CROP RECOMMENDATION SYSTEM USING NEURAL NETWORKS

INNOVATIVE AND CREATIVE PROJECT

Submitted by

SNEAHA M	**15BCS076**
SUSILKUMAR M	**15BCS094**
VIGNESH A	**16BCS316**

in partial fulfillment for the award of the degree

of

Bachelor of Engineering

in

Computer Science and Engineering

Dr. Mahalingam College of Engineering and Technology

Pollachi - 642003

An Autonomous Institution

Affiliated to Anna University, Chennai - 600 025

NOVEMBER 2018

Dr. Mahalingam College of Engineering and Technology

Pollachi - 642003

An Autonomous Institution

Affiliated to Anna University, Chennai -600 025

BONAFIDE CERTIFICATE

Certified that this project report,
"CROP RECOMMENDATION SYSTEM USING NEURAL NETWORKS"
is the bonafide work of

SNEAHA M	15BCS076
SUSILKUMAR M	15BCS094
VIGNESH A	16BCS316

who carried out the project work under my supervision.

Mrs. J. Bhavithra
SUPERVISOR
Assistant Professor(SS)
Dept. of Computer Science and Engineering
Dr. Mahalingam College of Engineering and Technology, Pollachi – 642003

Dr. G.Anupriya
HEAD OF THE DEPARTMENT
Professor
Dept. of Computer Science and Engineering
Dr. Mahalingam College of Engineering and Technology, Pollachi – 642003

Submitted for the Autonomous End Semester Examination for Innovative and Creative Project

Viva-voce held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

SMART AGRICULTURE THROUGH NUTRITION-BALANCED RECOMMENDATION AND OPTIMIZATION

ABSTRACT

Recommender systems are tools for interacting with large and complex information spaces. They provide a personalized view of such spaces, prioritizing items likely to be of interest to the user. Recommendation system can be broadly classified into three categories: content-based, collaborative, and hybrid recommendation approaches. Existing system uses Deep neural networks (DNNs) which is a powerful machine learning models and have succeeded in various artificial intelligence tasks. Although various architectures and modules for the DNNs have been proposed, selecting and designing the appropriate network structure for a target problem is a challenging task. Also, Optimization is the basic factor needed to enhance recommender systems. The existing model lacks in providing Optimization. To overcome the disadvantages in the existing model, it is proposed to use neural network approach.

The proposed model will predict the output using the neural networks concept. A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. Neural networks can adapt to changing input so the network generates the best possible result without needing to redesign the output criteria. The conception of neural networks is swiftly gaining popularity in the area of trading system development. The accuracy will be increased using gradient descent algorithm. Comparing the error rates, it is identified that the average rate in the existing model is not up to the mark and the prediction will be partially satisfied. The proposed system is developed to optimize the error rate and accuracy. The developed model will predict not only the recommended crops but also the hidden patterns which will be efficient and with reduced error rate.

ACKNOWLEDGEMENT

First and foremost, we wish to express our deep unfathomable feeling, gratitude to our institution and our department for providing us a chance to fulfill our long cherished of becoming Computer Science engineers.

We express our sincere thanks to our honorable Secretary **Dr.C.Ramaswamy** for providing us with required amenities.

We sincerely thank our director student affairs **Dr.RangaPalaniswamy**, for his moral support and encouragement for our project.

We wish to express our hearty thanks to the Principal of our college **Dr.A.Rathinavelu,** for his constant motivation and continual encouragement regarding our project work.

We are grateful to **Dr.G.Anupriya,**Head of the Department, Computer Science and Engineering, for her direction delivered at all times required. We also thank her for her tireless and meticulous efforts in bringing out this project to its logical conclusion.

Our hearty thanks to our guide **Ms.J.Bhavithra** Assistant Professor(SS) for her constant support and guidance offered to us during the course of our project by being one among us and all the noble hearts that gave us immense encouragement towards the completion of our project.

We are deeply grateful to our project coordinator, **Ms.A.Brunda** Assistant Professor(SS) for her guidance, patience and support. We also thank our review panel members **Dr. K.Thirukumar** Associate Professor(SS), **Dr.M.Pandi** Assistant Professor(SG) and **Ms.V.Priya** Assistant Professor(SS) for their continuous support and guidance.

TABLE OF CONTENTS

List of Figures

List of Tables

List of Equations

1. NTRODUCTION

Crop Recommendation System is the tool using which an algorithm will be able to predict which crop pattern is suitable for their field. Neural Network one among the successful concept which provides large availability for the inputs to get manipulated and devised. The learning problem in neural networks is formulated in terms of the minimization of a loss function. This function is in general, composed of an error and a regularization terms. The error term evaluates how a neural network fits the data set. On the other hand, the regularization term is used to prevent over fitting, by controlling the effective complexity of the neural network.

The model will be given a set of crop patterns, and trained to find which pattern is suitable and acceptable to the field conditions. These crop patterns will have some underlying qualities, which has to be identified by the model in order to successfully analyze the desirable crop. These qualities may be of several types, ranging from nutrition factors such as copper, sulphur, magnesium, boron, manganese, zinc, cobalt and weather patterns, crop duration and soil quality, etc. Earlier research has pointed towards Multi Layer Perceptrons and artificial neural networks for solving this problem. Multi Layer Perceptrons MLP[1] a common type of artificial neural networks (ANNs), are widely used in computer science research for object recognition, discrimination and classification, and have more recently found use in process monitoring and control. Neural networks are also composite in the sense that multiple neural building blocks can be composed into a single (gigantic) differentiable function and trained end-to-end. The key advantage here is when dealing with content-based recommendation. It is inevitable when modeling users/items on the web, where multi-modal data is commonplace. From earlier research, it can be seen that the performance of neural networks and humans vary drastically. Some sequences that could not be solved by any neural networks have been solved by some human subjects, and vice versa. However, further research is required in this area, as only simple architectures have been tested on this problem.

Ragni and Klein investigated the use of simple neural networks using back propagation. These networks use hyperbolic tangent or linear activation functions which cause the vanishing/exploding gradient problem. Rectified linear units, on the other

hand, have become very popular recently, and have been known to improve the performance of neural networks by eliminating the vanishing/exploding gradient problem. They are used extensively in computer vision and speech recognition.

1.1 Overview

Agriculture is getting worse in its way now-a-days. But many of young students are taking initiatives to develop agriculture. Since the world is getting digitalized we should also keep on its flow. This was the main reason why this project is going to be developed. The major intuition of this project is to get closer to farmers via digitalized technologies and make their lands more profitable beyond usual yield. The durability is also going to be taken as constraint so that the factor of dry lands will be avoided and during that season rather than usual crops some other crops will be recommended based on the rotation. The study focuses on developing a network of clusters containing various attributes as criteria using neural networks concept. The users can definitely use it in an efficient manner as well as it will be their beneficiary factor. Some attributes can be added to the networks only in the form of clusters with the help of admin rights. The Farmers can also mark their presence in the market through this app and can evaluate and compare their product prices with this application and they can sell and earn through this application.

1.2 Objective of the Project

- To improve the accuracy for recommendation of crops using neural networks concept.
- To reduce the error rate by applying optimization using Gradient Descent Algorithm.

1.3 Chapter Wise Organization

Chapter 2 presents a brief overview of the existing literature on this topic. ANN approaches to form clusters for the attributes, architecture of Recurrent Neural Network (RNN), and optimization factors will be discussed. Chapter 3 describes the methodology used to solve the serendipity problem in recommendation system for

crops. This includes preprocessing the input data to a form more suitable for training neural networks, and ways to initialize weights effectively using unsupervised learning methods such as auto encoder. An overview of the experiments conducted and the hyper parameters used are also described. Chapter 4 presents the results of the experiments. The performance of the networks using various architectures and weight initialization methods is tabulated. Chapter 5 describes the interpretation of the results, and gives a formal conclusion to the project.

2. LITERATURE SURVEY

A brief summary of earlier research is outlined below.

2.1 Introduction

The idea of ANNs is based on the belief that working of human brain by making the right connections, can be imitated using silicon and wires as living neurons and dendrites.The human brain is composed of 86 billion nerve cells called neurons. They are connected to other thousand cells by Axons. Stimuli from external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network. A neuron can then send the message to other neuron to handle the issue or does not send it forward.ANNs are composed of multiple nodes, which imitate biological neurons of human brain. The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data. The result of these operations is passed to other neurons. The output at each node is called its activation or node value.

2.2 Neural Networks using Case-Initialized Genetic Algorithms

The design of neural networks for time series prediction consists of the three steps - identification, estimation and evaluation. Several Application uses Time series prediction to support decision making[7]. A number of techniques has been developed for modelling and predicting time series, among which we highlight the Box-Jenkins approach Boxetal 1994 and the Artificial Neural Networks (ANNs) Dorffner 1996. The earlier approach is only capable to construct linear models. In contrast, ANNs are capable to model non-linear functions. However, problems have to be faced concerning the choice of an appropriated network architecture to model the series to be predicted.

A more promising technique to optimize the structure of ANNs is the use of Genetic Algorithms (GAs) Goldberg 1989. GAs are deployed to optimize, at the same time, different architecture's parameters, such as activation functions, hidden nodes, input variables, among others[14]. A problem to be faced here is that, if the GA's initial population is randomly generated, the algorithm may waste time exploring poor regions in the search space of parameters before converging to the good regions. One way to

improve the GAs' performance, by providing useful information about the search space, is through 'case-initialization', in which the first GA's population consists of well succeeded solutions to problems which are similar to the one being tackled.

After the execution of the GA, the resulting ANN architecture is ready for use and can also be associated to the input series, in order to form a new case to be inserted in the base. We expect that the 'case-initialization' will become better as more cases are inserted in the base[16]. The initial prototype was tested against a random search algorithm for 25-time series. For the validation errors, the GA showed superiority in 64% of the series and for the test errors the obtained gain was observed in 60% of the series.

2.3 Comparing Artificial Neural Networks and Traditional Methods

Artificial neural networks (ANNs) are compared to the more traditional time-series forecasting methods, including winter's exponential smoothing, ARIMA model, and multivariate regression. These methods are chosen because of their ability to model trend and seasonal fluctuations present in aggregate retail sales data[12]. The objectives of this article are two-fold: To show how to forecast aggregate retail sales using ANN and To display how various time-series forecasting methods compare in their forecasting accuracy of aggregate retail sales.

For data containing trend and seasonal patterns, failure to account for these patterns may result in poor forecasts. Over the last few decades several methods such as winter's exponential smoothing, Box-Jenkins ARIMA model and multiple regression have been proposed and widely used to account for these patterns. ANN is a new contender in forecasting trend and seasonal data. Franses and Draisma [2] suggested that ANNs be used to investigate how and when seasonal patterns change over time.

All the above approaches are based on symbolic computation. They are not only able to produce the next number, but also give the underlying function. However, the set of functions which can be learned is restricted. In contrast, artificial neural networks can approximate any arbitrary function. Patterns are generated from the sequences and used as input for network optimization[21]. Inductive reasoning systems require manual

programming but can be applied to several sequences without modification. ANNs do not require manual programming, but individual networks have to be trained for each sequence.

2.4 Model Selection in Neural Networks

There has been a growing interest in the modelling of nonlinear relationships and a variety of test procedures for detecting nonlinearities has been developed. If the aim of analysis is prediction, however, it is not sufficient to uncover nonlinearities. Moreover, we need to describe them through an adequate nonlinear model. Unfortunately, for many applications theory does not guide the model building process by suggesting the relevant input variables or the correct functional form.

This particular difficulty makes it attractive to consider an 'a-theoretical' but flexible class of statistical models. Artificial neural networks are well suited for this purpose as they can approximate virtually any measurable function up to an arbitrary degree of accuracy [1] . This desired flexibility, however, makes the specification of an adequate neural network model even harder. Despite the huge amount of network theory and the importance of neural networks in applied work, there is still little experience with a statistical approach to model selection.

Two fundamental ways can be used to add feedback into feed forward multilayer neural networks. Elman introduced feedback from the hidden layer to the context portion of the input layer[2]. This approach pays more attention to the sequence of input values. Jordan recurrent neural networks use feedback from the output layer to the context nodes of the input layer and give more emphasis to the sequence of output values. Gradient descent is a key concept in neural network optimization. Error back propagation in neural networks is based on this technique. While back propagation is relatively simple to implement, several problems can occur in its use in practical applications, including the difficulty of the network getting trapped in some local minima. Researchers have developed a variety of schemes by which gradient methods, and in particular back propagation learning, can be extended to recurrent neural networks. The back propagation through time approach approximates a recurrent neural network as a sequence of static networks using gradient methods[5]. In another

approach, a master neural network is used to identify suitable dynamical slave networks for processing the given data.

Thus, recurrent neural networks are suitable for working with data which has long-term dependencies, as they can “remember” previous inputs for longer periods of time. Back propagation technique can also be applied to RNNs for network optimization.

2.5 Estimation Methods

Feed-forward neural networks Multi-Layered Perceptron’s are used widely in real-world regression or classification tasks. A reliable and practical measure of prediction “confidence” is essential in real-world tasks[4]. This paper compares three approaches to prediction confidence estimation, using both artificial and real data. The three methods are maximum likelihood, approximate Bayesian and bootstrap[7]. Both noise inherent to the data and model uncertainty are considered.

Neural network predictions suffer uncertainty due to A inaccuracy in the training data and B the limitations of the model. The training set is typically noisy and incomplete not all possible input-output examples are available. Noise is inherent to all real data and contributes to the total prediction variance as data noise variance 2. Moreover, the limitations of the model and the training algorithm introduce further uncertainty to the network’s prediction[3]. Neural networks are trained using an iterative optimization algorithm (e.g. steepest descent). The resultant weight values often correspond, therefore, to a local rather than the global minimum of the error function. Additionally, as the optimization algorithm can only “use” the information available in the training set, the solution is likely to be valid only for regions sufficiently represented by the training data[10]. We call this model uncertainty and its contribution to the total prediction variance model uncertainty variance 2m.

Since neural networks converge much more effectively when using normalized inputs, the numerical values are normalized to the range [0,1] before processing. The linear activation function is used in this approach. Instead of using random initial weights, unsupervised pre-training is used as auto encoder for weight initialization. This

means, the network was trained to generate the input numbers of the series at the output. Such pre-training procedure can help to guide the parameters of the layers towards regions in parameter space where solutions are allowed; that is, near a solution that captures statistical structure of the input.

The network was trained for a maximum of 1000 iterations, omitting the last element. After every 10 training cycles the network was tested on the complete series. If it could predict the final element of the series, it was considered to have successfully learned the rule underlying the series[9]. For training, as for pre-training, the scaled conjugate gradient back propagation algorithm was used.

2.6 Statistical Methods

Learning and generalization in neural networks strongly depends on the complexity of the network used, including the architecture of the network, the number and types of parameters used by the network, the procedure used for initialization of its parameters and the details of the learning procedure[11]. Models that are too complex may learn the training data perfectly but will not generalize well. It is commonly believed that the simplest models have the best generalization capabilities, but proper regularization of the cost function may ensure good generalization even in over-parametrized models. Finding global minimum of a complex, nonlinear error function with many parameters is an NP-hard problem[12]. Construction of appropriate architecture and proper initialization of adaptive parameters should enable finding close to optimal solutions for real-world problems, significantly decreasing the learning time.

The algorithms that appeared most frequently as the top five were all of statistical nature, including four discriminant approaches: linear (LDA), logistic (LOGDA), quadratic discriminant (QDA) and a more involved DIPOL92 method that uses hyperplanes to discriminate between clusters[15]. The last of the top five method, ALLOC80, is based on cauterization/density estimation techniques.

Among the top 5 algorithms MLPs trained with the back propagation algorithm appear only once at the third position and 3 times at the fifth position. This clearly

shows that in most cases MLPs did not find solutions as good as those found by statistical discriminant function methods.

Long training times and the suboptimal results of MLPs seem to be due to the lack of a proper initialization[17]. In a series of computer experiments Schmidhuber and Hochreiter observed that repeating random initialization ("guessing" the weights) many times is the fastest way to convergence. In other words, even sophisticated learning procedures are not able to compensate for bad initial values of weights, while good initial guess leads to fast convergence even with simple gradient-based error minimization techniques[18]. Therefore, good strategy is to abandon training as soon as it slows down significantly and start again from random weights. Wrong initialization may create network of sigmoidal functions dividing the input space into areas where the network function gives constant inputs for all training data, making all gradient procedures useless.

Therefore, it is proposed that random initialization be used as the activation function in artificial neural networks for the task of recommendation system. Several random initialization schemes have recently been compared by Thimm and Fiesler using a very large number of computer experiments[19]. The best initial weight variance is determined by the dataset, but differences for small deviations are not significant and weights in the range ±0.77 give the best mean performance.

2.7 Summary

It can be inferred that neural networks can be used to recommendation system for crops. By using random initialization with identity initialization of recurrent weight matrix, better accuracy may be achieved in learning of datasets.

3. METHODOLOGY

Each input sequence is preprocessed to generate a set of patterns as training data. The crop-product coder is then used to train the initial weights of the network. The network is then trained on the data, withholding the last pattern. The last pattern is used to predict the next number of the series.

Training data is generated from the input sequences. Weight matrices are initialized randomly or using crop-product coder. The activation function is used to introduce non-linearity, so that the network can learn non-linear functions also. The cost function calculates the difference between the predicted and actual outputs. The gradient descent algorithm is used for weight optimization. Finally, the optimized weights are used to predict the last number of the series. The implementation was done in IBM cloud environment using Python with NUMPY, PANDAS, SEABORN, MATPLOTLIB libraries.

3.1 Existing system – Graph Based Recommendation System

Recommender systems are filters which suggest items or information that might be interesting to users. These systems analyze the past behavior of a user, build her profile that stores information about her interests, and exploit that profile to find potentially interesting items. The main limitation of this approach is that it may provide accurate but likely obvious suggestions, since recommended items are similar to those the user already knows. In this paper we investigate this issue, known as overspecialization or serendipity problem, by proposing a strategy that fosters the suggestion of surprisingly interesting items the user might not have otherwise discovered. The proposed strategy enriches a graph-based recommendation algorithm with background

knowledge that allows the system to deeply understand the items it deals with.The hypothesis is that the infused knowledge could help to discover hidden correlations among items that go beyond simple feature similarity and therefore promote non-obvious suggestions. Two evaluations are performed to validate this hypothesis: an in vitro experiment on a subset of the HETREC2011-MOVIELENS-

2K dataset, and a preliminary user study. Those evaluations show that the proposed strategy actually promotes non-obvious suggestions, by narrowing the accuracy loss

3.2 Artificial Neural Network based Recommendation System

The proposed system is based on neural networks to optimize the existing structure. This algorithm is developed to meet the time complexity as well as efficiency factor. Recommender system should need to be with the optimal time complexity factor to frame the definite structure. Recommending abnormal inputs is the major issue facing in the availing system. That is where the existing system is logging off. To make its possible our system will use neural networks to process the abnormal inputs. The neurons will consider all weighted parameters during the neural network training. We consider a probability distribution that generates network structures, and optimize the parameters of the distribution instead of directly optimizing the network structure. The proposed method can apply to the various network structure optimization problems under the same framework. The proposed system will contribute majorly on copper, potassium, iron, magnesium, oxygen, sulphur, manganese, calcium, boron, cobalt, molybdenum, chlorine. Along with the nutrient factor it also contributes to crop duration, weather condition, durability, soil testing. The experimental results show that the proposed method can find the appropriate and competitive network structures.

3.3 Hardware and Software Requirement

IBM's one-stop cloud computing shop provides all the cloud solutions and IBM cloud tools EnvironmentsDefine the hardware size and software configuration for the runtime associated with Watson Studio tools such as notebooks.

- 4 vCPU and 16 GB RAM

3.4 Modules

1. Processing Knowledge Infusion
2. Implementation of Graph Based Algorithm
3. Generation of Correlation Matrix
4. Implementation of Artificial Neural Network
5. Implementation of Gradient Descent Algorithm
6. Activation and Training Neural Network

3.4.1 Processing Knowledge Infusion

The Knowledge Infusion (KI) process builds a computer-understandable knowledge repository which constitutes the cultural and linguistic background of the system. The repository is automatically fed by information obtained from several knowledge sources freely available, such as Wikipedia. The main motivation for this choice, compared to the adoption of specific handcrafted ontologies, is the willingness to design a general strategy which allows to update the knowledge repository easily, as well as to plug in additional sources, without changing the overall organization and implementation of the process. KI consists of two steps:

1. Knowledge Extraction and Harmonization: Linguistic knowledge is extracted from WordNet, while encyclopaedic knowledge is obtained from Wikipedia. Due to the different organization of the sources (articles in Wikipedia, Synsets in WordNet), a harmonization phase turns the extracted concepts in a homogeneous format. Linguistic knowledge is useful to recognize general concepts into item descriptions, while encyclopaedic knowledge is useful to recognize specific concepts or named entities, usually not included in a dictionary.

2. Reasoning: It allows to make inference on the background knowledge and item descriptions, in order to discover information potentially useful for the recommendation step.

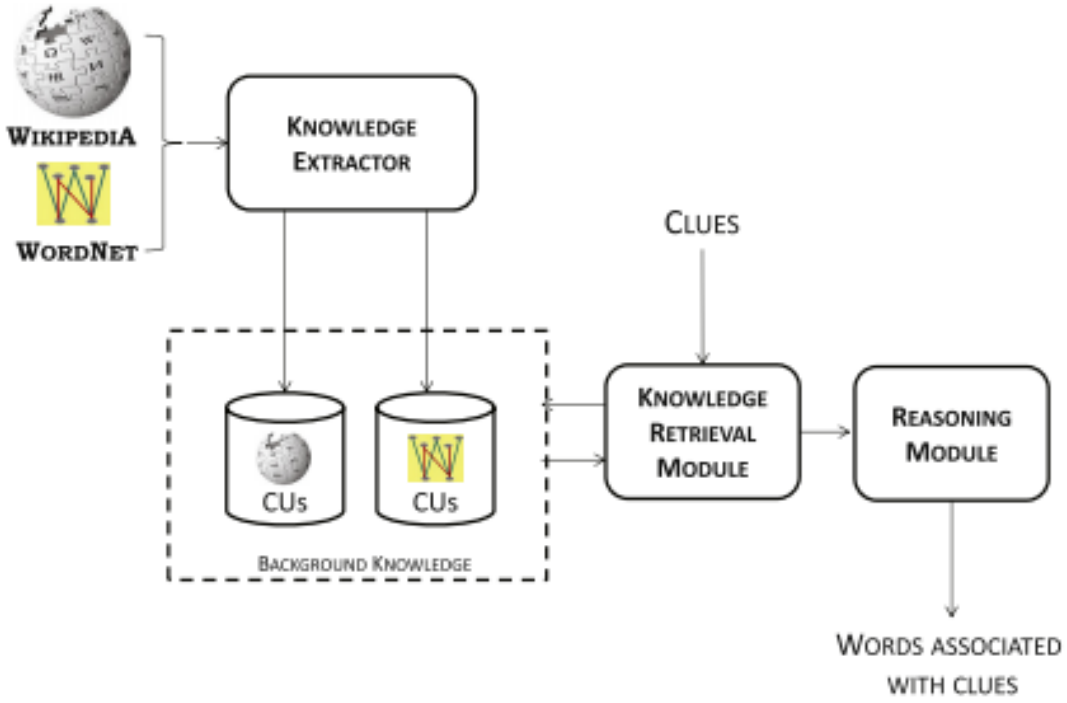

Figure 1 Existing System Block Diagram

3.4.2 Implementation of Graph Based Algorithm

The construction of the SAN is described in the following paragraphs and the result is shown in Fig. 3. Initially, two source nodes labeled with the two clues are included into the SAN. Then, retrieved CUs are included in the SAN. Each CU is linked to the corresponding source node; the edge is oriented from the clue to the CU and is labeled with the cosine similarity value between the clue and the CU. At this stage of the process, edges represent associations between clues and CUs, while similarity values measure the strength of those relationships. Finally, for each CU node, word nodes labeled with terms in the BOW of the CU are included in the SAN. Links are created from the CU node towards its word nodes and labeled with tf-idf scores of words.

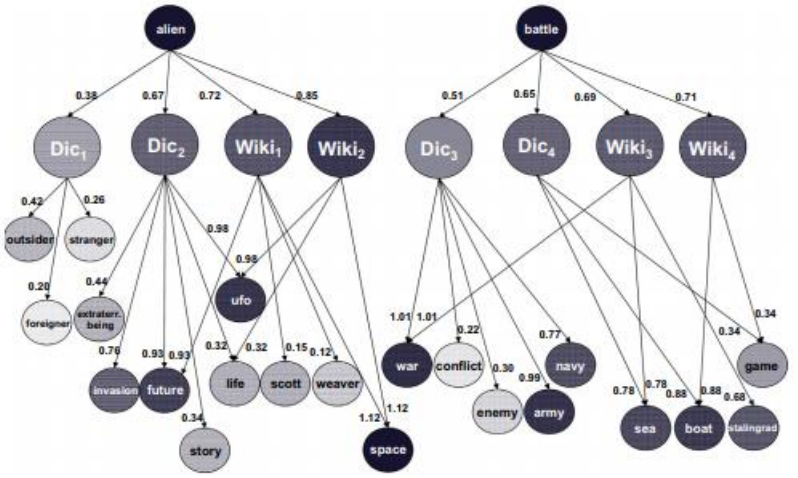

Figure 2 Graph algorithms workflow diagram

3.4.3 Generation of Correlation Matrix

Given an item I, the idea is to exploit the keywords associated with I by KI to compute the correlation index between I and other items in the collection. We adopt a content-based model in which each item I is represented as a vector in a n-dimensional space of features [3]: Features are keywords extracted from item descriptions, therefore the feature space is the vocabulary of the item collection, while wi is the score of feature ki in the item I, which measures the importance of that feature for the item. Given a query q, the ranking function adopted for searching in the item collection is based on the BM25 probabilistic retrieval framework [2]:

Id is frequency of the term t in the item I; a1 and b are parameters usually set to 2 and 0:75 respectively, avgdl is the average item length and id is the standard inverse document frequency of term t in the item collection. The procedure Algorithm for building the correlation matrix follows three main steps:

3.4.4 Implementation of Artificial Neural Network

A simple recurrent network is very similar to a normal neural network, but it has an extra context layer connected to the hidden layer. This context layer stores the output of the hidden layer from one-time step (t) and feeds it to the hidden layer during the next time step (t+1). The difference between their architectures is shown in below.

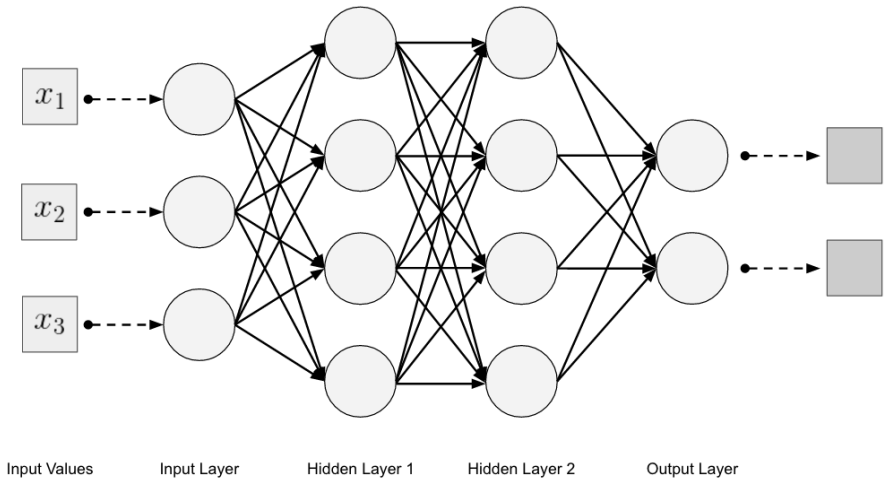

Figure 3 Architecture of Artificial Neural Network

3.4.5 Implementation of Gradient Descent Algorithm

The gradient descent algorithm, along with the backpropagation technique, is used to optimize the weights of the neural network. During the forward pass, the network uses the weights to predict the output. The "cost" or error value i.e. the difference between the actual and predicted output is back propagated through the network and the gradients are used to update the weight matrices. During each step, the gradient descent algorithm takes a small step in the direction which has the lowest slope. This is repeated several times until the global minimum is reached, and thus, the network is optimized. Equation 1 and 2 show the computations to get the gradients for each layer, from right to left.$\delta^{(l)}$denotes theerror values of nodes in layer *l*. $\Theta^{(l)}$denotes the weight matrix from layer *l* to layer *l*+1. *g* is the activation function, $z^{(l)}$ denotes the input values to layer *l*, and $a^{(l)}$ is the activation at layer *l*.

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) \ * \ g'(z^{(l)}) \qquad (1)$$

Equation 1 Gradient Descent Algorithm

The visualization of cost where the global minimum is at the center. J(w) denotes the cost for the weights w. The steps taken towards reaching the minimum are highlighted in black.

$$g'\left(z^{(l)}\right) = a^{(l)} * (1 - a^{(l)}) \quad (2)$$

Equation 2 Gradient Descent Algorithm

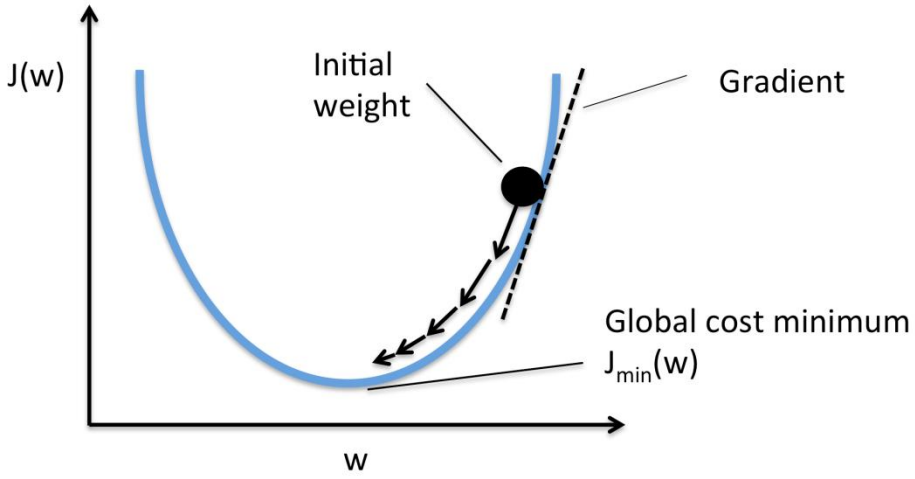

Figure 4 Illustration of Gradient descent

3.4.6 Activation and Training Neural Network

The activation function is used to introduce non-linearity in the network. Without non-linearity, the output can only be a linear combination of the inputs. But the dataset may contain sequences whose underlying function cannot be approximated to a linear combination of inputs. Thus, non-linearity is required to successfully learn such sequences. Common activation functions include hyperbolic tangent, shown in Equation 3and rectified linear function shown in Equation. Here, z denotes the linear combination of weights and inputs in each unit of the network.

$$tanh(z) = \frac{2}{1 + e^{-2z}} - 1$$
$$relu(z) = \max(0, z) \quad (3)$$

Equation 3 Activation Function and Training Neural Network

4. RESULT

4.1 Data Set

ID	Copper (Cu)	Oxygen (O)	Potassium (K)	Iron (Fe)	Sulphur (S)	Maganese (M)	Calcium (Ca)	Boron (B)	Magnesium (Mg)	Chlorine (Cl)	Molybdenum (Mo)	Cobalt (Co)	Crops
1	0.001840628	0.846480123	0.792425808	0.95396414	0.144008693	0.057557329	0.942612066	0.358538846	0.66071358	0.154560807	0.184617004	0.172706131	Tobacco
2	0.504545666	0.519360865	0.864398811	0.353121735	0.55107833	0.317567829	0.442820764	0.489526245	0.412252446	0.469095401	0.35862341	0.405991712	Beans & Mutter(Vegetable)
3	0.660861952	0.118320343	0.058324758	0.35442098	0.422685188	0.381403206	0.836888573	0.10280745	0.624289746	0.422879806	0.668280315	0.785780355	Coriander
4	0.364463392	0.571946721	0.466140653	0.089576091	0.504399277	0.085952615	0.681279086	0.902109955	0.408923579	0.441937822	0.64094136	0.467315691	Bajra
5	0.49136828	0.623866025	0.778153948	0.360337446	0.163507298	0.408652546	0.798647202	0.835477915	0.670103264	0.704099997	0.754202193	0.356694274	Jack Fruit
6	0.580389892	0.67119719	0.540267463	0.094830433	0.844869077	0.275849614	0.275162687	0.147179811	0.154821581	0.968825655	0.994910953	0.228364135	Other Fresh Fruits
7	0.354748037	0.753358739	0.786053931	0.664829927	0.72470386	0.045600892	0.630232106	0.241547502	0.965049614	0.710918831	0.701200792	0.451235863	Blackgram
8	0.056304664	0.698806883	0.394517802	0.04122954	0.54715927	0.230264926	0.851867806	0.738495677	0.02601022	0.131027272	0.339853031	0.471819417	Pome Fruit
9	0.266470943	0.544255831	0.074289349	0.489480055	0.431741527	0.34141218	0.733582678	0.42615788	0.932914269	0.530232825	0.266076277	0.502294746	Dry ginger
10	0.214757335	0.06066292	0.175718565	0.898270098	0.5007234	0.873578021	0.260967239	0.007251817	0.799517086	0.293236787	0.657988585	0.79095239	Oilseeds total
11	0.488027613	0.15890929	7.07E-04	0.763109893	0.800202416	0.103288539	0.41875357	0.610702575	0.855450149	0.329031055	0.478511959	0.599108516	Maize
12	0.949140799	0.750996453	0.698857103	0.259525769	0.973755425	0.104152034	0.933526807	0.217139801	0.692408998	0.870229786	0.372251064	0.683566512	Cashewnut
13	0.54088441	0.502310192	0.489835885	0.823773994	0.350498367	0.725370051	0.521138141	0.992504079	0.605306959	0.413310918	0.938498943	0.270327324	Snak Guard
14	0.558223286	0.854858215	0.878494271	0.126799123	0.556029489	0.124397084	0.02206379	0.309694451	0.074323464	0.572901758	0.747449177	0.094865601	Safflower
15	0.822149423	0.868054079	0.661293963	0.865104668	0.842745635	0.213765309	0.926413935	0.651924387	0.507538964	0.418078962	0.674335994	0.169059562	Cashewnut Processed
16	0.820892727	0.934997736	0.064366086	0.075611416	0.835505961	0.476700028	0.643877015	0.853638708	0.305126275	0.334056152	0.793401882	0.433590536	Pear
17	0.009201306	0.139861474	0.731818097	0.053061039	0.356181438	0.884395895	0.863141963	0.726179726	0.186759005	0.212915323	0.69671842	0.30349399	Potato
18	0.569647306	0.296516158	0.113009	0.142388936	0.365365547	0.789035995	0.894552264	0.729816641	0.884369499	0.901102262	0.951357672	0.261052358	Sesamum
19	0.266999551	0.606831011	0.730698254	0.336685387	0.616923599	0.363860041	0.960392018	0.346004964	0.127660917	0.403315129	0.516619876	0.744920049	Varagu
20	0.703844097	0.425386429	0.78385469	0.141674633	0.116885455	0.508112717	0.605472094	0.307353376	0.188781972	0.753954152	0.409484656	0.171097485	Grapes
21	0.390800291	0.604544703	0.844451617	0.651233141	0.220538757	0.223979571	0.789739273	0.890475688	0.343735752	0.137018008	0.17584923	0.08306157	Pineapple
22	0.748287312	0.475596355	0.18479494	0.040016956	0.945743592	0.137328695	0.898707924	0.861785677	0.030830277	0.894898564	0.685288138	0.251710716	Masoor
23	0.167810114	0.093298319	0.382584759	0.289210791	0.047301503	0.606408648	0.141117351	0.938189978	0.564315201	0.721370769	0.899211157	0.424292243	Samai

Table 1 Dataset table

Open Government Data Platform (OGD) India is a single-point of access to Datasets/Apps in open format published by Ministries/Departments.

- https://data.gov.in/
- https://github.com/CreativeCrops/Review-II/blob/master/dataset_v2.csv

Total of Eight Attributes

Name	**Datatype**
ID	Number
Copper (Cu)	Float
Oxygen (O)	Float
Potassium (K)	Float
Iron (Fe)	Float
Sulphur (S)	Float
Maganese (M)	Float
Calcium (Ca)	Float
Boron (B)	Float
Magnesium (Mg)	Float
Chlorine (Cl)	Float
Molybdenum (Mo)	Float
Cobalt (Co)	Float
Day's	Number
Crops	String

Table 2 Total of Eight Attributes

4.2 Evaluation Metric

TN / True Negative	Crops recommended were not Relevant
TP / True Positive	Crops Recommended were Relevant
FN / False Negative	Crops which are not Recommended are Relevant
FP / False Positive	Crops which are not Recommended are not Relevant

Table 3 Symbolic Notation

4.2.1 Precision

Imagine there are 100 positive cases among 10,000 cases. You want to predict which ones are positive, and you pick 200 to have a better chance of catching many of the 100 positive cases. You record the IDs of your predictions, and when you get the actual results you sum up how many times you were right or wrong. There are four ways of being right or wrong

$$\text{Precision} = \frac{TP}{\text{FP} + \text{TP}} \tag{4}$$

Equation 4 Precision

4.2.2 Recall

Recall is considered successful when upon starting from an initial cue the network converges to a stable state which corresponds to the learned memory nearest to the input pattern. Inter-pattern distance is measured by the Hamming distance between the input and the learned item encodings. If the network converges to a non-memory stable state, its output will stand for a 'failure of recall' response.

$$\text{Precision} = \frac{TP}{\text{FN} + \text{TP}} \tag{5}$$

Equation 5 Recall

4.2.3 Fallout Rate

Dropout momentarily (in a batch of input data) switches off some neurons in a layer so that they do not contribute any information or learn any information during those updates, and the onus falls on other active neurons to learn harderand reduce the error.

$$\text{Fallout Rate} = \frac{FP}{FP + TN} \tag{6}$$

Equation 6 Fallout Rate

4.3 Expected Outcome

The network will predict the crop with its durability in accordance with reduced error rates and considerable accuracy. For each network architecture, the number of crops predicted is used as the evaluation metric.

ID	**Crops**		**Prediction**
43	Linseed		Linseed
33	Khesari		Khesari
22	Tobacco		Tobacco
67	Other Cereals & Millets		Millets
65	Other Vegetables		Vegetables
87	Beet Root		Beet Root
Correct: 12 / 12 : 100.0 %			

Table 4 Expected Outcome

5. CONCLUSION

Thus, artificial neural networks can be used to predict agricultural crops. In presenting and justifying the modules, It is provided a broad and in-depth review of our prior work related to example critiquing regarding crop prediction in recommender systems. Most importantly, a framework of three evaluation criteria was proposed to determine the usability of such systems: decision accuracy, error rate, and prediction time. Within this framework, there are selected techniques, which have been validated through empirical studies, to demonstrate how to implement the neural network. Emphasis was given to those techniques that achieve a good balance on all of the criteria. Adopting these guidelines, therefore, should significantly enhance the usability of crop recommender systems and their usage among the platform. Collaborative Filtering is a widely used solution for this problem which is used in this project.

Recommendation systems provide content for us by taking what other people recommend as well as our selections into account. The presence of random initialization and crop-product coder improves the performance of the network by a small margin on the provided dataset. Using artificial neural networks, it is identified that it can predict the hidden pattern which is not in use till now. With the neural networks, there are infinite number neurons and each neurons will be updated with hidden components. With the given input parameters, the matched pattern will get activated through random initialization and the matched pattern will be predicted as output. Similar matches will be returned as output. Such that it can recommend up to 10 crops. The Farmer has the privilege to choose among the 10 crops through which his land will get improved nutrient content. A key issue that emerges from this study, asking for future research, refers to the effectiveness of the recommendations generated by such a system and how this can be improved using the neural networks to the recommendations. In this way the system will not only use soil data as input, but also the crops which are already cultivated to the recommendations, which is the most basic measure of its effectiveness. With a crop-product coder, the SAN is able to solve about 32% of the sequences, but when the weights are randomly initialized, 42% of the sequences can be solved.

REFERENCES

[1] L.R.Medsker," Recurrent Neural Networks": *Design and Applications,* 2001, , vol. 7006, pp. 255-259, 2011.

[2] S. Gluge and W. Andreas, "Solving Number Series with Simple Recurrent Networks," *Lecture Notes in Computer Science,* vol. 7930, pp. 412-420, 2013.

[3] U. Schmid and M. Ragni, "Comparing Computer Models Solving Number Series Problems," *Lecture Notes in Computer Science,* vol. 9205, pp. 352-361, 2015.

[4] U. Schmid and E. Kitzelmann, "Inductive rule learning on the knowledge level," *Cognitive Systems Research,* pp. 237-248, 2011.

[5] M. Ragni and A. Klein, "Predicting Numbers: An AI Approach to Solving Number Series," *Lecture Notes in Artificial Intelligence,* vol. 7006, pp. 255-259, 2011.

[6] J. L. Elman, "Finding structure in time," *Cognitive Science,* vol. 14, no. 179, 1990.

[7] M. Jordan, "Generic constraints on underspecified target trajectories," in *International Joint Conference on Neural Networks,* 1989.

[8] N. Jaitly, Q. V. Le and G. E. Hinton, "A simple way to initialize recurrent networks of rectified linear units," *arXiv,* vol. 1504, no. 00941, 2015.

[9] M. Meredith, "Seek-whence: a model of pattern perception," Technical report, Indiana Univ., Bloomington (USA), 1986.

[10] P. Sanghi and D. Dowe, "A computer program capable of passing I.Q. tests," in *7th Conf. of the Australasian Society for Cognitive Science*, Sydney, Australia, 2003.

[11] "The Online Encyclopedia of Integer Sequences," [Online]. Available: http://oeis.org. [Accessed June 2016].

[12] "Activation Function," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Activation_function. [Accessed August 2016].

[13] M. Ragni and A. Klein, "Solving number series - Architectural Properties of Successful Artificial Neural Networks," *Neural Computation Theory & Applications,* pp. 224-229, 2011.

[14] M. a. F. R. Aitkin, "tatistics and Computing," *Statistical modelling of artificial neural networks using multi-layer perceptron,* vol. 3, no. 25, p. 227–239, 2003.

[15] I. Q. M. a. S. Alon, "A comparison of artificial neural networks and traditional methods.,," *Journal of Retailer and Consumer Services* , p. 147–156, 2001.

[16] U. a. K. Anders, "A comparison of artificial neural networks and traditional methods," *Journal of Retailer and Consumer Services 8,* vol. 8, p. 147–156, 2010.

[17] U. a. K. Anders, "Model selection in neural networks," *Neural Networks* , vol. 12, p. 309–323, 2011.

[18] C. L. Barat, "Neural and statistical classifiers.," *Instrumentation and Measurement Technology Conference,* vol. 3, p. 1480–1486, 2014.

[19] R. a. L. Bastos Cavalcante, "Design of neural networks for time series prediction using case-initialized genetic algorithms," *International Conference on Neural Information Processing,* vol. 3, p. 382–387, 2012.

[20] J. a. S. Benediktsson, "Neural network approaches versus statistical methods in classification of multisource remote sensing data," *Geoscience and Remote Sensing* , vol. 2, p. 540–552, 2010.

[21] J. G. Bernier, "Evaluating the impact of multiplicative input perturbations on radial basis function networks.," *European Symposium on Artificial Neural Networks,* vol. 4, p. 237–244, 2013.

APPENDIX A : SAMPLE CODE

```
# coding: utf-8
# In[1]:
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns # visualization
import matplotlib.pyplot as plt
# In[2]:
import sys
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share your notebook.
client_37709ff639f94158ac87880ea34e5fa2 = ibm_boto3.client(service_name='s3',
ibm_api_key_id='bRKbd0AkhpRvbRLWUFKkvNsZ1fFc6cMB2O_G1o7WEiCy',
ibm_auth_endpoint="https://iam.eu-gb.bluemix.net/oidc/token",
config=Config(signature_version='oauth'),
endpoint_url='https://s3.eu-geo.objectstorage.service.networklayer.com')

body = client_37709ff639f94158ac87880ea34e5fa2.get_object(Bucket='dataanalysisv2-donotdelete-pr-xbeaxc4dj5ykwg',Key='Iris.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

data = pd.read_csv(body)
data.head()
```

```
# In[3]:
body = client_37709ff639f94158ac87880ea34e5fa2.get_object(Bucket='dataanalysisv2-donotdelete-pr-xbeaxc4dj5ykwg',Key='dataset_v1.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

df_data_1 = pd.read_csv(body)
df_data_1.head()



# In[4]:
body = client_37709ff639f94158ac87880ea34e5fa2.get_object(Bucket='dataanalysisv2-donotdelete-pr-xbeaxc4dj5ykwg',Key='dataset_v2.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

df_data_1 = pd.read_csv(body)
df_data_1.head()



# In[5]:
#data.sample(n=5)
df_data_1.sample(n=5)



# In[6]:
data.describe()
df_data_1.describe()



# In[7]:
#sns.pairplot( data=data, vars=('SepalLengthCm','SepalWidthCm','PetalLengthCm','PetalWidthCm'), hue='Species' )
```

```
sns.pairplot( data=df_data_1, vars=('Copper (Cu)','Oxygen (O)','Potassium (K)','Iron (Fe)','Sulphur (S)','Maganese (M)','Calcium (Ca)','Boron (B)','Magnesium (Mg)','Chlorine (Cl)','Molybdenum (Mo)','Cobalt (Co)'), hue='Crops' )


# In[8]:
#df_norm = data[['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']].apply(lambda x: (x - x.min()) / (x.max() - x.min()))
#df_norm.sample(n=5)

df_norm = df_data_1[['Copper (Cu)','Oxygen (O)','Potassium (K)','Iron (Fe)','Sulphur (S)','Maganese (M)','Calcium (Ca)','Boron (B)','Magnesium (Mg)','Chlorine (Cl)','Molybdenum (Mo)','Cobalt (Co)']].apply(lambda x: (x - x.min()) / (x.max() - x.min()))
df_norm.sample(n=5)


# In[9]:
df_norm.describe()


# In[14]:
listCrops=list(df_data_1['Crops'].unique())


# In[18]:
listCropCount=list(range(len(listCrops)))


# In[21]:
listname=[]

target = df_data_1[['Crops']].replace(listCrops,listCropCount);
target.sample(n=5)


# In[22]:
df = pd.concat([df_norm, target], axis=1)
df.sample(n=5)
```

```
# In[23]:
train_test_per = 90/100.0
df['train'] = np.random.rand(len(df)) < train_test_per
df.sample(n=5)

# In[24]:
train = df[df.train == 1]
train = train.drop('train', axis=1).sample(frac=1)
train.sample(n=5)
# In[25]:
test = df[df.train == 0]
test = test.drop('train', axis=1)
test.sample(n=5)


# In[26]:
X = train.values[:,:4]
X[:5]


# In[29]:
targets = [[1,0,0],[0,1,0],[0,1,1]]
y = np.array([targets[int(x)] for x in train.values[:,4:5]])
y[:5]


# In[30]:
num_inputs = len(X[0])
hidden_layer_neurons = 5
np.random.seed(4)
w1 = 2*np.random.random((num_inputs, hidden_layer_neurons)) - 1
w1


# In[31]:
num_outputs = len(y[0])
w2 = 2*np.random.random((hidden_layer_neurons, num_outputs)) - 1
w2


# In[64]:
# taken from> https://gist.github.com/craffel/2d727968c3aaebd10359
```

```
def draw_neural_net1(ax, left, right, bottom, top, layer_sizes):
'''
Draw a neural network cartoon using matplotilb.

    :usage:
        >>> fig = plt.figure(figsize=(12, 12))
        >>> draw_neural_net(fig.gca(), .1, .9, .1, .9, [4, 7, 2])

    :parameters:
    - ax : matplotlib.axes.AxesSubplot
        The axes on which to plot the cartoon (get e.g. by plt.gca())
    - left : float
        The center of the leftmost node(s) will be placed here
    - right : float
        The center of the rightmost node(s) will be placed here
    - bottom : float
        The center of the bottommost node(s) will be placed here
    - top : float
        The center of the topmost node(s) will be placed here
    - layer_sizes : list of int
        List of layer sizes, including input and output dimensionality
    '''
    n_layers = len(layer_sizes)
    v_spacing = (top - bottom)/float(max(layer_sizes))
    h_spacing = (right - left)/float(len(layer_sizes) - 1)
    # Nodes
    for n, layer_size in enumerate(layer_sizes):
        layer_top = v_spacing*(layer_size - 1)/2. + (top + bottom)/2.
        for m in range(layer_size):
            circle = plt.Circle((n*h_spacing + left, layer_top - m*v_spacing),
                v_spacing/4.,
                                color='w', ec='k', zorder=4)
            ax.add_artist(circle)
    # Edges
    for n, (layer_size_a, layer_size_b) in enumerate(zip(layer_sizes[:-1],
        layer_sizes[1:])):
        layer_top_a = v_spacing*(layer_size_a - 1)/2. + (top + bottom)/2.
        layer_top_b = v_spacing*(layer_size_b - 1)/2. + (top + bottom)/2.
        for m in range(layer_size_a):
            for o in range(layer_size_b):
                line = plt.Line2D([n*h_spacing + left, (n + 1)*h_spacing
                    + left],
```

a. [layer_top_a - m*v_spacing, layer_top_b - o*v_spacing], c='k')

2. ax.add_artist(line)

```
def draw_neural_net(ax, left, right, bottom, top, layer_sizes, coefs_, intercepts_, n_iter_, loss_):
    '''
    Draw a neural network cartoon using matplotilb.

    :usage:
        >>> fig = plt.figure(figsize=(12, 12))
        >>> draw_neural_net(fig.gca(), .1, .9, .1, .9, [4, 7, 2])

    :parameters:
    - ax : matplotlib.axes.AxesSubplot
        The axes on which to plot the cartoon (get e.g. by plt.gca())
    - left : float
        The center of the leftmost node(s) will be placed here
    - right : float
        The center of the rightmost node(s) will be placed here
    - bottom : float
        The center of the bottommost node(s) will be placed here
    - top : float
        The center of the topmost node(s) will be placed here
    - layer_sizes : list of int
        List of layer sizes, including input and output dimensionality
    '''
    n_layers = len(layer_sizes)
    v_spacing = (top - bottom)/float(max(layer_sizes))
    h_spacing = (right - left)/float(len(layer_sizes) - 1)

    # Input-Arrows
    layer_top_0 = v_spacing*(layer_sizes[0] - 1)/2. + (top + bottom)/2.
    for m in range(layer_sizes[0]):
        plt.arrow(left-0.18, layer_top_0 - m*v_spacing, 0.12, 0,  lw =1, head_width=0.01, head_length=0.02)

    # Nodes
    for n, layer_size in enumerate(layer_sizes):
        layer_top = v_spacing*(layer_size - 1)/2. + (top + bottom)/2.
        for m in range(layer_size):
```

iii. circle = plt.Circle((n*h_spacing + left, layer_top - m*v_spacing), v_spacing/8.,
1. color='w', ec='k', zorder=4)
iv. if n == 0:
1. plt.text(left-0.125, layer_top - m*v_spacing, r'$X_{'+str(m+1)+'}$', fontsize=15)
v. elif (n_layers == 3) & (n == 1):
1. plt.text(n*h_spacing + left+0.00, layer_top - m*v_spacing+ (v_spacing/8.+0.01*v_spacing), r'$H_{'+str(m+1)+'}$', fontsize=15)
vi. elif n == n_layers -1:
1. plt.text(n*h_spacing + left+0.10, layer_top - m*v_spacing, r'$y_{'+str(m+1)+'}$', fontsize=15)
vii. ax.add_artist(circle)

124. # Bias-Nodes
125. for n, layer_size in enumerate(layer_sizes):
i. if n < n_layers -1:
ii. x_bias = (n+0.5)*h_spacing + left
iii. y_bias = top + 0.005
iv. circle = plt.Circle((x_bias, y_bias), v_spacing/8., color='w', ec='k', zorder=4)
v. plt.text(x_bias-(v_spacing/8.+0.10*v_spacing+0.01), y_bias, r'1', fontsize=15)
vi. ax.add_artist(circle)
126. # Edges
127. # Edges between nodes
128. for n, (layer_size_a, layer_size_b) in enumerate(zip(layer_sizes[:-1], layer_sizes[1:])):
i. layer_top_a = v_spacing*(layer_size_a - 1)/2. + (top + bottom)/2.
ii. layer_top_b = v_spacing*(layer_size_b - 1)/2. + (top + bottom)/2.
iii. for m in range(layer_size_a):
iv. for o in range(layer_size_b):
1. line = plt.Line2D([n*h_spacing + left, (n + 1)*h_spacing + left],
a. [layer_top_a - m*v_spacing, layer_top_b - o*v_spacing], c='k')
2. ax.add_artist(line)
3. xm = (n*h_spacing + left)
4. xo = ((n + 1)*h_spacing + left)
5. ym = (layer_top_a - m*v_spacing)
6. yo = (layer_top_b - o*v_spacing)

7. rot_mo_rad = np.arctan((yo-ym)/(xo-xm))
8. rot_mo_deg = rot_mo_rad*180./np.pi
9. xm1 = xm + (v_spacing/8.+0.05)*np.cos(rot_mo_rad)
10. if n == 0:
    a. if yo > ym:
        i. ym1 = ym + (v_spacing/8.+0.12)*np.sin(rot_mo_rad)
    b. else:
        i. ym1 = ym + (v_spacing/8.+0.05)*np.sin(rot_mo_rad)
11. else:
    a. if yo > ym:
        i. ym1 = ym + (v_spacing/8.+0.12)*np.sin(rot_mo_rad)
    b. else:
        i. ym1 = ym + (v_spacing/8.+0.04)*np.sin(rot_mo_rad)
12. plt.text(xm1, ym1, str(round(coefs_[n][m, o],4)), rotation = rot_mo_deg, fontsize = 10)

129. # Edges between bias and nodes
130. for n, (layer_size_a, layer_size_b) in enumerate(zip(layer_sizes[:-1], layer_sizes[1:])):
    i. if n < n_layers-1:
    ii. layer_top_a = v_spacing*(layer_size_a - 1)/2. + (top + bottom)/2.
    iii. layer_top_b = v_spacing*(layer_size_b - 1)/2. + (top + bottom)/2.
    iv. x_bias = (n+0.5)*h_spacing + left
    v. y_bias = top + 0.005
    vi. for o in range(layer_size_b):
    vii. line = plt.Line2D([x_bias, (n + 1)*h_spacing + left],
        i. [y_bias, layer_top_b - o*v_spacing], c='k')
    viii. ax.add_artist(line)
    ix. xo = ((n + 1)*h_spacing + left)
    x. yo = (layer_top_b - o*v_spacing)
    xi. rot_bo_rad = np.arctan((yo-y_bias)/(xo-x_bias))
    xii. rot_bo_deg = rot_bo_rad*180./np.pi
    xiii. xo2 = xo - (v_spacing/8.+0.01)*np.cos(rot_bo_rad)
    xiv. yo2 = yo - (v_spacing/8.+0.01)*np.sin(rot_bo_rad)
    xv. xo1 = xo2 -0.05 *np.cos(rot_bo_rad)
    xvi. yo1 = yo2 -0.05 *np.sin(rot_bo_rad)
    xvii. plt.text(xo1, yo1, str(round(intercepts_[n][o],4)), rotation = rot_bo_deg, fontsize = 10)

```
        # Output-Arrows
        layer_top_0 = v_spacing*(layer_sizes[-1] - 1)/2. + (top + bottom)/2.
        for m in range(layer_sizes[-1]):
            i. plt.arrow(right+0.015, layer_top_0 - m*v_spacing,
               0.16*h_spacing, 0,  lw =1, head_width=0.01, head_length=0.02)
        # Record the n_iter_ and loss
        plt.text(left + (right-left)/3., bottom - 0.005*v_spacing,
    'Steps:'+str(n_iter_)+'    Loss: ' + str(round(loss_, 6)), fontsize = 15)


        # In[83]:
        dataset = np.mat('-1 -1 -1; -1 1 1; 1 -1 1; 1 1 -1')
        X_train = dataset
        y_train = np.mat('0; 1; 1; 0')
        #-----2-2-1
        #my_hidden_layer_sizes= (5,2)
        #------2-2-8-1
        #my_hidden_layer_sizes= (2, 8,)
        #------2-16-16-1
        my_hidden_layer_sizes= (5, 10,3,6,8,4,)

        XOR_MLP = MLP(
        activation='tanh',
        alpha=0.,
        batch_size='auto',
        beta_1=0.9,
        beta_2=0.999,
        early_stopping=False,
        epsilon=1e-08,
        hidden_layer_sizes= my_hidden_layer_sizes,
        learning_rate='constant',
        learning_rate_init = 0.1,
        max_iter=5000,
        momentum=0.5,
        nesterovs_momentum=True,
        power_t=0.5,
        random_state=0,
        shuffle=True,
        solver='sgd',
        tol=0.0001,
        validation_fraction=0.1,
```

```
166.        verbose=False,
167.        warm_start=False)

168.        XOR_MLP.fit(X_train,y_train)

169.        fig = plt.figure(figsize=(20, 20))
170.        ax = fig.gca()
171.        ax.axis('off')

172.        layer_sizes = [2] + list(my_hidden_layer_sizes) + [1]
173.        draw_neural_net(ax, .1, .9, .1, .9, layer_sizes, XOR_MLP.coefs_,
    XOR_MLP.intercepts_, XOR_MLP.n_iter_, XOR_MLP.loss_)


174.        # In[81]:
175.        fig = plt.figure(figsize=(12, 12))
176.        ax = fig.gca()
177.        ax.axis('off')
178.        draw_neural_net1(ax, .1, .9, .1, .9, [4, 5, 3])


179.        # In[34]:
180.        # sigmoid function representation
181.        _x = np.linspace( -5, 5, 50 )
182.        _y = 1 / ( 1 + np.exp( -_x ) )
183.        plt.plot( _x, _y )


184.        # In[35]:
185.        learning_rate = 0.2 # slowly update the network
186.        for epoch in range(50000):
187.        l1 = 1/(1 + np.exp(-(np.dot(X, w1)))) # sigmoid function
188.        l2 = 1/(1 + np.exp(-(np.dot(l1, w2))))
189.        er = (abs(y - l2)).mean()
190.        l2_delta = (y - l2)*(l2 * (1-l2))
191.        l1_delta = l2_delta.dot(w2.T) * (l1 * (1-l1))
192.        w2 += l1.T.dot(l2_delta) * learning_rate
193.        w1 += X.T.dot(l1_delta) * learning_rate
194.        print('Error:', er)


195.        # In[36]:
```

```
196.    X = test.values[:,:4]
197.    y = np.array([targets[int(x)] for x in test.values[:,4:5]])

198.    l1 = 1/(1 + np.exp(-(np.dot(X, w1))))
199.    l2 = 1/(1 + np.exp(-(np.dot(l1, w2))))

200.    np.round(l2,3)


201.    # In[41]:
202.    test


203.    # In[44]:
204.    yp = np.argmax(l2, axis=1) # prediction
205.    res = yp == np.argmax(y, axis=1)
206.    correct = np.sum(res)/len(res)

207.    testres = test[['Crops']].replace(listCropCount, listCrops)

208.    testres['Prediction'] = yp
209.    testres['Prediction']                                                    =
    testres['Prediction'].replace(listCropCount,listCrops)

210.    print(testres)
211.    print('Correct:',sum(res),'/',len(res), ':', (correct*100),'%')
```

APPENDIX B : SCREENSHOTS

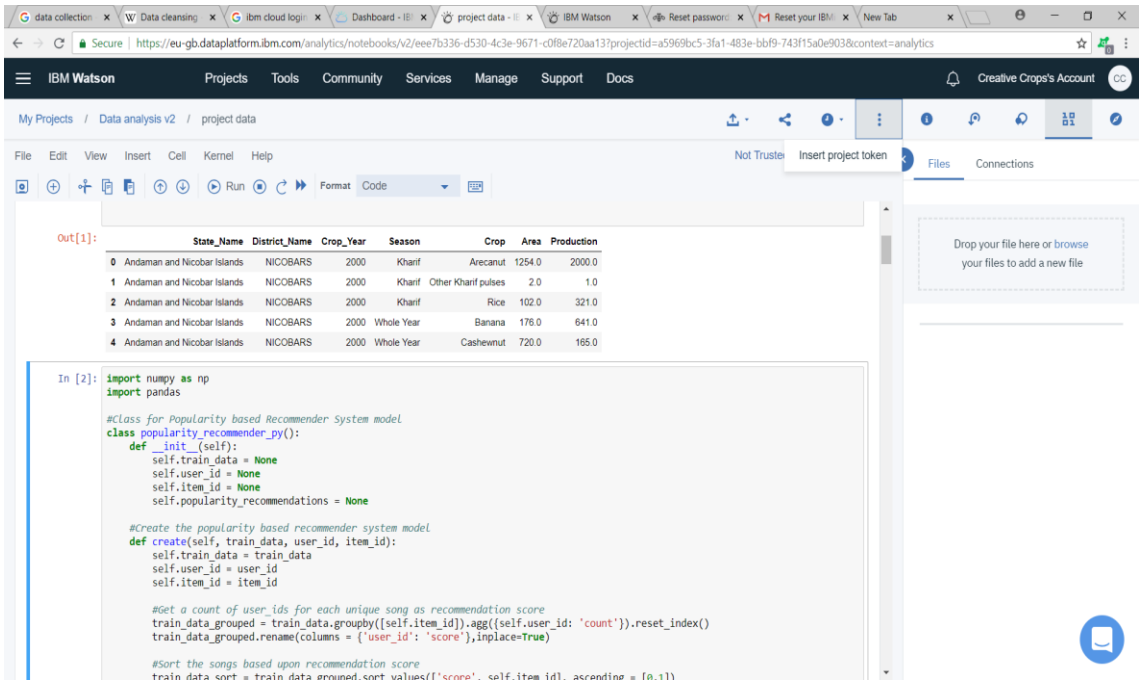

Figure 5 Dataset preprocessing

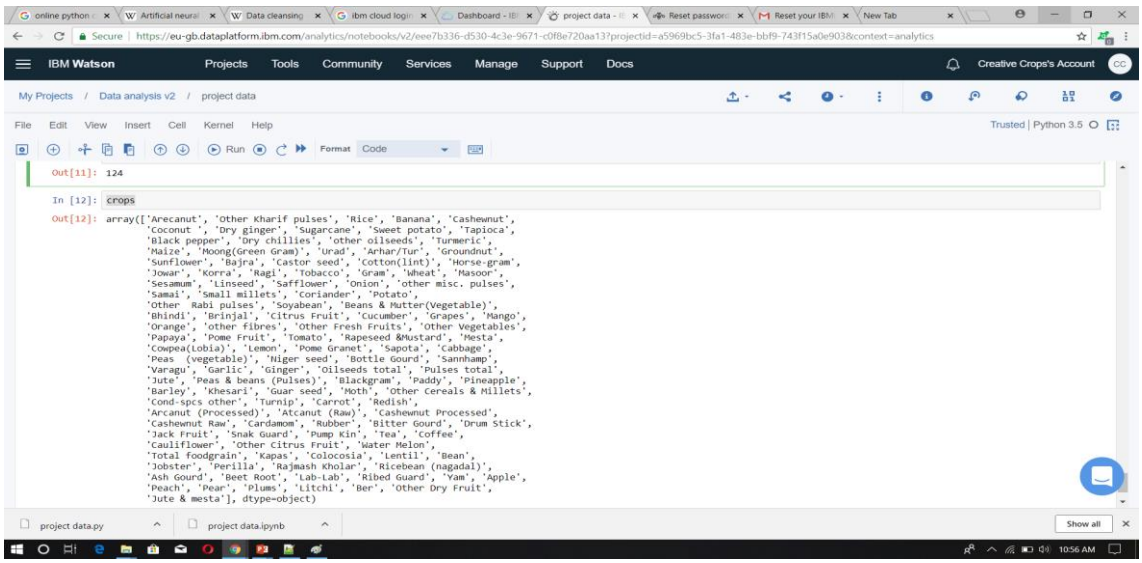

Figure 6 Crops list

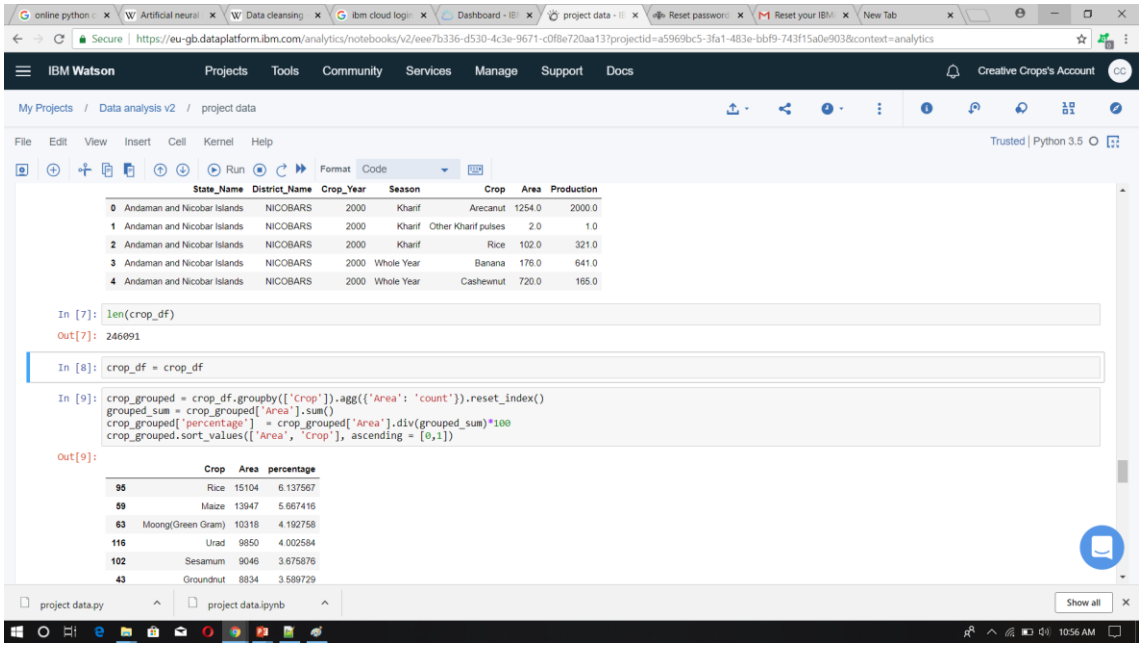

Figure 7 Algorithm design

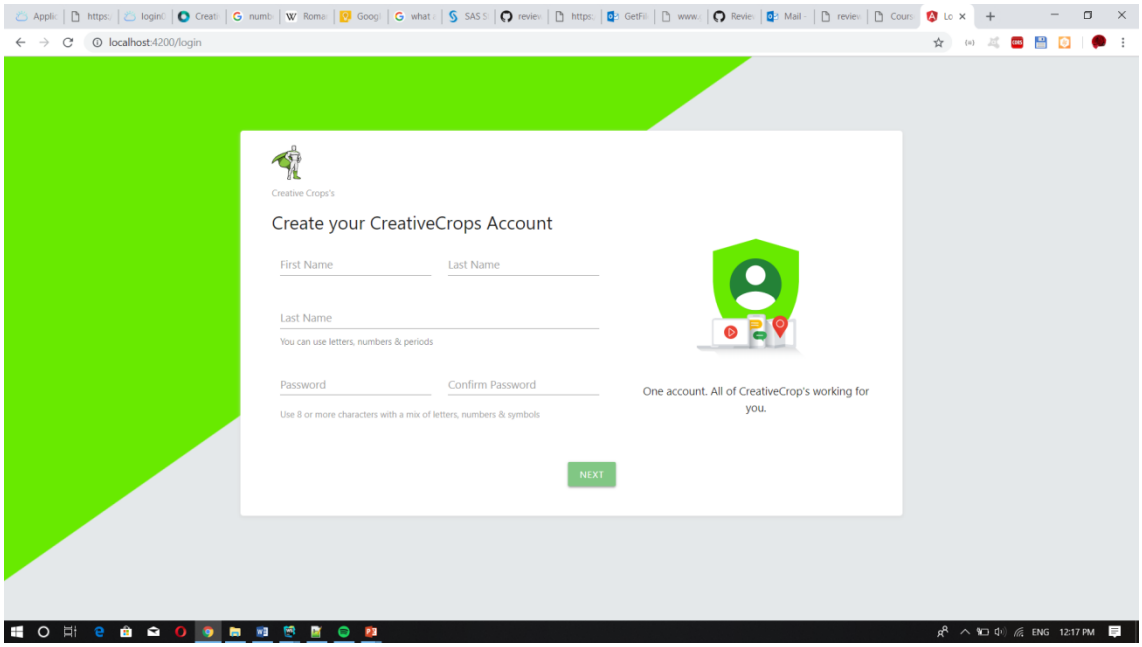

Figure 8 User details

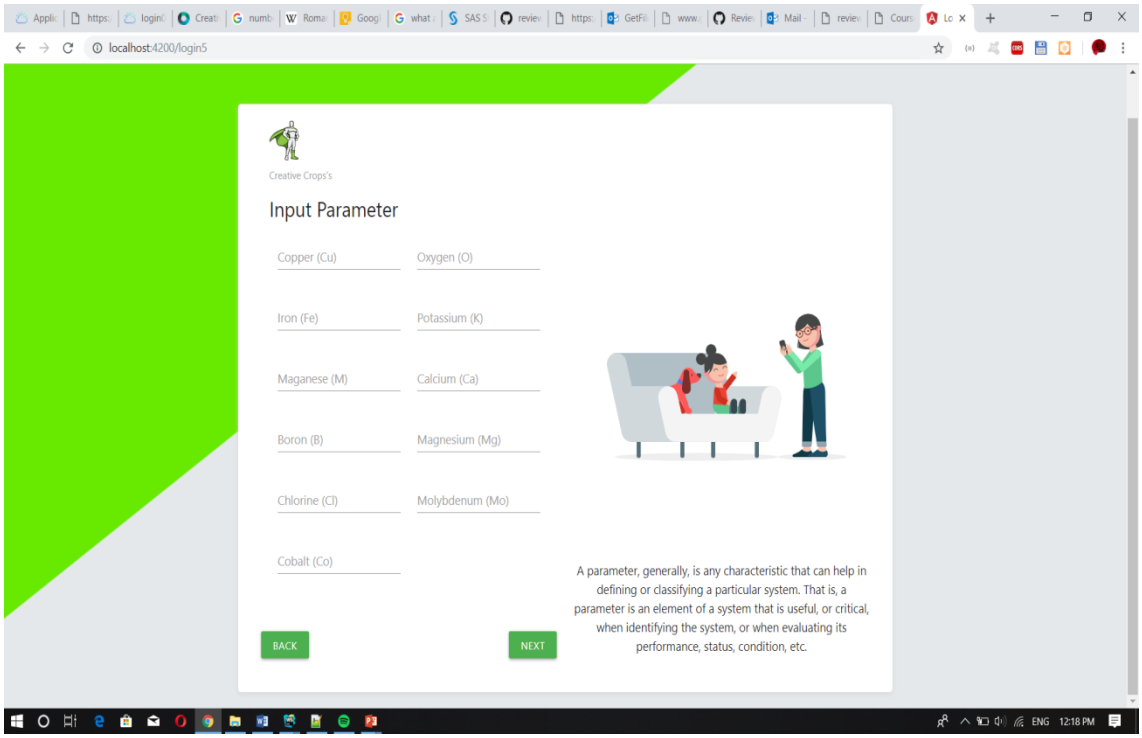

Figure 9 Input parameter

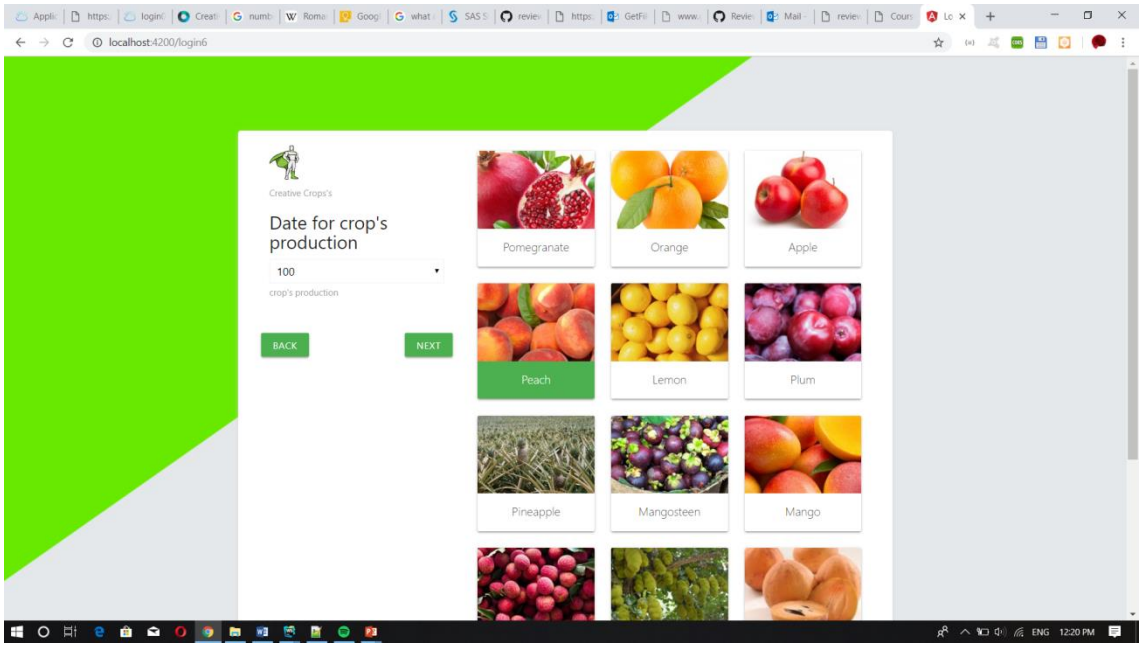

Figure 10 Crop recommendation system

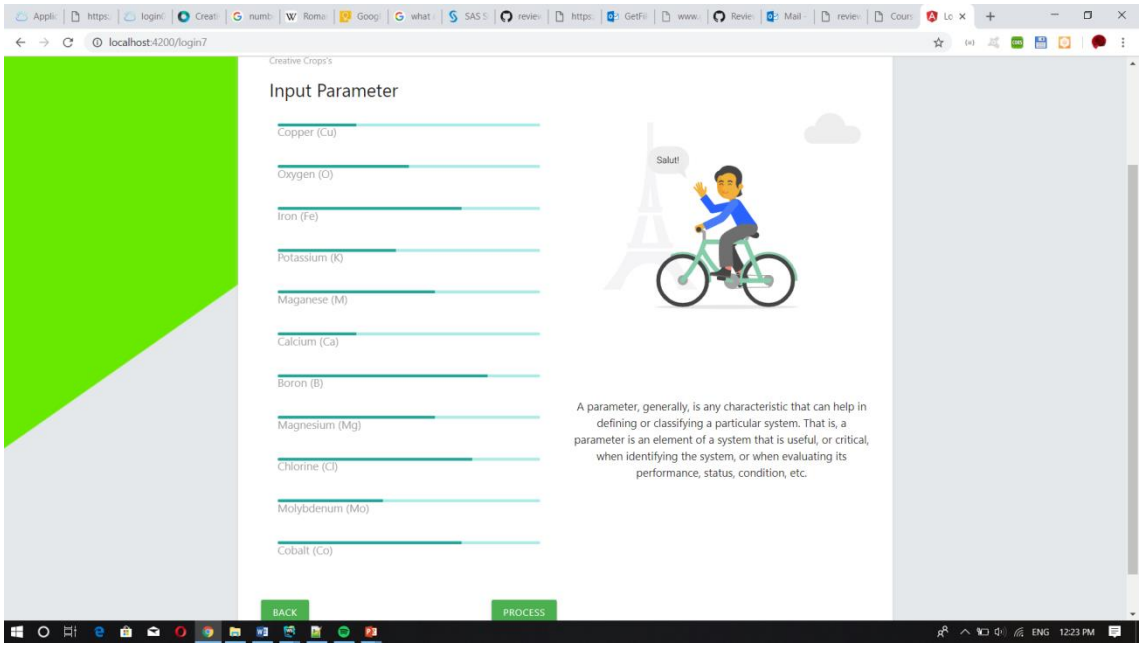

Figure 11 values of input parameter

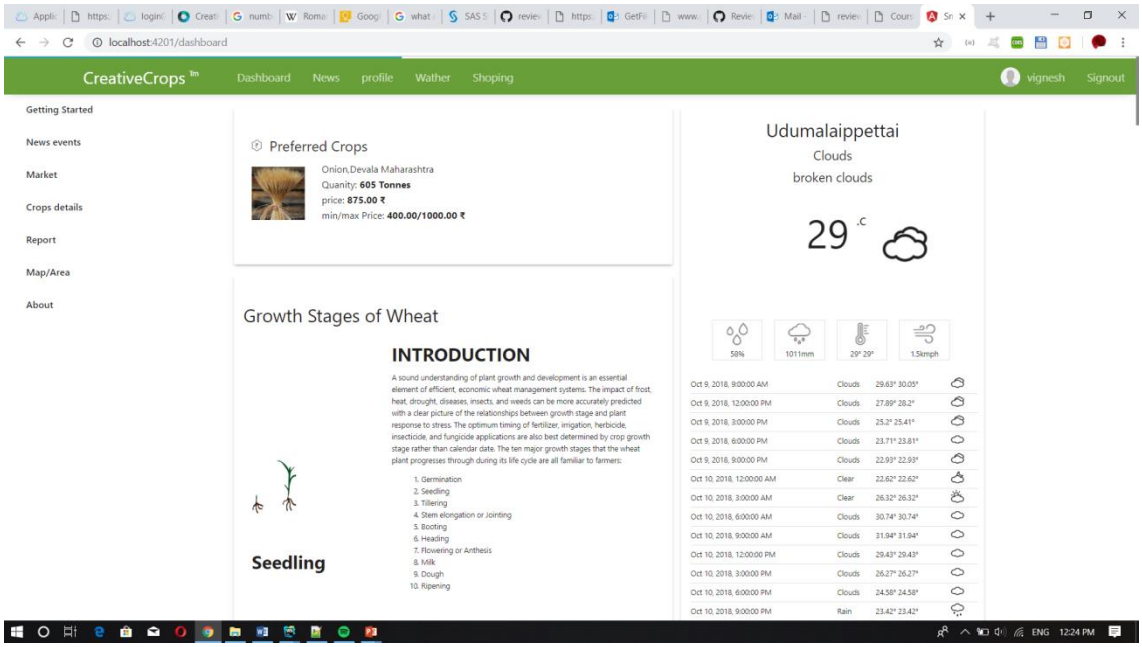

Figure 12 Dashboard for Creative crops