CROP YIELD PREDICTION USING ARTIFICIAL NEURAL NETWORKS

ABHISHEK PARASHAR

Department of Information Technology, Maharaja Agrasen Institute Of Technology, Delhi

Abstract: Agriculture is the basic source of food supply in all the countries of the world whether under-developed, developing or developed. Besides providing food, this sector has contributions to almost every other sector of a country. According to the Bangladesh Bureau of Statistics (BBS), 2017, about 17% of the country's Gross Domestic Product (GDP) is a contribution of the agricultural sector, and it employs more than 45% of the total labor force. In light of the decreasing crop production and shortage of food across the world, one of the crucial criteria of agriculture now-a-days is selecting the right crop for the right piece of land at the right time. Therefore, in our research we have proposed a method which would help suggest the most suitable crop(s) for a specific land based on the analysis of the data of previous years on certain affecting parameters using machine learning. In our work, we have implemented Random Forest Classifier, Gaussian Naïve Bayes, Logistic Regression, Support Vector Machine, k-Nearest Neighbor, and Artificial Neural Network for crop selection. We have trained these algorithms with the training data and later these were tested with test dataset. We then compared the performances of all the tested methods to arrive at the best outcome.

keywords: Agriculture, Crop yield, Logistic Regression, k-Nearest Neighbors, ANN

1.   INTRODUCTION:

For a country, one of the most crucial aspects of its development circles around its capacity to produce food. For decades, agriculture has been associated with the production of essential food crops. The rate of urbanization at present is by-far the most superior aim of our civilization. In doing this, we are ignorantly diminishing our capacity for agriculture; especially in terms of land and fertility. As the amount of land will not be increasing in this era of urbanization and globalization, we will have to focus on making the most of what we have. Due to this issue, we have to device newer ways to farm arable lands and extract the absolute most from these limited land resources. In this age of technology and data-science, if implemented properly, the agricultural sector may also be greatly affected. It is true that a farmer is the best decider of crop selection and crop cultivation. However, machine learning techniques can be applied in this field for far greater precision and stability of selection. In this research, we have attempted to come up with a few techniques that will lead us to choose suitable crops based on specific state, specific district, season, and some other environmental aspects. By analyzing all these issues and problems like weather, temperature and several factors, there is no proper solution and technologies to overcome the situation faced by us. In India there are several ways to increase the economic growth in the field of agriculture. There are multiple ways to increase and improve the crop yield and the quality of the crops. Data mining also useful for predicting the crop yield production.

Data Mining is emerging research field in crop yield analysis. Yield prediction is a very important issue in agricultural. Any farmer is interested in knowing how much yield he is about to expect. In the past, yield prediction was performed by considering farmer's experience on particular field and crop. The yield prediction is a major issue that remains to be solved based on available data. Data mining techniques are the better choice for this purpose. Different Data Mining techniques are used and evaluated in agriculture for estimating the future year's crop production. This research proposes and implements a system to predict crop yield from previous data. This is achieved by applying association rule mining on agriculture data. This research focuses on creation of a prediction model which may be used to future prediction of crop yield. This paper presents a brief analysis of crop yield prediction using data mining technique based on association rules for the selected region. The experimental results show that the proposed work efficiently predict the crop yield production.

2.   IMPLEMENTATION OF PROPOSED METHOD:

2.1 Data Classification

Classification is a data mining function that assigns items in a collection to target categories or classes. The goal of classification is to accurately predict the target class for each case in the data. For example, a classification model could be used to identify a particular crop as to the possibility of its production given a set of features, as per our research. A classification task begins with a data set in which the class assignments are known. For example, a classification model that predicts crop selection could be developed based on observed data for many crop selections over a period of time.

In addition to the crop data, the data might track a number of features. A specific crop would be the target, the other attributes would be the predictors, and the data for each crop would constitute a case. Classifications are discrete and do not imply order. Continuous, floating-point values would indicate a numerical, rather than a categorical, target. A predictive model with a numerical target uses a regression algorithm, not a classification algorithm. The simplest type of classification problem is binary classification. In binary classification, the target attribute has only two possible values: for example, rice or no rice. Multi-class targets have more than two values: for example, low, medium, high, or unknown possibility of rice production. In the model build (training) process, a classification algorithm finds relationships between the values of the predictors and the values of the target. Different classification algorithms use different techniques for finding relationships. These relationships are summarized in a model, which can then be applied to a different data set in which the class assignments are unknown. Multiple pathways can be taken for research of this sort. The field of machine learning is vast enough to accommodate a great number of ways this type of prediction work can be done. For our model, we have implemented all the algorithms and data processing codes using the Python programmable language. The Integrated Development Environment (IDE) that we have used is Anaconda Spyder, during our initial data processing phases. Our model consists of all the aforementioned algorithms which has led us to propose a comparison between all of them, based on performance. There are 8 steps that we have followed in our model:

- Dataset collection.
- Data visualization.
- Data pre-processing in the form of data cleaning and feature extraction.
- Data splitting into train and test sets.
- Fitting the algorithm.
- Parameter tuning (only for Artificial Neural Network).
- Testing the accuracy of the model.
- Data post-processing in the form of performance metrics.

2.2 Data Collection

For research of this sort, it is crucial to have an available dataset to work upon. It is very difficult to find legible and reliable datasets of this sort. It took us a lot of time and effort to find one suitable for us. The dataset that we finally found contained all the accurate features that we really wanted. The features were perfect for research of this sort. There was a total of 12 columns and around 250,000 rows. The columns were "State Name", "District Name", "Crop Year", "Season", "Area", "Rainfall", "Humidity", "Temperature", "Previous Year's Rainfall", "Previous Year's Humidity", "Previous Year's Temperature" and "Crop". 4 of the columns contained data which were in string notation. The rest of the columns contained data which were numerical. The "State Name" column had the names of a number of important states in the country. The "District Name" column had names of districts in those states. The "Crop Year" column had the crop years for the past 19 years. "Season" contained 4 different seasons. "Area" had area data in meters squared. "Area", "Rainfall", "Humidity", "Temperature", "Previous Year's Rainfall", "Previous Year's Humidity", and "Previous Year's Temperature" all contained data respective of the names just mentioned. Finally, the "Crop" column, our most important column, contained 135 different crops which were grown in the places mentioned in the respective columns. All this data is for India, and were taken from the Indian Government Agricultural website upon email request. Research in the field of machine learning, especially deep learning within machine learning, requires heavy usage of computational power. In our research, we had to use Kera's and Tensor flow as a back-end engine which actually supports the Scikit-learn library. Scikit-learn is basically run by Kera's. Furthermore, as our dataset contained a massive number of rows, algorithms such as the Artificial Neural Network algorithm, required high computational power.

2.3 Data Pre-Processing

One of the primary tasks we completed was to convert all of our string data into numerical data. In order to do this, we converted all the string features into dummy variables. This greatly increased our column number. We then cleaned our data in a very singular pattern. We had a small portion of null values in the production column. Due to the miniature amount, we dropped the fields off of the dataset. This did not affect much, as the amount was minimal. We also performed feature extraction on the dataset. Some of our data were categorical and some were continuous numeric. This type of mixed data always causes problems to the algorithms. Hence, we performed standard feature scaling on all of the data to bring them into a common scale. Feature scaling is extremely important due to the fact that, most algorithms feature a lot of internal calculation. Additionally, feature selection was done to reduce over fitting issues.

2.4 Data Splitting

Data splitting is the process of splitting the dataset into training and testing data. This process is very useful for any machine learning process as the main idea of machine learning depends on training and testing data and finding the accuracy of the machine given result. In our research, we divided our dataset because we trained our algorithms on the test dataset where the particular crop had its data in there. Here the algorithms were trained using that data. We deduced

that data having 1 to be yes, and 0 to be no. The algorithms were trained and we will apply the trained algorithm to our test set and measured the accuracy of the machine. The datasets are usually divided into an 80:20 ratio. However, for our model the dataset was split into both 80:20 and 60:40 ratio. Thus, the 80%/60% of the selected data was chosen as training set and the remaining 20%/40% was test set. There are many built in python tool-kits for splitting data such as, 'pandas', 'keras', 'scikit-learn' etc. although we used "scikit-learn" for the machine learning approaches in this research because of its built-in libraries.

2.5 Algorithm Fitting

The most crucial part of the model was to fit the algorithm with the data. All the algorithms were easily fitted as the programming of this part was comparatively easy. Simple method callings were all that were required. The algorithms, upon being implemented, processed all the data using all the internal calculations. Data frames were created and could be viewed in the variable explorer. Because of the fact that the data had been split into training and testing datasets, the algorithm could start the core process: learning. The machine learned from the train set. This learning was to be used later on while predicting from the test set. Fitting is similar to training. For example, let us assume that we have measured the production for a group of crops and decided that your model would be a normal distribution. Then determining the mean and variance of the normal distribution that best explains our observed data is called fitting: we are determining the parameters' mean μ and variance σ. Suppose we have an algorithm that estimates μ and σ given our data. We can ask our algorithm to run "n" iterations where each iteration takes the same amount of time but more iterations yield slightly more accurate estimates of the parameters μ and σ. "n" is a value we supply that may influence the estimates and is called a hyper-parameter.

2.6 Testing Accuracy

To test the accuracy, we implemented different methods on different algorithms based on requirement. Some were direct accuracy-check method calls from scikit-learn libraries. While in some other algorithms, we implemented manual accuracy checks, again based on the algorithm itself. In Random Forest of instance, mean was calculated. In Artificial Neural Network, despite calling an accuracy-check method, all the accuracy from all the epochs were taken in for mean value of accuracy. The accuracy check is crucial in understanding the viability of the algorithms and also the research itself. A very low accuracy in all the algorithms would mean the entire research was a dead end. It would mean this method altogether is not viable for this research. A low accuracy in a few algorithms and a high accuracy in the others would mean the ones with the low accuracy are not efficient in this model, but the others are. We would have discarded the low accuracy yielding algorithms. However, in our case, all the algorithms yielded a very high accuracy. Although this issue did cause us a few problems later on when comparing the accuracy amongst all the algorithms, the fact that the accuracy is high on all, was a strong point in determining the methods to be viable in this field of research. An Effective Model is a model which basically predicts the testing data most accurately as compared to other models and hence, can be deployed successfully. Testing accuracy of k-NN is shown

2.7 Data Post-Processing

After all the accuracy have been taken into account, a few other data processes can still be implemented. This part of the model is not necessary for the primary target of the research, but we still used it for certain confirmation purposes. We implemented a method which would create a confusion matrix. A confusion matrix is a technique for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading if we have an unequal number of observations in each class or if we have more than two classes in our dataset. Calculating a confusion matrix can give us a better idea of what our classification model is getting right and what types of errors it is making. Figure 3.2 shows the confusion matrix we got by implementing Logistic Regression for crop prediction.

```
[[   0    0    0 ...     0    0    0]
 [   0   33    0 ...     0    0    0]
 [   0    0   57 ...     0    0    0]
 ...
 [   0    0    0 ...   152    0    0]
 [   2    0    0 ...     0   54    0]
 [   0    5    0 ...     0    0    0]]
```
Figure 3.2 Confusion Matrix for Crop Prediction

The methods to reduce any over fitting issues that were implemented earlier in the data pre-processing stage, can even be implemented here instead the other stage. However, it was extremely necessary for us to use those in the model, any form of over fitting or under fitting would actually render the whole model lack of any real use.

2.8 Artificial Neural Network (ANN)

We followed standard procedures for all the algorithms apart from the Artificial Neural Network (ANN). Starting from the data pre-processing, to algorithm fitting, to accuracy checking, and finally predicting the desired outcome, the methods and codes used nearly remained the same. The only major variation was in the part where we called the algorithm fitting functions. However, the deep learning model that we created for the ANN,

was different to the other algorithms to a certain extent. We created the hidden layers and perceptions manually. Even after the predictions were being generated, we performed further checks in the form of k-Fold Cross Validation and Grid Search CV to ensure the highest viability of the accuracy obtained. The first step was to calculate the activation of one neuron given an input. The input was a row from our training dataset, as in the case of the hidden layer. It might also be the outputs from each neuron in the hidden layer, in the case of the output layer. Neuron activation was calculated as the weighted sum of the inputs, much like linear regression.

$$activation = sum (weight\_i * input\_i) + bias$$

Where weight was a network weight, "i" was the index of a weight or an input and bias was a special weight that had no input to multiply with. Once a neuron was activated, we needed to transfer the activation to see what the neuron output actually was. Different transfer functions could be used. It was traditional to use the sigmoid activation function, but we could also use the tan h (hyperbolic tangent) function to transfer outputs. However, due to the recent popularity and increase in efficiency, the rectifier transfer function had been used by us in our deep learning model. The sigmoid activation function looks like an S shape; it is also called the logistic function. It can take any input value and produce a number between 0 and 1 on an S-curve. It is also a function of which we can easily calculate the derivative (slope) that we use later when back propagating error.

The first step was to calculate the error for each output neuron, that gave us our error signal (input) to propagate backwards through the network. The error for a given neuron was calculated as follows.

$$error = (expected - output) * transfer\ derivative\ (output)$$

Where "expected" was the expected output value for the neuron, "output" was the output value for the neuron and transfer_derivative() calculated the slope of the neuron's output value, as shown above. This error calculation was used for neurons in the output layer. The expected value was the class value itself. In the hidden layer, things are a little more complicated. The error signal for a neuron in the hidden layer was calculated as the weighted error of each neuron in the output layer. The error traveled back along the weights of the output layer to the neurons in the hidden layer. The back-propagated error signal was accumulated and then used to determine the error for the neuron in the hidden layer, as follows.

$$error = (weight\_k * error\_j) * transfer\_derivative(output)$$

Where "error_j" was the error signal from the "j"th neuron in the output layer, "weight_k" was the weight that connects the "k"th neuron to the current neuron and output was the output for the current neuron. The network was trained using stochastic gradient descent. This involves multiple iterations of exposing a training dataset to the network and for each row of data forward propagating the inputs, back propagating the error and updating the network weights. Once errors were calculated for each neuron in the network via the back-propagation method above, they could be used to update weights. Network weights were updated as follows.

$$weight = weight + learning\_rate * error * input$$

Where "weight" was a given weight, "learning_rate" was a parameter that we specify, "error" was the error calculated by the back-propagation procedure for the neuron and "input" was the input value that caused the error. The same procedure might have been used for updating the bias weight, except there was no input term, or input was the fixed value of 1.0. Learning rate controls how much to change the weight to correct for the error. For example, a value of 0.1 will update the weight 10% of the amount that it possibly could be updated. Small learning rates are preferred that cause slower learning over a large number of training iterations. This increases the likelihood of the network finding a good set of weights across all layers rather than the fastest set of weights that minimize error. If errors were accumulated across an epoch before updating the weights, this would be called batch learning or batch gradient descent. Once all the aforementioned steps were completed, the network became eligible for training. The rest of the procedure followed a similar structure to the rest of the algorithms, as discussed above.

2.9 Algorithms for Proposed Model
2.9.1 Logistic Regression

Methods involving regression are essential to any data analysis models which attempt to describe the association between a response variable and any number of predictor variables. Situations involving discrete variables constantly arise. For

instance, the dataset we have implemented has an outcome involving the presence or absence of a particular crop, given a set of features. Logistic regression analysis extends the techniques of multiple regression analysis to investigate and inquire situations in which the outcome is categorical, which is, taking on multiple values. This is a very basic branch of data science. Although the name suggests a regression technique, logistic regression is a statistical classification model which deals with categorical dependent variables. Classification is decision. To make an optima decision we need to assess the utility function, which implies that we need to account for the uncertainty in the outcome, i.e., a probability. Logistic regression is emphatically not a classification algorithm on its own. It is only a classification algorithm in combination with a decision rule that makes dichotomous the predicted probabilities of the outcome. This is one of the very first algorithm any machine learning practitioner attempts when faced with a classification problem. The basic mechanism and output of this algorithm is similar to many other machine learning algorithms. It is the appropriate regression analysis to conduct when the dependent variable is dichotomous or binary. Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. This algorithm works with binary data, where either the event happens, represented by "1", or the event does not happen, represented by 0. So given some feature "X", it tries to find out whether some event "y" happens or not. So "y" can either be "0" or "1". In the case where the event happens, "y" is given the value "1". If the event does not happen, then "y" is given the value of "0". For example, if "y" represents whether a particular crop among a huge variety of crops, then "y" will be "1" if the crop does grow or "y" will be "0" if it does not. This is known as Binomial Logistic Regression. There is also another form of Logistic Regression which uses multiple values for the variable "y". This form of Logistic Regression is known as Multinomial Logistic Regression. Figure 3.3 shows a simple flow chart representation of logistic regression.
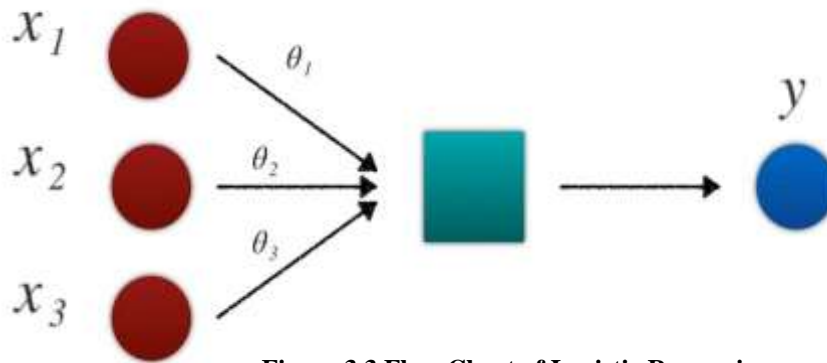

Figure 3.3 Flow Chart of Logistic Regression

Logistic Regression uses the logistic function to find a model that fits with the data points. The function gives an "S" shaped curve to model the data. The curve is restricted between "0" and "1", so it is easy to apply when "y" is binary. Logistic Regression can then model events better than linear regression, as it shows the probability for "y" being "1" for a given "x" value. Logistic Regression is used in statistics and machine learning to predict values of an input from previous test data. Scatter plot classification of data by Logistic Regression is shown in Figure 3.4
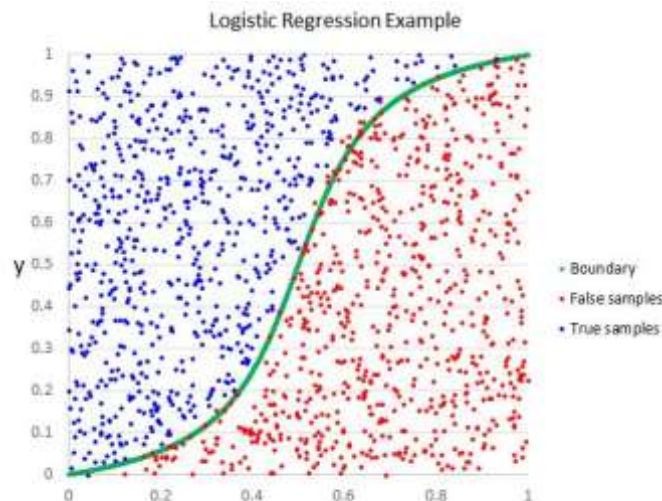

Figure 3.4 classification of data by Logistic Regression.

3.9.2 Support Vector Machine (SVM)

"Support Vector Machine" (SVM) is a supervised machine learning algorithm which is a very useful technique for data classification. However, this learning algorithm can also be used for regression challenges. A classification task usually involves separating data into training and testing sets. Each instance in the training set contains one "target value" (i.e., the class labels) and several "attributes" (i.e., the features or observed variables). The goal of SVM is to produce a model (based on the training data) which predicts the target values of the test data given only the test data attributes.

Linear SVM

Support Vector Machine classifier plots each data item with the value of each feature as a point in an n-dimensional space (where n is number of features) being the value of a particular coordinate. SVM maps data to a high-dimensional feature space so that data points can be categorized, even when the data are not otherwise linearly separable. Then, it performs classification by finding the hyper-plane that differentiates the two classes very well. points can be categorized, even when the data are not otherwise linearly separable. Then, it performs classification by finding the hyper-plane that differentiates the two classes very well.

When the data can be linearly separated in two dimensions, as shown in Figure 3.5, any machine learning algorithm tries to find a boundary that divides the data in such a way that the mis-classification error can be minimized. Nevertheless, there can be several boundaries that correctly divide the data points as shown below in Figure 3.6.
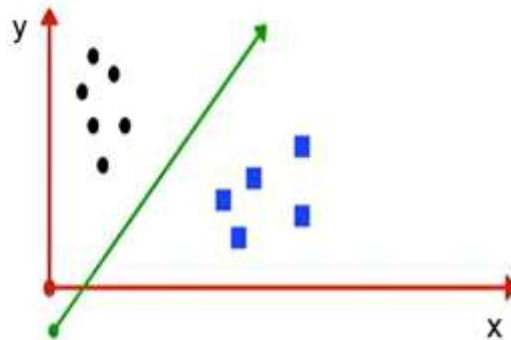

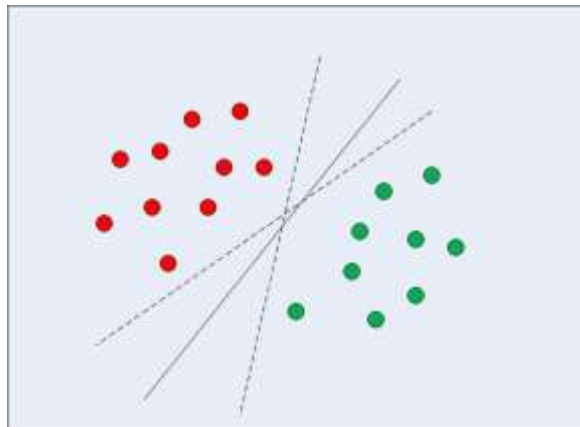**Figure 3.5 Classifications of Two Classes Using Boundary.**

Figure 3.6 Multiple Decision Boundaries.

SVM is different from the other classifiers in the way that it chooses the decision boundary that maximizes the distance from the nearest data points of all the classes. This boundary has the maximum margin from the nearest points of the training class as well as the test class. As a result, SVM classifier does not only find a boundary; it finds the most optimal decision boundary. This boundary resulting from SVM is called the maximum margin classifier, or the maximum margin hyper plane. The nearest points from the hyper plane that maximize the distance between the decision boundaries are called support vectors.

3.9.3 K-Nearest Neighbours

K-Nearest Neighbors (KNN) algorithm is one of the simplest, easy to understand, versatile and one of the topmost machine learning algorithms. KNN is a non-parametric supervised learning algorithm. Additionally, it is an instance-based learning or a lazy algorithm. When a query to the database is made, the algorithm uses the training instances to spit out an answer. That is why, for KNN the training phase is very fast compared to other classifier algorithms. However, the testing phase becomes slower and costlier, that is in terms of time and memory. KNNs purpose is to use a database in which the data points are separated into several classes to predict the classification of a new sample point. Informally, this means that for a labeled dataset consisting of training observations (x, y) and the algorithm is used to capture the relationship between x and y. KNN Algorithm is based on feature similarity: How closely out-of-sample features resemble our training set determines how we classify a given data point. An example of KNN classification is shown below in Figure 3.7.
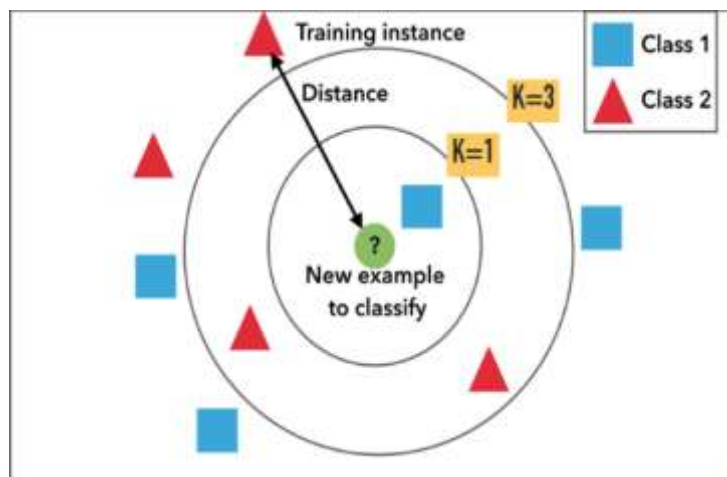

Figure 3.7 K-Nearest Neighbours Classification

Here, the test sample (inside circle) has to be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 (outside circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If, k = 5 it is assigned to the first class (3 squares vs. 2 triangles outside the outer circle). In KNN, K is considered as the number of the nearest neighbours. The number of neighbours is the core deciding factor for the classification. The algorithm is known as the nearest neighbour algorithm when K=1. In the classification setting, the K-nearest neighbour algorithm essentially boils down to forming a majority vote between the K most similar instances to a given "unseen" observation. Similarity is defined according to a distance metric between two data points. The "Euclidean distance" is a popular choice. But other measures can be better suited for a particular setting that includes the distance Manhattan, Minkowski and Hamming. The equations are shown in the Figure 3.8 below:

$$\text{Euclidean} \quad \sqrt{\sum_{i=1}^{k}\left(x_i - y_i\right)^2}$$

$$\text{Manhattan} \quad \sum_{i=1}^{k}\left|x_i - y_i\right|$$

$$\text{Minkowski} \quad \left(\sum_{i=1}^{k}\left(\left|x_i - y_i\right|\right)^q\right)^{1/q}$$

Figure 3.8 Formula for Distance Metric Calculation.

More formally, given a positive integer K, an unseen observation x and a similarity metric d, KNN classifier performs the following three steps:

• It runs through the whole dataset computing d between x and each training observation.   We'll call the K points in the training data that are closest to x the set A. Note that K is usually odd to prevent tie situations.

• It then estimates the conditional probability for each class, that is, the fraction of points in A with that given class label.

• Finally, the input x is assigned to the class with the largest probability.

The number of neighbors (K) is a hyper-parameter that needs to be chosen at the time of model building. Research has shown that there is no optimal number of neighbors which suits all kind of data sets. Each dataset is different and needs to fulfill its own requirements. In most cases, it is better to choose it as an odd number if the number of classes is even. When the value of K is small, the region of a given prediction is being restrained. And the classifier is being forced to be "more blind" to the overall distribution. A small value for K provides the most flexible fit, which will have low bias but high variance. Graphically, the decision boundary will be more jagged, which is shown in the Figure 3.9 below. On the other hand, a higher valued K averages more voters in each prediction. Hence it is more resilient to outliers. Larger values of K will have smoother decision boundaries which mean lower variance but increased bias, shown below in the Figure 3.10 below.
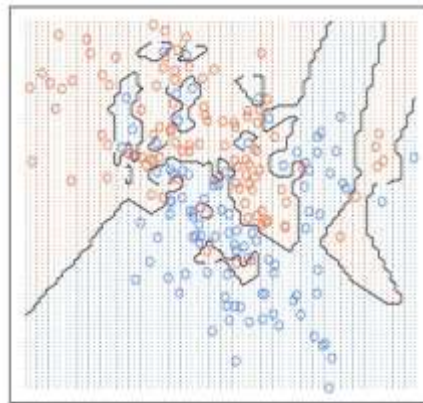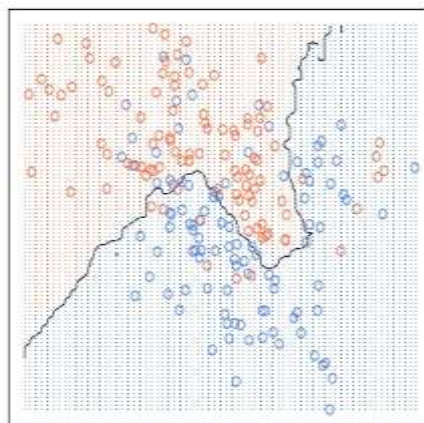

Figure 3.9 Data Classification When K=1

Figure 3.10 Data Classification When K = 20.

This graph clarifies that at K=1, the boundaries were being over-fitted. So, the error rate initially decreases and reaches minima. After the minima point, it then increases with increasing K. To get the optimal value of K, the training and validation can be segregated from the initial dataset. Then by plotting the validation error curve, it is easier to get the optimal value of K. This value of K should be used for all predictions.

3.   EXPERIMENTAL RESULTS:

5.1 Result Analysis:

The core aim of our research was to establish a model that will efficiently predict a particular crop based on a set of features. During the course of this research, we have also successfully created a model that can predict the possibility of

a particular crop to be able to be grown, given a set of features. In order to do this, as mentioned above, we have implemented 6different machine learning algorithms. Amongst them, The SVM was done using 3 different aspects, and the KNN was done using 2. This ensured us the ability to bring a comparison between varieties of different algorithms. Our target was to establish the best performing algorithm for this field of work, based on our data.

5.2 Accuracy Analysis:

As mentioned before, we have extracted accuracy from 9 different processes. Only the ANN was run on one instance only. Apart from that, all the other algorithms were tested on a variety of instances of the same dataset. We took 5 samples of our primary dataset. The samples had an increasing number of rows starting from 2000 and ending at 11000. We ran each algorithm on these samples in 2 separate train test splits. Initially, we used the 60%-40%train test ratio. Eventually, we changed that to 80%:20%. Both gave us fair results, but the latter gave us a better accuracy, which led us to finalize the model on that. Two separate outcomes were extracted from our algorithms with slight tweaking. We could both predict a particular crop and the percentage chance of a specific crop to be able to be

grown, given a particular set of features. However, the former method fetched bad accuracy levels, which we found highly unconvincing. The highest accuracy we got was from the KNN (K=optimal) algorithm, which was 81.3%, on a set of 11000 features. In similar features, but with a specific crop as the dependent variable, all the algorithms gave strong accuracy levels. The lowest accuracy was given by Random Forest Classifier which was still 92.30%. The highest accuracy was again given by the KNN (K=optimal) algorithm. The ANN was tested only once with 11000 features, and only for the specific crop model. We trained and tested the network up to 1000 epochs. The accuracy that we obtained was 96.95%. All the accuracy levels in this part of the thesis indicated to a strong viability of our research in this field. In the (Table. 3.1) and (Table. 3.2) below we have shown the comparison among all the classifiers we have implemented for specific crop possibility prediction and crop prediction respectively.

Table 3.1 Accuracy comparison for crop possibility

Data	GNB	SVM	Linear SVM	RBF SVM	KNN (k=1)	KNN (k=optimal)	LR	RF	ANN
2000	80.61	95.07	92.97	94.38	90.44	95.50	95.22	95.07	
4000	84.26	94.95	96.35	94.53	93.26	95.65	96.07	95.16	
6000	89.04	97.28	97.09	97.84	96.16	96.44	95.78	95.78	
8000	85.36	95.27	95.59	95.74	95.59	96.81	95.31	93.71	
11000	95.5	97.0	97.2	96.23	96.38	97.70	97.38	92.30	96.95

Table 3.2 Accuracy comparison for crop prediction

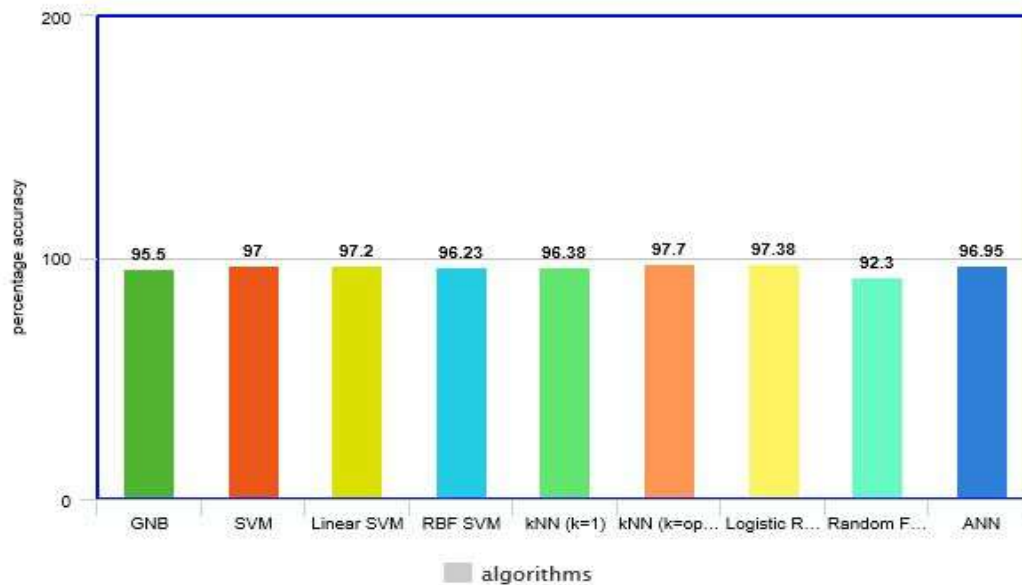| Data	GNB	SVM	Linear SVM	RBF SVM	KNN (k=1)	KNN (k=optimal)	LR	RF
2000	70.18	70.50	65.44	67.13	65.44	66.01	62.58	30.04
4000	70/88	49.92	49.64	58.34	38.84	48.80	57.50	44.37
6000	55.97	63.12	67.50	62.92	54.02	56.46	52.52	45.41
8000	58.79	63.52	58.90	64.79	57.33	60.0	43.75	45.84
11000	56.2	63.7	56.8	68.8	61.38	81.3	66.46	55.0

Figure 3.11 Graphical representation of accuracy among all classifier's comparison for crop prediction.

4.   CONCLUSION:

We believe this model can play a very essential part in today's world. Agriculture is a fundamental aspect of modern civilization. With increasing world hunger and economy breakdown, the proper selection of crop emerges as a massive factor in this. Our proposed model can predict the proper crop for a particular piece of land in a way that is very efficient. We have implemented 6 different types of machine learning and deep learning algorithms in this research. All the accuracy of the models was carefully obtained through various different methods, and compared with each other. Using multiple algorithms helped to understand which algorithm is more suitable for this system. The crops can be predicted based on a very suitable set of features included in the dataset used. From this research work, we found that, the cleaner the data, the better the accuracy of the result. The entire length of this research was very enjoyable, as we were able to work in the field of machine learning and deep learning. Some of the python library usage, algorithm fitting, and accuracy checking methods were very interesting in practicality. We trust all our algorithms and research work to efficiently work on any platform and any new type of data. The predictions made were solid and robust. Such strength in the model delights us, and we hope this keeps working over the years without issues.

REFERENCES

[1]. Aggarwal Sachin (2001). Application of Neural Network to Forecast Air Quality Index. Thesis submitted in partial fulfillment of requirements for a degree in Bachelor of Technology, April 2001.

[2].   B. J I ET AL Artificial neural networks for rice yield prediction in mountainous regions. Journal of Agricultural Science (2007), 145, 249–261.

[3].   B.A. Smith et al Artificial Neural Networks for Automated Year-round Temperature Prediction. Computers and Electronics in Agriculture 68 (2009) 52–6.

[4].   Cheng, B. and Titterington. D. M. (1994). Neural networks: A review from a statistical perspective Statistical Science, 9: 2-54.

[5].   D.L. Ehret et al, Neural network modeling of greenhouse tomato yield, growth and water use from automated crop monitoring data. Computers and Electronics in Agriculture 79 (2011) 82–89.

[6].   Enfield, D. B., 1996. Relationships of inter-American rainfall to tropical Atlantic and Pacific SST variability. Geophysical Research Letters 23(23): 3305-3308.

[7].   Everingham, Y. L., R. C. Muchow, R. C. Stone, and D. H. Coomans, 2003. Using southern oscillation index phases to forecast sugarcane yields: a case study for Northeastern Australia. International Journal of Climatology 23(10): 1211-1218.

[8].   Handler, P, 1990. USA corn yields, the El Niño and agricultural drought: 1867-1988. International Journal of Climatology 10(8): 819-828.

[9].   Hansen, J. W., J. W. Jones, C. F. Kiker, A. W. Hodges, 1999. El Niño-Southern Oscillation impacts on winter vegetable production in Florida. Journal of Climate 92-102.

[10]. Hansen, J. W., A. W. Hodges, and J. W. Jones, 1998. ENSO Influences on agriculture in the southeastern United States. Journal of Climate 11(3): 404-411.

[11]. Haykin. S, 1999. Neural Networks: A Comprehensive Foundation (Second Edition). Upper Saddle River, NJ: Prentice Hall.

[12]. Izaurralde, R. C., N. J. Rosenberg, R. A. Brown, D. M. Legler, M. T. Lopez, R. Srinivasan, 1999. Modeled effects of moderate and strong, Los Niños" on crop productivity in North America.