

In the following graph we see the unfiltered histogram for yield in our dataset and note the behavior described above.



It is customary thus in yield prediction, to remove the zeros of the field and try to predict the rest of the values in the dataset. This is an appropriate technique, but it fails to account for “problematic” almost zero values. One statistically valid method to reduce the number of outliers is to introduce a lower value threshold incrementally, until the resulting set of yields is normally distributed as follows:

Consider the set of all yield values Y. Choose an increment step a>0. A sensible choice of the step can be 0.5% of the maximum yield in the dataset.

* Create the subset
* Test the subset with a normality test, for example the Kolmogorov-Smirnov Test and compute the corresponding p-value,
* If is large enough (i.e., the subset can be thought of as following the normal distribution) we stop. Otherwise replace a with k\*a, k=1,2,3… until the process terminates. Theoretically a p value greater than 0.05 would suffice.

It is true that the normality tests do not accept the null and no “large enough” p-value will guarantee that the data is normally distributed. Thus, the following heuristic improvement of the method above can be used:

Follow the method above for all multiples i\*a, i=1,2, 3, …, m until . Then choose the first multiple k\*a, for which or in other words look for the first time that there is a significant change in the p-value. It is not unusual with this technique to observe a change of an order of magnitude in the p-value, from one iteration to the next. We believe that this happens when almost all “problematic entries” are removed from the dataset.

**3.4 Spatial Train Test split**

To account for the spatial component introduced by field data we implement the following spatially based algorithm in QGIS for the train test split processes:

* First step is reprojecting data. Our dataset was in an unprojected format, we need to reproject them to a format that supports accurate distance measurements. LA is in UTM15N, so generally, this will be EPSG:32615 – UTM15N for any data in LA state. Other states should be treated similarly.
* The second step is segmenting the data points. We use the *Measure Line* tool to measure the approximate height and width the area encompassing our points. We then divide the length by the desired number of horizontal cells and divide the height by the desired number of vertical cells. This will give us approximate cell dimensions to use when generating our grid in the next step.
* The last step is creating grid. QGIS has a built-in tool for creating grid. We choose Rectangle(polygon) grid type first and then enter the horizontal cell width value and vertical height value that we calculated from the previous step. This will generate a grid that covers the field area points. It will be larger than the data point area. We use the move and rotate from the QGIS Advanced Digitizing to align the grid with the data points. Repeat the steps using different grid horizontal and vertical spacing values until the cells properly align on the data points. Once this is achieved, we delete the extra cells.
* All the data points in same grid have same grid ID, which becomes another variable in our dataset.

As an example, in figure () we have 20 different gird id’s (rectangles) with similar areas.

A picture containing pattern, colorfulness, art, ground

Description automatically generated

Assume now that our model will use an 80-20 train test split, or in other words, 80% of the data will be used to train the model and 20% will be used for testing. Our algorithm will use 80% of the rectangles created above at random for the training and the remaining 20% for the testing. This is implemented in Python through the gridID variable instead of the standard train-test split tool in scikit-learn.

**3.5 Neural Network Incremental Approach and Mathematical Justification**

We propose a novel algorithm for the determination of the optimal architecture and hyperparameters for a Neural Network (NN) model in agricultural yield prediction. The algorithm is implemented in Python, leveraging the high-level neural networks API, Keras, for model development and training. Our algorithm encompasses two key phases: architecture determination and hyperparameter tuning.

**Architecture Determination**: The objective is to construct an optimal neural network structure that offers the lowest Mean Squared Error (MSE) in yield predictions. The steps of the algorithm are as follows:

* **Sequential Model Initialization:** Our algorithm initiates with the creation of an empty Sequential model in Keras, a flexible tool for building a linear stack of layers in the neural network.
* **Layer Addition and Neuron Count Optimization:** We add an initial Dense layer to the model and assign a starting value for 'k', representing the total number of neurons. The model is then trained on our dataset, following which the performance is evaluated by calculating the MSE between the predicted and actual yield. Subsequently, the 'k' value is iteratively increased, with the model being retrained and the MSE recalculated at each step.
* **Statistical Analysis of Neuron Count:** A regression analysis is conducted on the MSE and corresponding 'k' values. If the regression coefficient for 'k' is statistically insignificant, the process of incrementing 'k' is halted. The current 'k' is then deemed optimal for that layer.
* **Iterative Layer Addition:** The previous steps are repeated with the addition of a new layer to the model each time. The process continues until the point where the inclusion of another layer does not provide any further improvement in the MSE. This approach ensures the model is neither underfit nor overfit.

The process described above is based on the idea that dense feedforward Neural Networks with fixed hyperparameters form a semi-group [Howie 95]. Namely, let be the collection of all dense feedforward Neural networks with a fixed set of hyperparameters, the same input layer (l) and one node in the output layer. A specific architecture can be described as a finite sequence , where is the number of nodes in layer i, with being the number of input nodes. Suppose also that B with and without loss of generality let’s assume that then we can define the addition:

This operation is associative and commutative. One can endow this structure with a “zero” element to be the architecture with only the input layer and the output layer of 1. Thus is a monoid [Jacobson 51]. Since (X, +) is a semigroup, one can define a partial order on X as follows: Let A, B in X, then if and only if there exists C in X such that . Clearly this is only a partial ordering since for example, A=(l,2,1,1) and B=(l,1,2,1) are not comparable. Still for any two architectures A, B there is always a unique minimal architecture that is greater than both, namely the architecture C=A+B. Of course, the mathematical description above cannot be directly converted to an optimization method for identifying the optimal architecture.

Still, the incremental approach presented above has the benefit of respecting the mathematical ordering since given an architecture , a one node increment in layer i, is equivalent to the addition with the architecture , where B has the same number of layers as A resulting in the element A+B which is greater than A. Similarly, the addition of a new layer with one node is equivalent to the addition with the architecture B= with one more layer than A which leads to A+B=, which is thus greater than A. Unfortunately, more complex architectures don’t necessarily guarantee higher accuracies (i.e., lower MSE’s) but it has been observed that deeper neural networks tend to outperform simpler ones due to their ability to uncover more complicated patterns. Thus, a statistical analysis of the MSE’s based on the complexity (total number of nodes) is added with a statistical criterion, to account for the intractable changes in the average MSE over various iterations.

**3.6 Hyperparameter Tuning:**

With the optimal architecture determined, the second phase involves tuning the hyperparameters - specifically, the number of training epochs and the batch size.

a. **Epoch and Batch Size Variation:** We systematically vary the number of epochs and batch size across a defined range. The model is trained with each combination, and the performance is evaluated using the MSE.

b. **Optimal Hyperparameter Selection:** The combination of epochs and batch size that delivers the lowest MSE is chosen as the optimal set of hyperparameters for our model.

To further enhance the algorithm's performance, we incorporate several advanced techniques:

**Early Stopping:** To prevent overfitting and unnecessary computation, we employ an early stopping mechanism during the training phase. If the model's performance on a validation set doesn't improve after a certain number of epochs, the training is halted.

**(todo)Learning Rate Scheduling:** We use a dynamic learning rate that adjusts over time, ensuring faster convergence during initial training phases and finer tuning in the later stages.

**(todo)Dropout Regularization:** To prevent overfitting and improve generalization, we include dropout layers in our network, which randomly drop out neurons during training.

**(todo)Batch Normalization:** We include batch normalization layers to accelerate training, reduce the chance of getting stuck in poor local minima, and provide a form of regularization.

This iterative and systematic algorithm, enriched with advanced techniques, offers a comprehensive approach to create an optimized neural network model for yield prediction using structured agricultural datasets.

**4. Results:**

**5. Discussion/ Conclusions:**

**6. Bibliography:**

Bergstra, James, and Yoshua Bengio. "Random search for hyper-parameter optimization." *Journal of machine learning research* 13.2 (2012).

Brenning, A.: Spatial prediction models for landslide hazards: review, comparison and evaluation, Nat. Hazards Earth Syst. Sci., 5, 853–862, https://doi.org/10.5194/nhess-5-853-2005, 2005.

Chollet, F., & others. (2015). Keras. https://keras.io. https://keras.io

Crane-Droesch, A., “Machine learning methods for crop yield prediction and climate change impact assessment in agriculture”, <i>Environmental Research Letters</i>, vol. 13, no. 11, 2018. doi:10.1088/1748-9326/aae159.

Feurer, M., Hutter, F. (2019). Hyperparameter Optimization. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds) Automated Machine Learning. The Springer Series on Challenges in Machine Learning. Springer, Cham. <https://doi.org/10.1007/978-3-030-05318-5_1>

Howie John M, Fundamentals of semigroup theory. London Mathematical Society Monographs. New Series. 12. Oxford: Clarendon Press. x, 351 p. (1995), Chapter 1

Jacobson Nathan, Lectures in Abstract Algebra, volI, D.Van Nostrand Company, ISBN 0-387-90122-1

Jonne Pohjankukka, Tapio Pahikkala, Paavo Nevalainen & Jukka Heikkonen (2017) Estimating the prediction performance of spatial models via spatial k-fold cross validation, International Journal of Geographical Information Science, 31:10, 2001-2019, DOI: [10.1080/13658816.2017.1346255](https://doi.org/10.1080/13658816.2017.1346255)

Liakos KG, Busato P, Moshou D, Pearson S, Bochtis D. Machine Learning in Agriculture: A Review. *Sensors (Basel)*. 2018;18(8):2674. Published 2018 Aug 14. doi:10.3390/s18082674

Patrick Schratz, Jannes Muenchow, Eugenia Iturritxa, Jakob Richter, Alexander Brenning,Hyperparameter tuning and performance assessment of statistical and machine-learning algorithms using spatial data, Ecological Modelling, Volume 406, 2019, Pages 109-120, ISSN 0304-3800,https://doi.org/10.1016/j.ecolmodel.2019.06.002.

https://towardsdatascience.com/simple-guide-to-hyperparameter-tuning-in-neural-networks-3fe03dad8594

Chris Thornton, Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2013. Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In <i>Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining</i> (<i>KDD '13</i>). Association for Computing Machinery, New York, NY, USA, 847–855. https://doi.org/10.1145/2487575.2487629

Jia Wu, Xiu-Yun Chen, Hao Zhang, Li-Dong Xiong, Hang Lei, Si-Hao Deng, Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization, Journal of Electronic Science and Technology,Volume 17, Issue 1,2019,Pages 26-40,ISSN 1674-862X, <https://doi.org/10.11989/JEST.1674-862X.80904120>. (https://www.sciencedirect.com/science/article/pii/S1674862X19300047)

Appendix/Others

The paper “Machine learning methods for crop yield prediction and climate change impact assessment in agriculture” by Crane-Droesch (Crane-Droesch 2018) proposes a semiparametric variant of a deep neural network, which can simultaneously account for complex nonlinear relationships in high-dimensional datasets, as well as known parametric structure and unobserved cross-sectional heterogeneity. During their analysis they showcase the superior predictive capabilities of such techniques over traditional models and discuss the importance of the hyperparameter tuning as a means of improving prediction even further.

In their review paper “Machine Learning in Agriculture” (Liakos 2018) the authors present an overview of the new applications of ML in yield prediction, disease and weed detection, crop quality, species recognition, livestock, water, and soil management. Based on their findings, ML with images for crop yield prediction, is much more prevalent than just the use of data records, hence most optimization techniques are currently focused on image-based datasets.

<http://users.cecs.anu.edu.au/~Tom.Gedeon/conf/ABCs2018/paper/ABCs2018_paper_96.pdf>

<https://mcsc.sc.mahidol.ac.th/publications/theses/chaloemrat_thesis.pdf>

Table : Common Variable Descriptions

|  |  |
| --- | --- |
| CropSeason | is generally the year in which the crop used to generate a given data file was grown and harvested. For example, 2008 corresponds to a crop grown in 2008. However, this can be any integer value and is not necessarily a year value. These data are of the integer data type and should be treated as categorical, label values in all analyses. |
| HarvestId | is an internal record identifier corresponding to AgAnalytics’s harvest record id used to retrieve the harvest data (i.e., VRYieldVol aka volumetric yield volume). These data are text and should be treated as categorical in all analyses. |
| Crop name | holds the name of the crop associated with a given record. These values are all text and are generally the common name of a given crop, such as sorghum or corn. These values should be treated as categorical in all analyses. |
| SeedingDensity | holds the seeding rate (# seeds/acre) planted at a given data point (10m x 10m cell). Values are positive integer values. Seeding density can affect plant yields and growth rates. Optimum seeding density varies from crop to crop and with environmental and agroclimatic conditions of a given growing region and season. |
| SeedingDensity\_missing | is a boolean value indicating whether seeding density data are missing. A value of 1 indicates that no seeding density data are available for a given point, while a value of 0 indicates that seeding density data are present. |
| Seeding\_Variety1 | is a categorical variable indicating the variety of a given species that was planted at a given data point. This value can be null, and null values may be indicated by nan in the data files. If a value is present, it will be an alphanumeric value. |
| GDD2 to GDD11 | these fields hold the growing degree days (GDD) accumulated at measurement period. These values are numeric with 3 decimals of precision. GDD’s are used to estimate the growth stage of a given crop across the growing season. GDD’s accumulate if the mean temperature is above a certain minimum, which varies from crop to crop. The equation for GDD calculation is GDD = mean temp – base temp if mean temp > base temp. The theory behind this is that crops will only develop while the temperature is above a certain minimum threshold. Crops generally reach a certain growth stage at specified ranges of GDD accumulation. |
| Precipitation 2 to Precipitation 11 | – these fields hold the cumulative precipitation amounts for a given measurement period for a given 10m x 10m cell record. Values are numeric with two decimals of precision. |
| Trait 1 to Trait 5 | - these fields indicate whether any genetically modified (GM) traits are present in the crop planted in each record, for example glyphosate resistance. These are text values and should be treated as categorical values in any analysis. |
| FIPSCode | indicates the federal information processing series (FIPS) unique identifier for the record’s geographic area. These values are integers but should be treated as categorical values in any analysis. |
| PlantingDay1 | indicates the day of year in which the crop was planted. These are integer values from 1 to 365 (or 366 in leap years). |
| HarvestDay | indicates the day of year in which the crop was harvested. These are integer values from 1 to 365 (or 366 in leap years). |

–

1. <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/> [↑](#footnote-ref-1)
2. <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/> [↑](#footnote-ref-2)